Achieving Cost-Effective Testing for Serverless Based Applications

A thematic analysis on practitioner's experiences and opinions

Bachelor of Science Thesis in Software Engineering and Management

Edvin Danielsson

Gregory Sastrawidjaya

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023

A thematic analysis on practitioner experiences and opinions with cost-effective testing on serverless-based applications.

Supervisor: Gregory Gay
Examiner: Fransisco Gomes

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
University of Gothenburg
Chalmers University of Technology
Gothenburg, Sweden 2023

Achieving Cost-Effective Testing For Serverless-Based Applications

Edvin Danielsson
Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
gusdanieed@student.gu.se

Gregory Sastrawidjaya
Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
gussasgr@student.gu.se

***Abstract*—Testing serverless applications poses unique challenges compared to traditional applications, such as the abstraction of its inner workings and how the pay-per-usage model means tests incur monetary costs. With the goal of studying practitioner's approaches on this topic of cost-efficiency, we conducted five interviews and distributed a survey which received six responses. The questions were around testing practices, cost factors, and evaluation of cost-efficiency. After an inductive thematic analysis, the results revealed that the importance of unit tests increases as the development of serverless applications progresses. Participants emphasized the significance of monitoring metrics, particularly response time, to ensure optimal performance and validity. To mitigate costs, they employed strategies such as the incorporation of observability tools, granularity, wherein they reduced the size and scope of functions, and utilized local emulation for smaller tasks. Additionally, the choice of serverless provider emerged as a topic of discussion, as participants faced challenges in smoothly transitioning between vendors due to the varying advantages offered by each provider.**

These results provide insight into the challenges of achieving cost-effective testing for serverless-based applications — though we believe the topic should receive further attention in research due to how new this field is.

I. Introduction

Serverless computing is a growing global trend within the IT industry [10]. This trend entails the deployment of independent, stateless functions on a cloud provider which hosts the infrastructure needed for these functions to execute. This abstracts away the need to provision and maintain a server, and rewards the developer with consistent uptime, scalable services, quick deployments, and pay-per-usage costs [7]. Known public cloud providers such as Amazon (AWS), Google (Google Cloud Functions), and Microsoft (Azure) provide the core functionality of different backend services such as cloud storage, authentication, and cloud computing [16].

However, a challenge of serverless applications stems from the architectural design, the abstractions, as well as its pay-per-use model, directly correlating to difficulties when performing traditional testing practices in traditional applications based on the monolithic and microservice-based architectures.

For instance, serverless-based applications are a step above in terms of granularity, which directly impacts the testing of the entire application. Because monolithic applications are simple and all dependencies reside in one executable, testing and debugging can be done locally before deployment [9]. In microservices, the monolithic architecture is decomposed even further in which each part of the monolithic application becomes cut down into its own service. An individual microservice can then be considered as a small monolithic system delegated for a single purpose, e.g. authentication, data processing, etc. These small microservices can then be tested locally using well-established unit-testing strategies and criteria such as branch coverage and mutation testing [9]. However, when it comes to serverless applications, they are way more granular than either of the two. Instead of small monolithic applications with delegated responsibilities, i.e. microservices, this is then decomposed even further into independent stateless functions. These stateless functions interact with each other through events. These events can be diverse in nature, such as client requests, events produced by external systems, data streams, or even complex rules describing a combination of all of the above [8].

Testing a serverless-based application can prove to be difficult and sometimes impossible to test and debug because of the contexts these events can occur [19], resulting in a system with event-based dependencies that becomes available and observable at run-time [9] [8]. Leitner et al. suggests the possibility of testing the serverless application directly in production or in different test environments in the cloud to get around this issue [9] [8]. However, the pay-per-use model of serverless results in having to pay for test invocations towards these functions in these environments [9] [8].

Leitner et al. suggests another possibility of replicating or emulating the test environment in order to perform tests locally, but this proves to be difficult because the internal implementation of these serverless solutions are abstracted away from the developer. Visibility is limited and there is no introspection into the host computer's processes [8] [2]. No information related to the runtime environment of the underlying serverless platform is given from the cloud provider [8] [2]. Since the cloud vendor manages the infrastructure required to run the code, developers do not have direct control over the internal conditions in which the code is actually executed.

Lenarduzzi and Panichella touch upon different ways of approaching this inherent unpredictability of serverless applications through the potential usage of structured and centralised logging frameworks, as well as automated test generation. Though industry experts have acknowledged these testing techniques to be useful and valid for debugging serverless applications, a point of contention remains as to what method is generally favorable for time and cost-effectiveness [9].

In addition to the difficulty of testing serverless-based applications without incurring costs, there is a mentioned lack of ways to evaluate test practices towards these types of applications. Winzinger mentions this, stating that although there is coverage criteria available for traditional applications, there is a lack of coverage criteria focusing on serverless-based applications [16].

By the same token, an exploratory case study done recently by Eismann et al. over the span of ten months highlighted that performance was a potential issue in serverless applications, as the set-up time for a serverless application's environment to get up and running can vary based off certain use cases [7]. After performing certain metrics to measure performance impacts, the result provided solidifies the fact that this performance varies from vendor to vendor.

From the findings of these studies [8] [9] [7] [16] [2], testing such isolated functions proves to be difficult and time consuming, resulting in less productivity towards the development of the actual application.

Therefore, the purpose of this thesis is two-fold. First, this is a relatively new area of software development with several distinct challenges. Therefore, we must explore how testing of such code is done in the industry. Second, testing is time consuming, and cost is very important in this area, so we explore the factors that impact cost and how testing has been made cost-efficient in industry.

For our methodology, we interviewed 5 practitioners about their testing experiences with serverless systems. Our questions focused on how they use and create tests, and strategies to remain cost-efficient. We performed a thematic analysis on the data as well as triangulated by also sharing a survey within developer networks, of which we had 6 responses.

The results showed the differences in testing practices from practitioner to practitioner, in which unit tests began growing in importance the more complex the serverless-based application became. The participants highlighted the importance of metrics and observing them in order to evaluate cost-efficiency of their testing practices. Key themes were mentioned such as local emulation, which was incorporated in order to reduce costs incurred in the cloud. Additionally, granularity was mentioned as a design consideration in order to fine-tune resource allocation of these serverless functions to positively impact cost-efficiency. There was also discussion around the choice of serverless provider, with the issue of not being able to migrate effectively from one vendor to another, as some providers offer advantages and features in their products that other providers might not.

We aim for the thesis to contribute towards enterprises and practitioners within the IT industry that plan to employ or currently employ serverless components within their software systems. First, we intend for business owners to use our provided data to help them strategize around any cost-inefficiencies before they occur, thereby reducing budget expenditure and developer time spent. Furthermore, we aim to contribute to practitioners' workflow by streamlining the deployment of their code, through the usage of cost-efficient testing methods from our gathered data. Since serverless is also a growing trend the thesis will grant other researchers data to build upon, so as to further optimize testing within serverless-based applications.

II. Background

A. Serverless-Based Applications

A serverless-based application is an application that contains business and domain specific logic that leverage the usage of cloud provided solutions, namely FaaS [14]. The business and domain specific logic are split into stateless and isolated functions that are then deployed through the cloud provider's FaaS implementation.

Cloud providers allow the end user to purchase additional resources to provision this container based off of use cases and demand, such as RAM or additional CPU cores. This execution model ensures that the application becomes highly available and scalable due to this on-demand provisioning. Once the serverless function is triggered, it is instantiated in its own separate virtual container. This requires an initial startup time if the function has not been invoked for a certain time period, generally around the ballpark of several minutes. This initial startup time is known as a *cold start*.

B. Cloud Computing and What It Entails

According to NIST (US National Institute of Standards and Technology), the definition of cloud computing is:

> "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." - NIST [11]

These computing resources are pay-on-demand/pay-per-usage, and such solutions span from container storage, messaging queues, data streams, machine learning, API gateways, and authentication provider implementations to name several examples [7]. In general, these computing resources are commonly classified into four different types of services [11]:

SaaS - Software as a Service: Software as a service (SaaS) is a cloud computing model that is based around the cloud provider providing a piece of software that can be used over an internet connection. The cloud provider would then be responsible for the maintenance and operation of the infrastructure behind the software itself.

PaaS - Platform as a Service: Platform as a service (PaaS) is a cloud computing model that is based around the cloud provider providing a virtual platform with the underlying infrastructure already built in for the purposes of deploying applications. The user would then have full control of the application within this platform, such as hosting, building, and deployment of the application itself.

IaaS - Infrastructure as a Service: The simplest and lowest level form of cloud computing comes in the form of Infrastructure as a service (IaaS), which is a cloud computing model that delivers infrastructure such as storage and computing power in the form of virtual machines. Through the infrastructure provided, developers and administrators are able to then create platforms and delegated services for the application. Furthermore, the user of the IaaS solution would be able to manage the runtime environment and underlying application runtime configurations.

FaaS - Function as a Service: FaaS is a cloud computing model which provides a centralised platform for developers to write and run independent snippets of code that is then run on the cloud vendor's hardware. Isolated functions are written as business logic, which are then configured to trigger based off of an event. These events could range from anything such as an HTTP request, or even a database read/write operation [7]. As long as the code works, the cloud provider guarantees it will execute. As mentioned in the previous section, the upside to this execution model is that maintenance and provisioning of resources is abstracted away from the developer.

The main focus of this paper leans towards providing insight into cost-effective ways to test serverless applications incorporating FaaS.

C. Cloud Computing Deployment Options

Although there are different models of cloud computing, there are different ways as to how cloud computing services are deployed and served to customers.

These deployment options are commonly classified as public and private. Public cloud resources are maintained and managed by a third-party which is then served over the internet for public access. All infrastructure, such as hardware and software, and the business logic behind the cloud resources are solely owned by the cloud provider [11]. Examples would be the serverless providers listed in the Introduction of this thesis.

Compared to private cloud resources which are only accessible for usage by one business or organisation. All infrastructure is maintained on a private network. By doing so, resources can be open for customisation while fulfilling specific business privacy requirements [11]. A reason private cloud deployments exist is because it is suitable for enterprises operating under certain privacy regulations such as GDPR [4].

This thesis focuses more on serverless applications incorporating cloud computing on a public cloud deployment, as it is accessible and the most popular choice of deployment today [1].

III. Related Work

A. Current Testing Practices In Serverless Applications

Numerous industry experts appear to prioritize different testing practices when developing serverless applications. Yan Cui, an industry expert focused on serverless application and architectures, advocates reversing the testing pyramid to favor testing that contributes towards integration of the entire system, rather than the traditional approach of testing the application as a unit via individual code snippets [9]. However, other companies such as Epsagon and Thundra, both cloud based monitoring solutions for serverless microservices, adopt various levels of testing [9]. For instance, Epsagon seeks a balanced approach by developing test cases to cover possible events and values their serverless functions might return synchronously and asynchronously [9]. On the other hand, Thundra has a heavy reliance on unit testing of the different components within their system. More specifically, they focus on writing test scenarios at the requirement gathering stages of software production which are then broken down into individual unit tests [9]. All industry practitioners unanimously acknowledged the key importance of a centralised logging system, as limited observability remains a common pitfall within serverless systems [9].

B. Testing The Performance of Serverless Applications

The performance of serverless applications have been tested over the years, which remains as a direct issue within serverless applications due to cold starts [8]. A study performed by Scheuner et al. evaluated the performance of serverless applications through the deployment and invocation of a thumbnail generator which consisted of two small serverless functions. These functions were invoked from an asynchronous storage bucket event trigger, and then timed for benchmarking purposes [15]. Benchmarks were critiqued against twelve criteria such as similar runtimes and resources to ensure fairness within the evaluation [15], as the experiment was overseen on two popular cloud platforms - AWS (Amazon Web Services), and Microsoft Azure. Scheuner et al. discovered that the benchmarks varied from provider to provider, and that the implementation design of both cloud providers were inherently different, namely the multi-tenancy focus on services in AWS versus the stronger per-tenant isolation within Azure [15]. AWS experienced higher-percentile latency common in multi-tenancy focused systems when running functions in bursts [20], whereas running the same workload on Azure proved to be relatively predictable yet slower due to the in-process function scheduling done internally on Azure's infrastructure [15]. The study proves that different serverless vendors have different internal runtimes and that depending on the workload, performance can fluctuate unpredictably [15]. However, cost-efficiency was not touched upon during this study, as one of the criterion in the experiment was to use subscription tiers that consisted of similar resource delegation during method invocation. No comparison was done between the billing time of one cloud vendor to the other. These studies compared the execution and runtime performance of a small serverless-based application on AWS and Azure. Since serverless functions are charged on a pay-per-invocation basis that includes duration and time spent running the actual function, these studies could serve to clarify potentially discovered cost-efficient testing methodologies that depend on the cloud vendor, as it is clear namely from Scheuner et al. that different cloud providers have different underlying infrastructures [15].

C. Coverage Criteria For Serverless Applications

In serverless applications, integration tests are prominent due to the component's complexities being difficult to test in isolation. Measuring the quality of tests requires coverage criteria [16], which refers to the metrics used to evaluate the extent to which a software application has been tested in accordance with testing practices.

Winzinger et al., in their 2019 papers [17], [18], created their own coverage criteria based upon a classical testing approach that models functions as nodes interconnected on a graph to describe control flow of business logic. Winzinger et al. proposed three main coverage criteria to evaluate testing practices for serverless applications, which were then tested. They noted that AWS does not really support the evaluation of the coverage of this criteria very well. Below are the coverage criteria that were developed:

All-resources is when every serverless function is executed at least once during the execution of the problem. requires coverage of not only the serverless function, but other resources provided by the cloud platform provider. These resources, such as hardware specifications and performance metrics, i.e. latency and throughput, are unfortunately private information.

All-resource-relations is when every serverless function and its interactions with one another is tested, i.e. all edges between the nodes are covered. Winzinger and Wirtz state that this requires an application that is built with observability in mind. What this means is that when a serverless function is called, there needs to be information on the source of this function call.

All-resource-sequences is when every possible execution sequence of these serverless functions is tested. Winzinger and Wirtz states that it is always possible to recreate the sequence when a serverless function is being called. They suggest passing in a unique identifier that contains logs of the sequence taken. However, they also note that identifying all feasible paths (possible execution sequences) is not always possible due to how serverless applications are highly granular, consisting of tasks broken down into small functions. Therefore, in order to take this consideration into account, Winzinger and Wirtz suggest that worst case assumptions can be made for the identification of these paths [19].

After discovering these coverage criterion for the purposes of testing serverless applications, Winzinger and Wirtz test the applicability of this coverage criteria in a follow-up study, and note that not all of this criteria is practically relevant and feasible. For instance, in the case of *All-resource-relations*, Winzinger and Wirtz states that if a function call triggers an event that depends on a dynamic parameter, identifying these cases can prove to be difficult or even impossible [19]. Furthermore, because they modeled their serverless application as a graph with functions being the nodes, they note that the level of coverage achieved is influenced by the level of abstraction in the application. They point out that depending on the granularity of how the serverless application was created, it will have fewer nodes, which could lead to overlapping coverage criteria [19].

D. Summary and Points To Address

With regards to the related work on serverless testing over the years *(Current Testing Approaches In Serverless Applications, Testing The Performance of Serverless Applications, and Coverage Criteria For Serverless Applications)*, there is a clear lack of insight as to which testing practices prove to be cost-efficient in a general business context. Additionally, there seems to be a lack of definition of cost-effectiveness in regards to testing practices of serverless based applications.

For instance, Winzinger mentions a lack of coverage criteria focusing on serverless-based applications [16]. Evaluating testing practices proves to be up in the air due to how recent the serverless trend is. Furthermore, because of the lack of coverage criteria or methods to verify testing practices, cost-efficiency cannot be verified.

Additionally, there seems to be a difficulty with developing a consensus for common testing practices that are time and cost-efficient in respect to the pay-per-use model offered by cloud providers. This can be seen from the studies performed by Lenarduzzi and Panichella [9], in which although numerous participants prioritised integration testing, companies such as Thundra prioritised unit testing.

Despite the fact that there are various testing methodologies currently being used and advocated by industry practitioners and enterprises, there remains no consensus in research on which methods are the most cost-efficient. The variability in response time from different cloud providers remains an issue to address when determining ways to test serverless applications based on performance metrics in a cost-effective manner.

Given the differences in opinions from these industry practitioners, there is then an established need to gather more information behind the scenes as to what cost-effective testing practices actually are.

This thesis aims to gather data from the practitioner's point of view to help further reach a consensus as to what cost-efficiency is within testing of serverless applications and provide a foundation for future practitioners to build upon.

IV. Research Methodology

In relation to the problem and purpose stated in the introduction, we seek answers to the following research questions:
RQ1: What practices are commonly employed to test systems with serverless architectures?

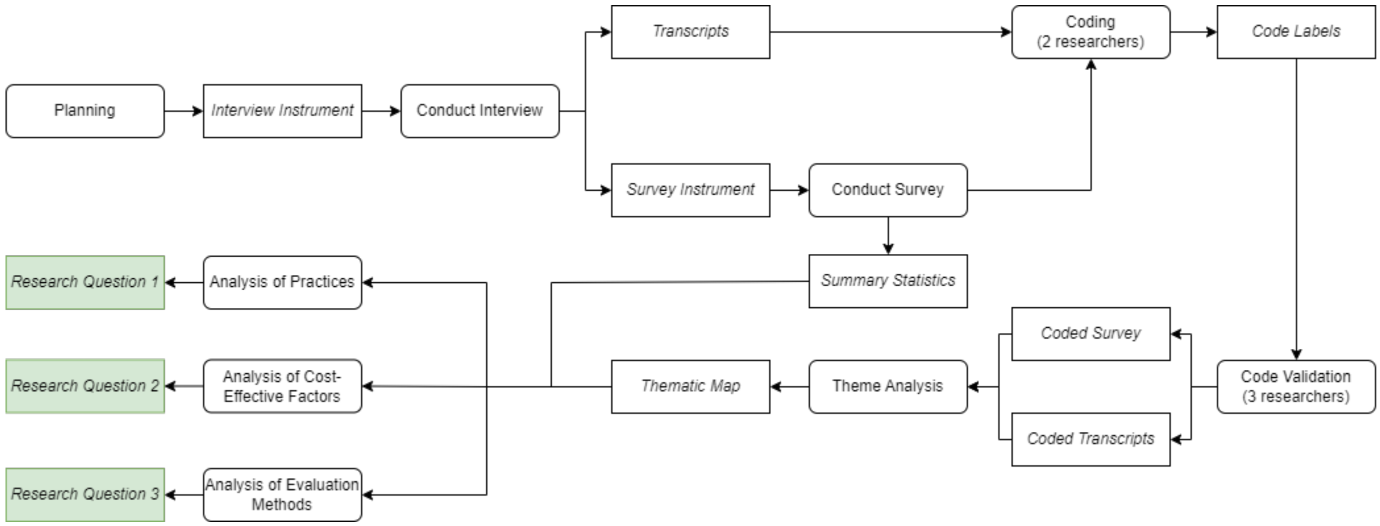

Fig. 1. Research Methodology steps. Square boxes represent artifacts created during activites, which are rounded boxes.

Motivation: To avoid ambiguity, our own definition of what a testing practice is: A testing practice is a form of test case creation or a choice or action taken with the direct purpose of supporting the creation or execution of test cases. Due to the inherent unpredictability of serverless applications [10], different testing practices might or might not be relevant on a technical level. Integration of the cloud solution within the application as well as the pitfall of reduced observability [13] can prove to be difficult challenges wen testing such applications. RQ1 aims to arrive at a consensus as to what practices mitigate this unpredictability and reduced observability, but moreover, derive time and cost-effective testing solutions based off information extrapolated.

RQ2: What are the factors within a software application's life cycle that affect the cost of testing for systems with serverless architectures?

Motivation: Not all software applications are built the same. Business logic changes based on the needs of the consumer, and depending on ever-shifting requirements over time, the projected cost and time efficiency of testing will inevitably change on a case-to-case basis. The goal of this question is to explore which factors have the most impact on testing cost when developing such software applications based on serverless application models, for the purposes of making accurate and constructive observations that point towards efficient testing.

RQ3: How are testing practices evaluated for cost-effectiveness for systems with serverless architectures?

Motivation: Serverless solutions abstract away fine-grained configuration and details regarding the host's system with the main goal of allowing ease of development - developers can just write code and run without worrying about scaling or maintenance of infrastructure. Therefore, there is a unique need to determine how developers judge their testing practices to applicable for serverless-based applications, as well as how they determine cost-effectiveness from these testing practices.

A. Research Methodology

Our thesis is an exploratory research study that analyse the current situation of testing within serverless systems. We planned on a series of interviews with a small sample of at least five software testers. We decided on this based on the given time frame so that we can efficiently perform in-depth insights rather than less detailed insights towards a bigger sample size.

We decided on having semi-structured interviews, meaning there were moments when we formulated follow-up questions that were improvised on the spot. This allowed us to be more flexible with the information we wanted to gather in order to gather supplementary information for the RQs based on the context of the answers provided to us.

To reduce bias, we made use of methodological triangulation, as in using multiple methods to gather data. Thus, after the interviews, we created the survey instrument and sent it out. See the first split of Figure 1 after the *Conduct Interview* activity where the different methods of gathering data start (*Transcripts* & *Survey Instrument*).

We hoped for more survey participants compared to the interviews. We expected a number of at least 20 participants in order to include enough variation in the data to make a strong case for statistical meaning, especially in consideration with the niche type of population we require.

We used the initial results of the interviews as a foundation for the survey. This is to be able to have questions which correlate directly to the interview answers, as it builds a stronger conclusion if the two methods of gathering data can support each other. The interviews come first because they contain a greater amount of detailed and nuanced data. Because of this, the majority of the survey's questions relate to topics mentioned from the interviewees.

The purpose of the interviews and survey was to get a groundwork of themes and detailed observations on the testing challenges for serverless systems. Due to the opinion-based methodology, it allows for direct understanding of practitioners perspectives of cost-effective testing of serverless-based applications. The data obtained is thus relevant to understand the bigger picture of how the industry works and thinks in relation to the topic.

B. Population and Sampling

A total of five interviews were conducted. The participants all work within the Gothenburg Metropolitan area in Sweden. They all worked on serverless applications, and were taken through convenience sampling from our immediate network of connections. Table I displays the participant's role, as well as their experience in development and testing.

The participants work in various software domains: automotive utility, streaming services, and order process systems.

The survey received a total of six responses. Look at Figure 2 for their demographics. One of the respondents was an intern with less than a year of experience, thus we analysed their input carefully.

C. Data collection

The raw data from the interviews and survey is available online at https://zenodo.org/record/7965951#.ZG3p23ZBxPY.

*1) **Interviews**:* A majority of the interviews were conducted virtually, whilst one was in-person, between March-May 2023. The interviews were semi-structured and varied between 30-45 minutes, and asked a predetermined set of questions related to testing and serverless, with the potential of improvised follow-up questions. See Table II for the interview instrument used. Besides demographics, the questions ask how they test for serverless systems (RQ1), the related challenges for cost (RQ2), how they evaluate their testing for cost-efficiency (RQ3), and strategies to reduce cost (RQ1 and 2). At the start of each interview, a brief overlook of the topic was provided. For some participants, the overlook was provided beforehand via e-mail. We also asked for consent to record, as well if the participants want to remain anonymous or not. The interviews were automatically transcribed within *OneDrive*, and then manually corrected in synchronization with the recording. There was a total of five interviews conducted.

*2) **Survey**:* The survey was a cross-sectional, unsupervised, online questionnaire, consisting of quantitative questions and open-ended qualitative questions. It was created via Google Forms.[1] See Table IV in the appendix for the survey instrument used. The questions prompted are similar to the ones in the interview - such as comparing types of tests between serverless and non-serverless (RQ1), ranking priority of different strategies (RQ2), and directly prompting how they evaluate their testing for cost-efficiency (RQ3). Some questions are multiple choice and some are open. The survey had a pilot test where one individual familiar with the subject made sure the survey was easy to understand. The survey was accepting responses during the month of May. The survey received six responses.

TABLE I
INTERVIEW PARTICIPANTS

ID	Role	Dev. Experience	Test. Experience
P1	Software Developer	4 years	1.5 years
P2	Software Developer	7 years	6 years
P3	Technical Lead	12.5 years	6 years
P4	Technical Lead	27 years	10 years
P5	Chief Technical Officer	25 years	10 years

TABLE II
INTERVIEW QUESTIONS

RQ	Interview Question
N/A	1. How big is the enterprise that you work for? 2. What is your development experience in years? 3. What is your testing experience in years? 4. What is your programming background? 5. What is your software development methodology? 6. What serverless providers do you currently use in your enterprise? 7. Tell us about your product that has serverless components.
RQ1	8. What types of testing do you typically perform on serverless applications? 9. How do you test the integration between serverless functions and other services or components in a serverless architecture? 10. What are the most important problems you find when testing serverless applications? 11. Are there traditional testing approaches that you feel are ineffective or inapplicable to testing serverless systems?
RQ2	12. How would you define the concept of cost-effectiveness within serverless testing? 13. Can you provide an example of a situation where you identified a cost inefficiency in testing and how you addressed it? 14. What strategies would you use to manage and reduce costs for serverless systems?
RQ3	15. What are the key test metrics you monitor in a serverless system? 16. How do you ensure that the testing environment is optimized for cost-efficiency while still being representative of the production environment? Can you even ensure this?
N/A	17. Do you know of any emerging trends or technologies in serverless testing that may impact cost-effectiveness?

D. Data analysis

As Figure 1 shows at the activities *Conduct Survey & Coding (2 researchers)*, whilst the survey was being conducted and gathering responses, the interviews were analysed.

*1) **Interviews**:* We followed an inductive thematic analysis for the interviews conducted, as per recommendations provided by [5] and [3]. With a thematic analysis, we analyzed qualitative data so as to find commonly held opinions around

[1]https://forms.gle/wPbCXX2GDwDXmUWg7

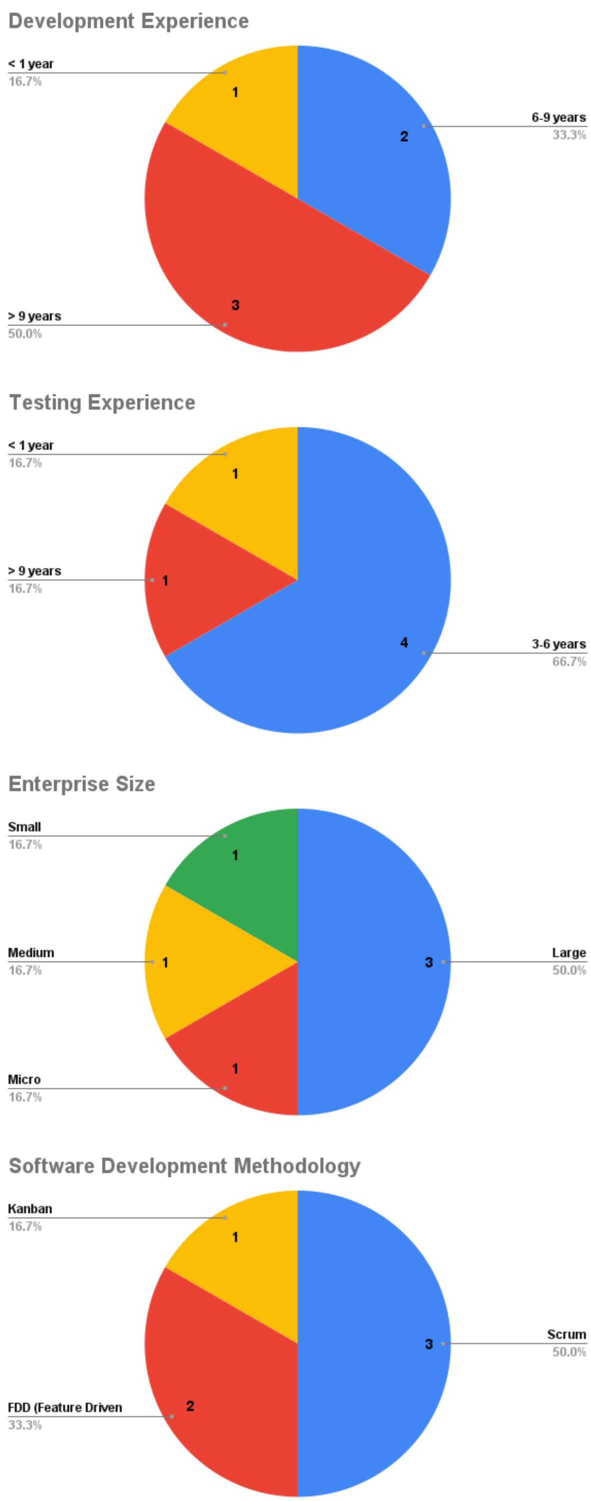

Fig. 2. Survey Demographics.

the subject, and condense them into themes and sub-themes. First, the interviews were coded by us, meaning the repeated subjects or items in the interview transcript were processed down into a singular word or short phrase. The codes were individually conducted and then per each interview, compared to see if we both as researchers agreed on the core aspect of the code. We then chose one of the two codes we felt were more detailed, or combined the two into a new one. If not, we would discuss our disagreement to provide understanding to why we coded it in a specific way. There were codes that we initially disagreed with. We would then explain our point of view, and a consensus was taken based around the arguments provided. After the codes had been labeled (*Code Labels*), they were validated by our supervisor (*Code Validation - 3 researchers*).

Once the codes were complete, we derived common themes among them to connect to our RQs and relevant literature. As inductive implies, we did not decide on any themes prior to the interview observations. Considerations were applied of the participant's different ranges in development and testing experiences, as to ensure that the research findings accurately reflected the diversity of perspectives and capabilities within the participant pool.

*2) **Survey**:* For the survey, the answers acquired from the open-ended questions were coded and applied to themes in the same way as the interviews. Due to the small amount of codes, they were grouped as one unit when applying them to themes. With the quantitative answers, we have summary statistics show the results via charts for each question posed in the survey.

Once both the survey and interviews had their themes generated, we wrote a discussion around these as a way to analyse the results with the research questions in mind. See the final activities and artifacts of Figure 1 implies, regarding the three analysis-activities after *Summary Statistics* and *Thematic Map*.

V. Results

The data gathered will be presented in this section. When quoted in this paper, the participants' answers may have been slightly edited as to remove, e.g. filler words and unnecessary repeats. The themes and sub-themes are defined in Table III.

A. Testing Practices for Serverless Systems

To first focus on **RQ1**, the data gathered from the interviews and survey suggested diverse opinions in regards to what testing practices should be prioritised.

*1) **Unit And Integration Tests**:* During the interviews, Participant 1 (P1) stated that unit tests were good for input validation and were cheap and quick to create, referring to how serverless functions were small, correlating with small tests. This opinion was echoed by P2, in which they also referred to the granularity of serverless applications as a key factor behind the reasoning to incorporating unit tests. Furthermore, P3 stated that they mainly relied on unit tests due to their

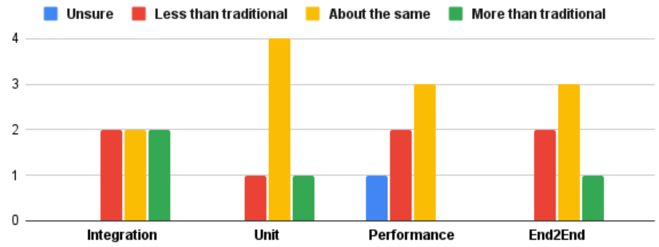

Fig. 3. Survey results on differences in the quantity of tests between serverless and traditional systems. (Q8)

serverless functions being simple and small. However. multiple participants in the interview expressed dissatisfaction with unit testing. P1 highlighted how unit tests fail to consider the execution context of the serverless functions:

> "One of the bigger problems with just unit tests, you can't really test those events. You can sort of generate your own events based on what you think happens, but the only true way to test those events is if you actually have a database, put something in it, and that event is generated by the cloud and then it's passed into your function. So, unit tests I would say work to only test the business logic of your code, but they don't cover what happens when your function gets an actual input when it's deployed." - P1

P2 stated that unit tests were not useful for specific metrics such as response time, and optimising resource allocation, though despite this fact noted that testing could at least be done for correctness and integrity.

> "Of course, performance will vary because you have different instances and different machines that they run on, and you don't really have control over that. That's kind of the purpose of the unmanaged serverless infrastructure. So while the performance unit tests will probably not be very giving, you can at least do testing for correctness and integrity." - P2

Finally, P5 stated that unit tests were not preferred as much as other forms of testing because they felt integration tests provided broader coverage given how small their serverless functions were. They anecdotally refer to a trivial command line application and explain that greater cost-effectiveness can be achieved through integration testing rather than unit testing, because of the initial high code coverage. P5 specifically justifies this by saying that it is "supremely nice to have coverage that shows you that the commands are working as expected", referring to integration tests. This was a common opinion confirmed through all participants except P2.

> "We do write both [unit and integration tests]. What I see is if it's a trivial command line application then it's not necessarily worthwhile to write a lot of unit tests. But even for a trivial command line application, it's supremely nice to have coverage that shows you that the commands are working as expected. Then, depending on how complex the internals of the thing gets, that's more motivation to write more unit tests." - P5

However, P5 mentions that, when their serverless application got more complex, they began incorporating unit testing, stating that the logic behind these integration tests began to get more and more complex which took away from development time. For P5, unit tests eventually became faster and more efficient both in terms of time, as well as cost.

These differences in opinions in testing practices were also reflected from the responses from the survey. The results from Figure 3 had a mixed response rate as to what tests were incorporated more over the other. Specifically, integration testing received mixed feedback, with an equal number of people stating that it was used less than testing traditional applications such as microservices and monolithic applications, whereas an equal amount of other people were contending that it was used more often.

In regards to performance and End To End (E2E) tests, there was a lack of sufficient discussion around them. P3 goes briefly into how their unit tests for performance only test for correctness and integrity.

> "While the performance unit tests will probably not be very giving, you can at least do testing for correctness and integrity." - P3

Figure 3 doesn't show a noticeable pattern for E2E tests, but we noticed that performance testing was used less or about the same in serverless applications. This data was still not sufficient to build a strong case around as we would need some qualitative data to back it up.

RQ1 (Practices): There was a mixed usage of integration and unit testing. Performance and E2E were not discussed much.

*2) **Testing Environments**:* Despite the fluctuations in opinion about the types of tests, all interviewee and survey participants unanimously agreed on the need for dedicated testing environments in order to test the serverless application. All of the participants in this study had at least one dedicated testing environment, and of course, the production environment. P2 mentions that one reason behind this is to be able to cover different scenarios that cover external factors and, therefore, contribute to positive observability in regards to errors and bugs. Curiously, emulation of the production environment in order to reduce costs was mentioned several times within interviews taken from P1, and P2.

P2 mentioned the creation of different testing environments using emulation and the tooling provided within AWS. They then parameterized these test environments with parameters such as runtime environment, and specific data sets in order to more accurately test the production environment and take into account different contexts that could possibly affect the application's behavior.

> "One of the benefits of deploying things out of local into staging in depth environments, is that whatever routines and tests would perform in the environment, we can actually then see and aggregate all the errors and successful tests and visualize them in one place where all the developers and the managers can see and keep track of." - P1

These observations were confirmed by a survey participant, who stated that the serverless provider they used had sufficient tooling to create an affordable test environment that was almost identical to the production environment in their enterprise. Furthermore, this survey participant mentioned that this was difficult and expensive when it came to non-serverless applications, but did not go into further detail why.

P1 elaborates further onto the specific software used for emulation, stating that they use a software called LocalStack which attempts to run the serverless functions against an emulated production environment. This was also similarly echoed by another survey participant, who pointed to the benefits of using this software.

> "In case of AWS, we use a framework called LocalStack which emulates all of the different AWS services on your machine and you you are able to connect all of those services with your serverless function and sort of get the idea of what's going to happen when you deploy." - P1

Emulating environments comes with its challenges though, as P1, P3, and P5 say its difficult to replicate without also utilizing the cloud.

> "So, even though we do this local testing, we still deploy to the cloud and use custom software to perform all sorts of different tests to cover different scenarios, different acceptance criterias. And I would say that is our end goal to have fully functional tests in the cloud, not just locally." - P1

> "I can't on demand spin up a replica of it on my own workstation or on a test machine somewhere else. It needs to run in Google Cloud." - P5

> "We do have an emulator when doing things, small things locally on your computer, but we can't emulate the production environment, so we just have literally an identical set-up for like private use, so it's not emulated" - P3

In Figure 8, respondents ranked it as slightly difficult. Out of those that didn't rank it as difficult, one of them was an intern with less than a year of experience, and thus was assumed to have less insight. The other one, an experienced owner, has a simple product that doesn't use a lot of serverless components, and thus the replication of environments might be easier than otherwise.

A lone outlier, P5, mentioned only using one testing environment instead of multiple like the other participants. With that, they had a problem that only one set of tests can be performed at a time.

> "The fundamental problem here is we have one test environment. Only one set of tests can be done against one test environment at the same time." - P5

> **RQ1 (Practices):** Emulating smaller parts of an environments can help with cost, though emulating an entire production locally is near impossible.

B. Factors that Affect Cost-Efficient Testing for Serverless Systems

*1) **Choice of Serverless Provider**:* Regarding **RQ2**, interview and survey participants pointed to the choice of serverless provider as a strong immediate factor on the cost-efficiency of testing the entire serverless system. All interview and survey participants used either AWS, GCP (Google Cloud Platform), or Microsoft Azure. Different experiences, both good and bad, were had based on the choice of serverless provider due to the features provided from each respective cloud vendor's solution. For instance, P2 praised the flexibility of AWS EC2, which was AWS' solution for virtual machines in the cloud. They specifically highlight the configurability and ease of deployment and management as a key factor in choosing AWS EC2, in comparison to the Google equivalent (Google Compute Engine). P1 also mentions the differences in performance with AWS EC2, stating that the new Graviton processors perform a lot better in the cloud than the previous generation, resulting in lower pay-per-use time due to being more efficient.

P3 had a good experience with incorporating Azure Functions, which is Microsoft's FaaS solution, due to its feature called Durable Functions which allow for labeling of functions with different labels that are optimised to do different things, such as performing read and writes from a database, or performing complex sequential tasks and orchestrating the execution of other serverless functions concurrently. For P3's product, which was a real-time video collaboration platform, the incorporation of Durable Functions allowed for a more performant system when streaming one user's actions to other users on their software product based on serverless technology.

On a negative note, P5 mentions that when choosing the serverless provider, the end-user is left at the mercy of the logging and observability tools provided by the serverless provider. For example, P5 discusses how for them, using AWS Lambda restricts them to using logging and observability tools designed for AWS Lambda in order to avoid technical debt by keeping it within one serverless provider, such as AWS CloudWatch, which is a monitoring platform tool for the purposes of collecting real-time logs and metrics from these Lambda functions. Curiously, P2 mentions the usage of AWS CloudWatch for monitoring purposes despite having a multi-cloud serverless application in which they use both GCP (Google Cloud Platform) and AWS offerings, suggesting that both participants valued different testing metrics whilst testing their serverless application.

In addition to this, P2 goes into specifics into the different offerings that cloud providers marketed, stating that in their personal opinion they felt that Google BigQuery, a cloud solution providing data warehouse storage for the purposes of data analytics, outshined the AWS equivalent (Redshift) in terms of pricing and applicability to their use-cases of creating a serverless-based software application revolving around real time data streaming. When asked further about the various serverless solutions available, they mentioned that if another serverless vendor happened to have features better suited to their specific use cases, the concern of being "locked in" to another vendor (vendor lock-in) becomes a very real concern for them. They cited development time as a factor in migrating to the more attractive serverless provider, mentioning that it would take too long for them to switch platforms given how established their product is in the other platform (AWS and Google Cloud). The differences in pricing from cloud vendor to cloud vendor were highlighted by P2, P4, and P5, mentioning that vendors oftentimes send notifications and emails five months in advance regarding price changes of specific serverless products from season to season.

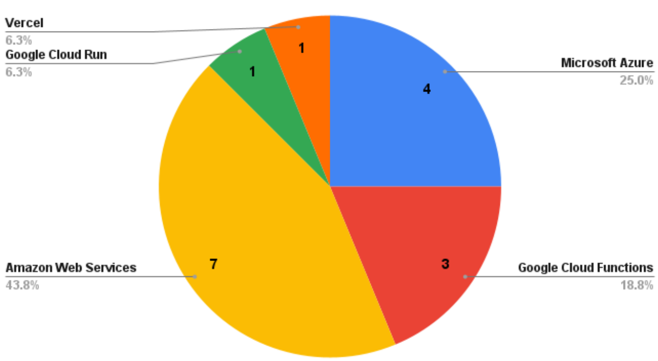

Fig. 4. Demographics of which serverless providers the interviewees and survey respondents use.

> "It would just take too long of the time from development perspective to move away. So kind of a vendor lock in, but something that we're still satisfied with. [...] If the vendor would provide similar service as the other vendor, and they would be priced in within the same area, then definitely using the same vendor would be good. But yeah, unfortunately some vendors have bit of an edge in certain products that they offer. " - P2

> **RQ2 (Factors):** Different serverless providers have their own benefits and drawbacks, and can be time-consuming to switch from later in development.

*2) **Observation Tools**:* Participants in this study explicitly referenced the usage of numerous monitoring tools as a key factor in influencing the cost of testing of serverless-based applications. In the context of serverless testing, monitoring tools are software tools designed to monitor, track, and give insights into metrics such as resource allocation, response time, and resource consumption.

P1, P2 and P4 justified the usage of monitoring tools such as AWS Cloudwatch and Datadog because it allowed them to effectively observe the execution of these serverless functions. They point to the convenience of being able to detect potential issues, bugs, and excessive costs incurred from failure to satisfy the aforementioned metrics. Notably, P2 correlates the usage of monitoring tools to the ability to motivate changes or suggestions to the product through well presented data.

> "It's good for the developer to know what's going on generally in the system. It's good for the team to know that. It's also good for the business side of things to know – business people to know it. Because then you can motivate any changes or improvements or suggestions that you want to introduce in your testing pipeline or product pipeline. It's like having the data and the data that is well presented." - P2

P3 discusses the importance of monitoring their serverless application through an observation tool called Durable Functions Monitor. Their product, a real-time video collaboration platform, proved to be demanding to test locally due to the millions of function invocations for given tasks, which overloaded their own computers containing high end processors for heavy workloads. Incorporating Durable Functions Monitor was a far more impactful factor for P3 in influencing the cost of testing for serverless-based applications because it allowed them to closely monitor the immense quantity of function invocations, which their own computers were not able to do.

> **RQ2 (Factors):** Keeping track of metrics via observation tools help testers observe cost-inefficiencies with their tests.

*3) **Budget And Way Of Working**:* On the topic of pricing, two interview participants (P4, P5) pointed towards budget as a minor factor in influencing cost-efficient testing of the serverless-based application.

The concern for budget when developing serverless applications was very much dependent on the enterprise and type of product. P5's product relied on a REST API created through serverless functions, allowing it to be available 24/7. However, P5 highlights that the total amount of network traffic per week did not justify the creation of a significant quantity of tests, additionally pointing out that the REST API was mostly for internal use and was developer-facing.

> "All the serverless tech that we created has been internal facing... developer-facing, and because of that, costs—like operating costs and call costs, API call costs—never became a significant factor. It's just not enough activity ... We make a couple of 1000 calls a week to the cloud function. We're not doing millions or billions calls a week." - P5

P4 had a similar situation as P5, in which they had parts of their application incorporating serverless functions performing tasks such as data transformation, but, ultimately for them, the usage and incoming request traffic was not enough to justify budget concerns. Interestingly, P4 brought up the creation of tests themselves as a factor for cost-efficiency. In their case, they focus on quick and rapid deployment, mentioning to the customer upfront that they offer a cheap solution in exchange for close and constant communication with the development team. In this way, they were able to omit tests completely in order to adapt to the ever-shifting requirements of the customer, which in turn facilitated rapid bug fixing and rapid deployment.

A survey participant had a similar train of thought:

> "Time is also a cost for us, so we seldom test features that are "immaterial" or "insignificant."" - Survey Participant 4

This was briefly reflected when asked about code coverage of serverless applications, as seen in Figure 5. In the follow-up question where respondents justify their code coverage percentage, multiple survey participants said it was due to time constraints—that covering too much will be expensive.

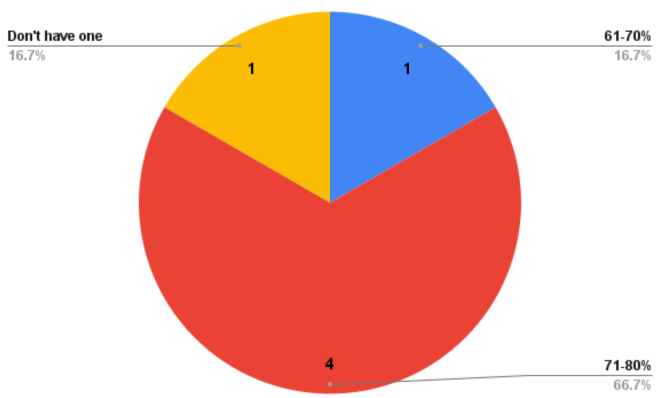

Fig. 5. Survey results of the code coverage goal for their serverless applications (Q16)

> **RQ2 (Factors):** Depending on the product and relation with customers, its not always cost-effective creating tests in the first place.

*4) **Granularity**:* A factor related to cost-efficiency that we noticed was the concept of reducing the size and scope of functions in serverless-based applications. This is called *granularity*. The granularity of the system was talked about positively in our interviews.

> "It costs less because running a function for a small task is cheap because you're paying for the duration of the function." - P3

The functions should not only be simple, but also focused on a specific task.

> "Because from what I've noticed a lot of serverless functions are not huge. They are very compact pieces of code that serve a specific purpose, so if designed correctly they will not hold up your system." - P1

P2 agrees to an extent, saying that the testing of their functions don't have many dependencies, though still a few.

> "Those functions don't really have too many dependencies. I believe it's three replies and two levels deep that we're going here, so one function triggers two more functions and then waste the reply from both of them and then performs a sort of behavior. And because the amount of data that goes through them is not that big, yeah, it's there's not really delay considerations that have to be kept in mind. We don't need to consider how long the function has to run before it gets killed, because usually it takes... less than a second to perform it" - P2

However, as P2 clarifies, they did not consider lowering the response time further as they were content with it.

P3 focuses on granularity a lot, mentioning that they implement their real-time video collaboration serverless-based platform with the fan-in fan-out architecture, which is a design consideration they performed in order to improve resource allocation and response time. The fan-in fan-out architecture introduces a high level of granularity and complexity, because it entails the splitting up of bigger tasks into smaller sub-functions. Because of this design consideration, they are then able to fine-tune each and every function for performance.

> "You're paying for the amount of CPU time that your invocation takes, so splitting up the task into different serverless functions means you have less time for each instance and they can be done in parallel. Usually, from what I've seen is splitting it up can actually be faster than doing them in sequence because of context switching." - P3

> **RQ2 (Factors):** Resource allocation and response time is affected by the size and scope of individual functions.

*5) **Automation**:* Automation of testing (e.g., through CI/CD), was briefly mentioned though held in high regard according to interview and survey respondents.

> "Automation and tests work well together when they're combined together." - P2

P5 made use of automation in their product. P2 and P3 wished for more automation:

> "There's no automated way to [retune serverless parameters]" - P3

> "The processes themselves are fine, they work, but the gluing things between them can always use more automation, in my

> personal experience." - P2

Taking a look at the survey, Figure 6 indicates that testing for CI/CD was clearly the most prioritized as a strategy to reduce costs. However, Figure 8 also shows that this task can be difficult to set up.

> **RQ2 (Factors):** Automation positively affects cost overall, though results suggest that this is potentially not simple to incorporate.

C. Evaluating Cost-Efficiency for Serverless Systems

On the topic of **RQ3**, our study participants highlighted that testing practices were evaluated for cost-effectiveness for serverless-based applications through performance metrics, and KPIs *(Key Performance Indicators)*. In short, KPIs are company-facing metrics that measure business goals, such as sales growth and customer retention, while performance metrics measure the technical aspect of the product.

All interviewees and survey participants emphasised response time as a key performance metric towards evaluating test practices for cost-effectiveness.

> "Cost is more often than not directly related to the performance of the actual system. If one part of your system takes ages to do something, that means that resource is in use longer than it should be, which naturally just drives up the cost of your whole operation." - P1

> "If you have a function that doesn't do much but it takes a long time to do, but it does the same work as a function that does little or takes a short time. You're paying for the amount of CPU time." - P3

The size of the tests affect the response time negatively according to P5.

> "For running tests against rest APIs, when the test gets large it takes a long time to run it and that become, that makes the iteration times for me as a developer long. That is cost inefficient with regards to my salary." - P5

P4 provided a concrete example in which they had a serverless function that transformed bytes into a PDF format. This serverless function was then called from a monolithic back-end residing on a slow server. This incurred a lot of costs for them, because the Lambda function would fetch data and encounter slow response time from the backend. Luckily, P4 praised AWS' Cloudwatch for notifications when a serverless function is incurring too many costs due to response time, which allowed them to alleviate the situation by scaling up on their hardware.

A survey participant faced a similar issue in which they misconfigured their test environment, resulting in them downloading an 8.4 GB file per test case. Consequently, each test case took 20 minutes to run in the cloud, incurring a lot of costs.

> "Last year we noted that one of our Cloud Functions had it's test cases running for >20 minutes every time, which drove up costs quite a bit. Turns out one of my colleagues was downloading a 8.4 GB file for every test case that was run, due to invalid configuration of the test setup." - Survey Participant 1

In relation to those cost-inefficiencies, P1 believes that a the biggest detriment to cost are developers being unfamiliar with serverless and, thus, misconfiguring the tests.

> "They will not hold up your system, they will execute in a time I would say from 100 to 500 milliseconds, which is not done a lot when you put it into a bigger system. I would say a lot of the cost usually just comes down to people not to being familiar with what they're using and misconfiguring things in the end." - P1

In order to satisfy the metric of response time in their testing practices, P3's use of the aforementioned fan-in fan-out technique from the results in **RQ2** lead to a more responsive system in their product use-case scenario of transcoding files from one video format to the other.

> **RQ3 (Evaluations):** Response time is a vital metric to track, due to how serverless is a pay-per-usage model.

A different perspective on evaluating cost-efficiency—P1 said that knowing what is cost-efficient comes through experience with the application being developed, that for each iterative deployment "you will know what's bringing the application down." This is, to an extent, related to organizational and product maturity.

Survey participants were asked if they followed recommendations from Winzinger and Wirtz's coverage criteria. As seen in Figure 7, Participants pointed to testing a lot for *All-restored* (stateless, idempotent results from function invocations) and somewhat for *All-max-time* (the peak response time required to perform the function).

VI. Discussion

In this section, the themes derived from the qualitative interview and survey results will be discussed in relation to the RQs.

We chose what we felt were the most prominent themes. The results of the study show that there is still no general consensus with regards to achieving cost-effective testing for serverless-based applications.

A. Defining Cost-Efficiency For Serverless-based Applications

The term "cost-efficient" has many meanings. For one, there is the monetary perspective—budget spent is cost. Secondly there is time—time spent is cost. Commonly, time is directly associated with the budget, as the expenses versus monetary gain of the enterprise needs to be considered.

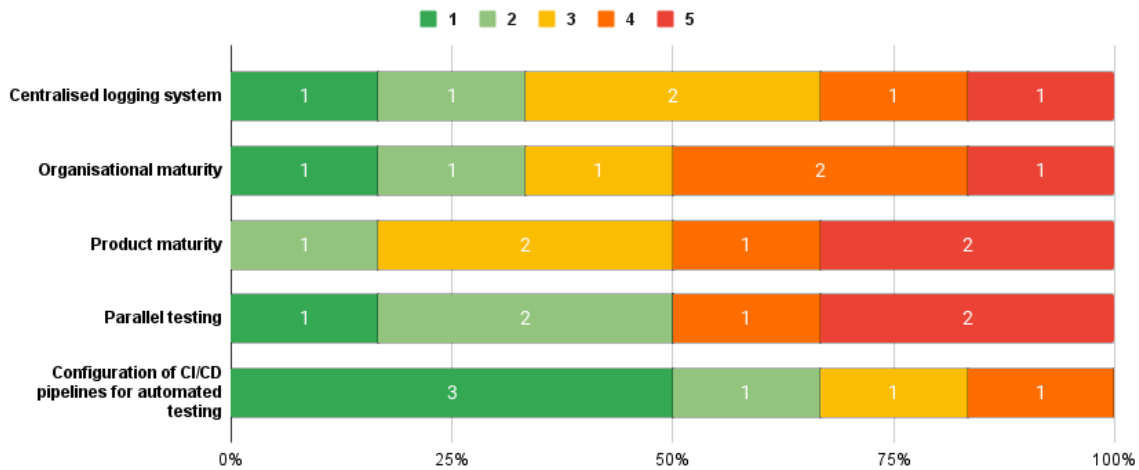

Fig. 6. Survey respondents' prioritization of strategies that reduce cost, **1** is highest priority. (Q13)

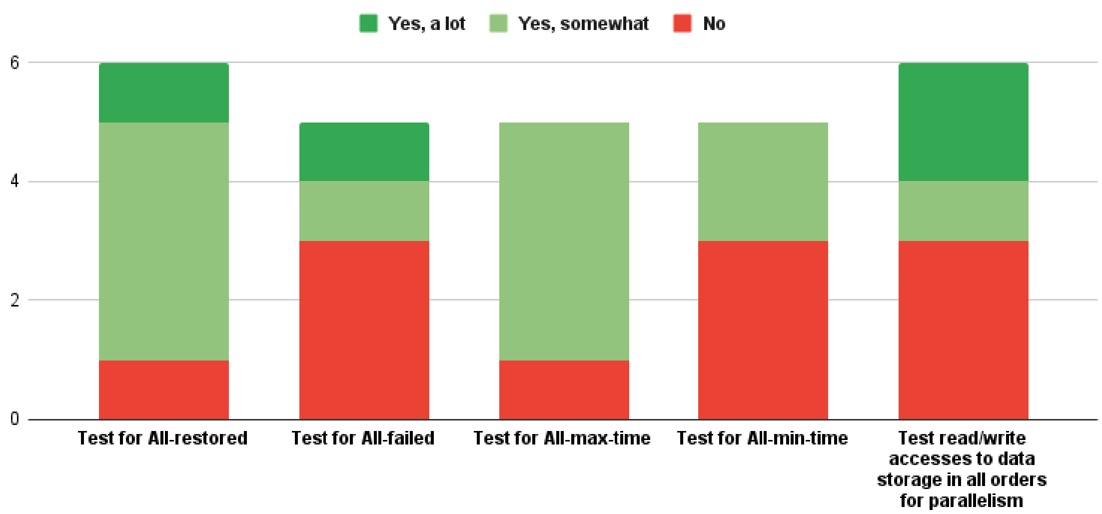

Fig. 7. Survey respondents if they follow recommendations from [17] in regards to characteristics of serverless. (Q18)

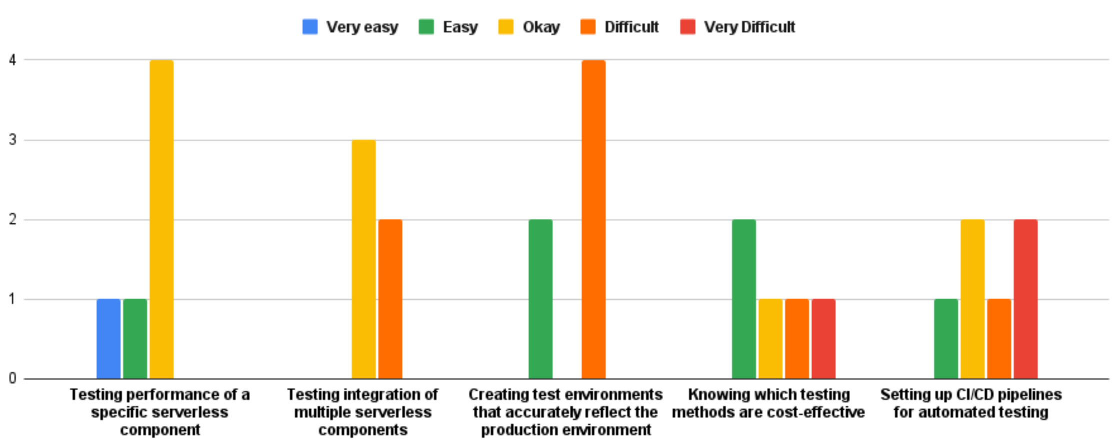

Fig. 8. Survey respondents difficulty rating of concepts related to serverless testing. (Q9)

TABLE III
THEMES (BOLD) AND SUB-THEMES

Theme	Explanation
Experience	Participants' development and testing experience.
Enterprise	Size and software development methodology of the participant's enterprise.
Type of Product	What product that make use of serverless technology.
Serverless Providers	Different providers have different advantages and disadvantages.
Testing Tools	Many participants expressed directly which tools they use to handle their testing.
Automation	Automatically deploying, creating, and performing tests is both used, and wished for more of.
Emulation of Environments	It is cost-effective to locally emulate the serverless testing environment, although difficult to entirely replicate or recreate the production environment.
Organizational Maturity	Developer skills, enterprise size and budget.
Budget Concerns	The available budget of the enterprise affects how important serverless testing's performance is.
Product Maturity	The functionalities, stability, and lifespan of the product.
Vendor Lock-in	Choice of serverless provider is difficult to change later in development.
Testing Metrics	The measurements to determine whether tests are sufficient.
Observability	To track metrics, various monitoring tools and strategies are deployed.
Resource Allocation	Minimizing the amount of resources tests take to affect metrics positively.
Response Time	A vital metric due to serverless being pay-on-demand.
Granularity	Reduce response time and resource allocation by dividing large functions into smaller ones.
Parallelism	Reduce response time by performing multiple tests at the same time.
Types of Tests	There are other expectations on what types of tests are prioritized in serverless.
Integration Testing	Achieves higher code coverage the less complex the serverless-based application is.
Unit Testing	Grew more common due to achieving higher code coverage the more complex the serverless-based application is

In regards to the pay-per-usage of testing serverless, monetary costs needs to justify the saving of time for the enterprise. Because of this, participants have different priorities what is cost-efficient for them.

P3 talks generally about reducing monetary costs for their product, such as reducing size of their functions in combination with running functions in parallel. Their product, which is video transcoding, requires a lot of traffic when running—including their tests, as they have millions of instances performing one simple task. This means there is a higher need to be cautious around the budget spent by the tests.

We can comparing this reponse to P2 and P5, who explicitly say they don't need as much concern for budget in their testing, as they don't make intense use of their serverless functions. Because of this, they tend to have grievances with the complexities of the serverless system—such as when P5 points to the testing environment they have and how being in sync with other developers is difficult. P2 find much of the unnecessary manual labour bothersome. In other words, their priorities of cost-efficiency comes down to making the developer's experience faster and easier.

In relation to the enterprise size, Mulani et al. mentions that the size of an enterprise is correlated to the budget [12], which points to P3's enterprise having a tighter budget due to it being micro-sized.

Potentially, this could explain some of the differences in responses found in the survey. For instance, the different prioritizations of parallel testing, as shown in Figure 6. Half of the result state that they prioritize parallel testing highly, compared to the other half who do not.

This difference in both product and organization's maturity could affect the definition of cost-efficiency. This entails that a practitioner's experience with serverless can highly depend on what they are intending to do with it, and what size, budget, and skills they have as an enterprise.

In conclusion, the concern for budget could affect what approaches to testing are prioritized. Is the monetary spending of running tests, or is the time spent dealing with the complexities of serverless more important? This will affect which testing practices and strategies need to be prioritised, and which problems will be more prominent.

RQ2 (Factors): The budget and skills of the enterprise and how the product intends to utilize serverless can greatly affect which factors of cost-efficiency matter.

B. The Cost-Efficiency Of Testing Practices Within Serverless-Based Applications

A common theme occurring within our study were the *Types of Tests* performed for the sake of cost-efficiency within serverless-based applications. The results we gathered pointed to a situational mixture of unit and/or integration tests, such as already mentioned briefly in the results regarding Figure 3. We were unfortunately not able to come to a consensus as to what testing practices were prioritised over the other for cost-effectiveness. Instead, we noticed that interviewees and survey respondents judged their testing practices for cost-efficiency based on the product's use case and demand.

Initially, when we began looking into testing practices correlated towards cost-efficiency on serverless-based applications, we made an assumption that integration tests would be prioritized more over unit tests. This assumption was mainly influenced from multiple factors. One of them was

Yan Cui's advocacy towards integration testing as the de-facto testing practice for serverless-based applications. We valued his background knowledge highly due to his reputation as an "AWS Hero"—a title awarded by Amazon for profound expertise in serverless practices and a deep understanding of the AWS platform based on the consensus of the AWS community of serverless practitioners [9] [6].

Cui explained that integration points are where the most bugs occurred, and justifies this stance by stating that serverless applications are event-driven, and therefore, the complex interactions from serverless function to serverless function should be tested rather than testing each individual serverless function [9].

Another factor behind the assumption of prioritising integration tests over unit tests comes from Lenarduzzi and Panichella. They foresaw the definition of integration-level test criteria over unit-level criteria, reasoning that there is a lot of complexity residing in serverless-based applications. According to them, this complexity comes from the heavy interactions between function to function in serverless-based applications, highlighting the importance of covering these interactions through integration tests rather than unit tests [9].

The interview participants tell a different story, however, with P2, P3, and P5 preferring unit testing. For example, P5 noticed that at the beginning in their early days of developing their product, a real-time game analytics platform, they relied on integration testing in order to be more cost efficient with their testing practices, citing that they "only really cared about the end response of data from the functions" rather than each function's execution. This falls in line with Cui's advocacy for integration testing [9].

What is curious, however, is that P5 then goes on to mention the enterprise's shift from integration testing to unit testing because of how complex the integration tests became the bigger the serverless-based application got, contradicting Cui's recommendations and Lenarduzzi and Panichella's reasoning [9].

P3 takes a similar approach with the fan-in fan-out architecture. Regardless of this high level of granularity and complexity, they advocate for prioritising unit tests, despite the aforementioned factors Cui, Lenarduzzi and Panichella described with the inherent complexity of serverless-based applications. Their reasoning behind this is that they found it cheaper to develop their test case towards the end result from the fan-in fan-out technique, rather than developing integration tests for the millions of invocations done from their smaller functions.

We speculate multiple reasons to why this could be the case. First, it could be a case of product and its architecture that need to be considered. The interview participants focusing on unit testing had a distinct focus, such as mobile game streaming (P2), real-time video collaboration (P3), order and logistics processing (P4), and game analytic tools (P5). The outlier that focused mainly on integration tests was P1, who had a product focusing on automotive services. The point with this observation is that these varying use cases could have influenced their choice around which tests to prioritize.

RQ1 (Practices): Unit tests could potentially change in importance depending on the serverless application's use case.

Our second speculated reason as to why they prioritised a mix of unit and integration tests is because we noticed a correlation between the testing practices introduced and the complexity of the serverless-based application as a whole. We refer back to P5's anecdote regarding their justification into switching from integration to unit testing.

> "We do write both [unit and integration tests]. What I see is if it's a trivial command line application then it's not necessarily worthwhile to write a lot of unit tests. But even for a trivial command line application, it's supremely nice to have coverage that shows you that the commands are working as expected. Then, depending on how complex the internals of the thing gets, that's more motivation to write more unit tests." - P5

To break this down simply, P5 takes an example of a trivial command line application. He mentions that it is not necessarily worthwhile to write a lot of unit tests given how trivial the command line application is. However, he refers to integration testing eventually, stating that it is "supremely nice to have coverage that shows that the commands are working as expected". They then go on to mention that there is a lot more motivation to write unit tests when the application has a higher level of complexity.

The way we interpreted this is that integration tests satisfied their expected level of code coverage the less complex the application is, or in this context, less trivial. In other words, they only cared that "the commands were working as expected".

For the sake of a deep analysis, they do not explicitly mention their expected level of code coverage in our results, and we were not able to ask follow-up questions regarding the matter.

Therefore, for the sake of argument, we take another example which is P3 and their firm stance on implementing only unit tests. In their case, their incorporation of the fan-in fan-out technique resulted in millions of functions performing one big task in order to have a more performant system, citing "context switching and better resource allocation" as key factors behind this architectural consideration. P3's decision to incorporate unit testing for their complex system involving millions of functions performing a single task concurrently directly corroborates P5's sentiment of unit testing being more prioritised after a higher level of complexity. This is because as mentioned before, the fan-in fan-out technique introduces greater levels of granularity and complexity, resulting in an increased amount of event interactions between serverless function to function.

Looking at Figure 8, the integration of multiple serverless components can be a struggle for developers to deal with, supporting how a bigger, and more complex system with many serverless components will require detailed testing.

From the aforementioned examples above, participants and their gravitation towards unit tests seem to suggest that unit tests satisfy the detailed testing required for a complex serverless-based application.

> **RQ1 (Practices):** Unit tests could potentially increase in value depending on the complexity of the serverless application.

C. Identifying Proper Testing Metrics Through Observation Tools For Serverless-based Applications

The theme of *Observability* and *Metrics* were connected together because interview participants mentioned the importance of identifying proper testing metrics and KPIs through observation tools in order to determine effective testing practices for serverless-based applications in a cost-effective manner. For a brief recap, monitoring tools are software tools designed to monitor, track, and give insights into metrics such as resource allocation, response time, and resource consumption.

Most notably, P2 goes in-depth into the reasoning behind identifying proper testing metrics, mentioning that without knowing any of these metrics, it becomes difficult and infeasible to evaluate new features, implementations, and/or changes to the entire system.

> "KPI's are everything in this regard. We want to keep track of how many tests are executed, how much every single test consumes the data, how much costly it is and just keep that data pouring in as much as possible so we can actually keep track of it. [...] And of course then developing visualization, or using visualization tools to effectively keep track of them on a daily basis. That's number one I would say because without knowing any of this data it's very difficult to measure any new implementations or any changes that you introduce and how effective they are." - P2

Furthermore, P5 backs up this statement. When asked about what testing approaches would be ineffective or inapplicable to testing serverless systems, they mentioned that "any kind of test approach based around detailed introspection of the performance of the underlying system" was ineffective. They justify this statement by citing the underlying abstractions the cloud providers implement in regards to configuration of the host environment.

The priority of identifying proper testing metrics through observation tools becomes even more apparent when P2 mentions that the observation and analytics tools differs from provider to provider, preferring to use Google Cloud's suite of observation tools rather than AWS tools **despite** the flexibility AWS offers with its serverless products. They choose Google Cloud's suite of observation tools because it fits better within their use-case of a mobile game streaming platform.

> "With AWS EC2 instances, there are many more tools and much more flexibility in how you can configure, deploy and manage them versus to what Google Cloud offers, including price. But at the same time, the analytics tools with what Google works and develops there, yeah, they're pretty much king in the market." - P2

These statements correlate with the initial findings of Winzinger and Wirtz several years ago in which they noted that cloud providers enforced a layer of abstraction over their serverless product by expressly choosing what the end-user can configure. Winzinger and Wirtz explicitly mention the difficulty in satisfying the *All-Resources* coverage criteria as a direct consequence of this abstraction [19]. Hence, it is no surprise that the serverless practitioners we interviewed and surveyed relied on observation tools in order to derive their own performance testing metrics and KPIs that can be effectively monitored using these observation tools, just like Winzinger and Wirtz did.

Building upon this from the results we gathered from this study, there seems to be a contradiction between the layers of abstraction such as what Winzinger and Wirtz mentioned. We found out that serverless-based applications implementing FaaS are typically comprised of hundreds of smaller functions interacting with each other, and that due to this granularity, test cases had to be adapted to fit the architecture. The contradiction here is that, although serverless applications offer a generous layer of abstraction, the abstractions themselves force the architect and developer to consider design choices aiming towards granularity that allows for fine-tuned control of low-level behavior.

For instance, in order for P3 to satisfy their derived testing metric of response time and resource optimisation, they found that the best way to do so was to incorporate the fan-in fan-out technique and introduce more granularity into their serverless-based system by splitting bigger tasks into hundreds of smaller functions, resulting in a system so granular that millions of invocations were needed to perform tasks such as transcoding and streaming within their application.

In order to justify this assumption that we have made, we refer to Panichella and Lenarduzzi in which they mention the difficulty in creating test cases due to how serverless-based applications restrict the developer towards architecting an event-driven system.

> "Typically, a serverless application has hundreds of lambda functions that interact with one another. We can locally test code coverage at the unit level (e.g., branch coverage). However, the question is this: 'how can we be sure that we tested all possible events a lambda function may receive from other functions?"' - [9]

> **RQ3 (Evaluations):** Develop metrics that are not only quantifiable, but trackable based on the observability tools incorporated into the serverless-based application in order to be cost-efficient.

D. Threats to Validity

*1) **Threats to External Validity**:* Interviews were conducted with software testers in the Gothenburg Metropolitan Area, meaning the codes could have had a bias. This is because the software testers might have a common culture or practices that differ from other parts of the world. We reduced this bias through the online survey, as that doesn't share this limitation as much as the interviews, because the results are from a larger, international pool. Unfortunately, the amount of responses were quite low in the survey. This means the quantitative results may not have been enough to give confident answers around the global industry.

*2) **Threats to Internal Validity**:* For the interviews, there was a threat to internal validity as the developers may not have given us their true opinions, out of a social fear that it can affect their career negatively. We mitigated this by allowing the participants we interviewed to be anonymous if they so wished.

Regarding the survey, some of the answers, especially in the open-ended questions, risked being incomplete or inaccurate. We attempted to reduce this bias by having the questions remain clear and focused, to not lead away anyone answering the question. We added descriptions of some questions so as to have the respondent have the same understanding as us. In hindsight, we unfortunately could have provided further description of certain concepts presented, i.e. *organizational maturity*. We also had a small pilot test of a participant trying out the survey to make sure it is within an acceptable standard before sharing it further.

*3) **Threats to Construct Validity**:* There was a risk that we might have had a strong bias to a specific answer we were expecting, either consciously or subconsciously, such as when the qualitative data was coded. We lacked experience in conducting this process too. Through asking for assistance from reputable sources more proficient than us (e.g. our supervisor), this threat became less of an issue.

*4) **Threats to Conclusion Validity**:* The validity of our conclusions can prove faulty due to the selected research methodology. Since interviews and survey are opinion-based results, they are not a direct measurement for cost-effectiveness as a whole for testing within serverless-based applications. There is a possibility our thesis's conclusions will instead provide a framework for future research, rather than finding a conclusive answer alone. For example, a case study that follows an organisation transforming their application to make use of serverless could allow for more detailed and direct measurements of cost-effectiveness. The reasoning behind this is that it does not rely on opinions as much as an interview or survey would. Though, our conclusions we had may still be beneficial to this research area. We remained tentative when analysing the data we acquired.

VII. CONCLUSION

This thesis focuses on researching the topic of achieving cost-effective testing for serverless-based applications primarily using FaaS. We interviewed five software developers on their experiences with testing serverless-based applications. Through this data, we developed a survey containing questions based on the interviewees' responses. Results were then gathered through performing an inductive thematic analysis. The themes gathered highlighted the testing practices performed in the industry today for cost-effectiveness, factors that influence the cost of such testing practices, as well as the metrics for evaluating cost-effective testing.

The results around testing practices, one finding was the difficulty in emulating a serverless environment by testing locally. It was too resource-demanding and/or inaccurate. Surprisingly, our findings regarding unit and integration contradicted related work [9]. A majority of the interviewees had a preference for creating unit tests, even though integration tests are the recommended standard in related work when testing within serverless applications. We speculate a major reason behind this is due to product use-case and budget of the enterprise.

On the same topic of product and enterprise, the speculate they both generally are an important factor for testing serverless. The concern for budget, paired with which practices become more important for cost-efficiency depends on these. The choice of which serverless provider is also relevant, and how the later in development the harder switch it will be from one provider to another.

We found that participants varied in their opinions of what metrics were prioritized for their test cases towards cost-efficiency. We discovered that response time and resource allocation were the most common metrics impacting cost-effectiveness due to its direct influence with the serverless pay-per-use model.

Additionally, we brought to light that observability was a real issue in keeping track of these cost-effective metrics. Participants craved more fine-grained control and configuration that cloud providers abstracted away. Therefore, observation tools were a key factor in keeping track of these cost-effective metrics, such as resource allocation and response time.

In summary, our study explores the intricacies, difficulties, and tangible ways of achieving cost-effective testing of serverless-based applications. We highlight the importance of determining a definition of cost-effectiveness that is unique and dependent on product use-cases and enterprise KPIs. Furthermore, we highlight the factors influencing cost-effective testing and the key role that observability tools has in tracking and optimizing cost-effective metrics. Finally, we suggest ways of how practitioners can achieve a step closer to cost-effective testing based off these results.

A. Future Research

In regards to creating a direct follow-up to this thesis, we speculate there is more data to gather, especially on some of the topics of which were not *directly* prompted to the participants. These are some highlighted topics we believe should have more detailed study:

- The immediate concern with budget that practitioners might have for running serverless tests.
- Quantitative data on which cost-effective metrics are prioritized.
- The correlation of granularity as a design consideration to cost-effectiveness of a serverless based application.
- More verification as to the effectiveness of emulating serverless environments.
- Other types of testing in serverless besides unit and integration, as we did not find satisfactory data around E2E and performance tests.

We received answers on some of these topics listed. However, some of these topics were brought up often via tangents, e.g. granularity, or us strongly assuming based on context, e.g. budget concerns from serverless practitioner to practitioner. Also, as our survey had a low response rate, it could have been beneficial to have more accurate quantitative results.

Another suggestion to this topic in consideration to what was mentioned in the *Conclusion Validity* section, is to perform research about the behind-the-scenes into an organization's costs on testing serverless. This could be using the case-study methodology. This could potentially follow one or more enterprises on their transition into serverless.

References

[1] Microsoft Azure. What are private, public, and hybrid clouds?, n.d. Accessed on April 21, 2023.

[2] E. Bolscher. Predicting student success in higher education using machine learning. 2021. Master's Thesis.

[3] Jack Caulfield. How to Do Thematic Analysis — Step-by-Step Guide 038; Examples. *Scribbr*, 11 2022.

[4] Intel Corporation. What is private cloud?, n.d. Accessed on April 21, 2023.

[5] Daniela S. Cruzes and Tore Dyba. Recommended steps for thematic synthesis in software engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 275–284, 2011.

[6] Chief Editor. How to Do Thematic Analysis — Step-by-Step Guide 038; Examples. *BitsToGigs*, 3 2023.

[7] Simon Eismann, Diego Elias Costa, Lizhi Liao, Cor-Paul Bezemer, Weiyi Shang, André van Hoorn, and Samuel Kounev. A case study on the stability of performance tests for serverless applications. *Journal of Systems and Software*, 189:111294, 2022.

[8] Philipp Leitner, Erik Wittern, Josef Spillner, and Waldemar Hummer. A mixed-method empirical study of function-as-a-service software development in industrial practice. *Journal of Systems and Software*, 149:340–359, 2019.

[9] Valentina Lenarduzzi and Annibale Panichella. Serverless testing: Tool vendors' and experts' points of view. *IEEE Software*, 38(1):54–60, 2021.

[10] Garrett McGrath and Paul R. Brenner. Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 405–410, 2017.

[11] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[12] Jamil Mulani, Guotai Chi, and Jun Yang. Effects of the budgetary process on sme's performance: An exploratory study based on selected sme's in india. *Research Journal of Finance and Accounting*, 6:135–153, 2015.

[13] Eetu Rinta-Jaskari, Christopher Allen, Tamara Islam Meghla, and Davide Taibi. Testing approaches and tools for aws lambda serverless-based applications. 03 2022.

[14] Joel Scheuner. Performance evaluation of serverless applications and infrastructures. 2022.

[15] Joel Scheuner, Rui Deng, Jan-Philipp Steghöfer, and Philipp Leitner. Crossfit: Fine-grained benchmarking of serverless application performance across cloud providers. 2022.

[16] Stefan Winzinger. Towards coverage criteria for serverless applications. In *11th Central European Workshop on Services and their Composition (ZEUS)*, 2019.

[17] Stefan Winzinger and Guido Wirtz. Coverage criteria for integration testing of serverless applications. 06 2019.

[18] Stefan Winzinger and Guido Wirtz. Model-based analysis of serverless applications. 05 2019.

[19] Stefan Winzinger and Guido Wirtz. Applicability of coverage criteria for serverless applications. pages 49–56, 08 2020.

[20] Yunjing Xu, Zachary Musgrave, Brian Noble, and Michael Bailey. Bobtail: Avoiding long tails in the cloud. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013*, Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, pages 329–341. USENIX Association, 2013.

Appendix

TABLE IV
SURVEY QUESTIONS

RQ	Survey Question	Format
	1. How big is the enterprise that you work for?	Multiple Choice
	2. What is your development experience in years?	Multiple Choice
	3. What is your testing experience in years?	Multiple Choice
N/A	4. What is your primary software development methodology, a.k.a. in what way does your enterprise work?	Free Text
	5. What serverless provider(s) do you currently use in your enterprise?	Multiple Choice
	6. What type(s) of product are currently using serverless at your enterprise?Describe the purpose or domain of the product(s)-under-test.	Free Text
	7. What is your role within your enterprise?	Free Text
RQ1	8. When writing test cases for a serverless product, how does the quantity of test cases created change for the following types of test cases from testing of traditional products?	Free Text
RQ1	9. Do you find it challenging within serverless when it comes to...	Multiple Choice (Grid)
	9.1 ...testing performance of a specific serverless component?	
	9.2 ...testing integration of multiple serverless components?	
	9.3 ...creating test environments that accurately reflect the production environment?	
	9.4 ...knowing which testing methods are cost-effective?	
	9.5 ...setting up CI/CD pipelines for automated testing?	
RQ1	10. How do you perform testing in serverless? For example, what testing practices and types of testing do you apply?	Free Text
RQ2	11. In a serverless context, can you provide an example of a situation where you identified a cost inefficiency in testing and how you addressed it?	Free Text
RQ3	12. How do you evaluate the success of your testing efforts?	Free Text
RQ2	13. Which of these concepts do you believe help with cost-efficiency within serverless applications? List in priority.	Multiple Choice
	13.1 Centralised logging system	
	13.2 Organisational maturity	
	13.3 Product maturity	
	13.4 Parallel testing	
	13.5 Configuration of CI/CD pipelines for automated testing	
RQ3	14. Regarding the above question, do you have any comments on why a strategy was helpful or not?	Free Text
RQ2	15. What other strategies improve the cost-effectiveness of your testing efforts?	Free Text
RQ3	16. For your serverless components, is there a specific level of code coverage that you aim to achieve? If so, what is that level?	Multiple Choice
RQ3	17. What decisions lead to this value?	Free Text
RQ1	18. Does your serverless functions have tests that cover the following?	Multiple Choice
	18.1 Test for All-restored	
	18.2 Test for All-failed	
	18.3 Test for All-max-time	
	18.4 Test for All-min-time	
	18.5 In parallelism, test read and write accesses to the data storage in all possible orders	
RQ2	19. Do you know of any emerging trends or technologies in serverless testing that may impact cost-effectiveness? If not, is there anything you'd wish for?	Free Text