Problems in Microservice Development: Supporting Visualisation

Oscar Manglaras
University of Adelaide
Adelaide, Australia
oscar.manglaras@adelaide.edu.au

Alex Farkas
University of Adelaide
Adelaide, Australia
alex.m.farkas@gmail.com

Peter Fule
Swordfish Computing
Adelaide, Australia
peter.fule@swordfish.com.au

Christoph Treude
The University of Melbourne
Melbourne, Australia
christoph.treude@unimelb.edu.au

Markus Wagner
Monash University
Clayton, Australia
markus.wagner@monash.edu

***Abstract*—In microservice architectures, developers can face significant problems understanding the structure of the system and how the different microservices interact. This difficulty results from the distributed nature of the system, and the abundance of inter-service communication within the architecture. We want to determine if network visualisations can address these problems given their ability to convey complex topologies. However, to identify what architectural characteristics should be visualised, and how this should be done, we must first determine the needs of microservice developers. This paper identifies and presents the impact and frequency of problems faced by a cohort of microservice developers using the results of an online survey. Our findings indicate that the most frequent problems were topology related and the highest impact problems were those related to system faults and data structures. Our results support the use of network visualisations to address microservice development problems and provide context that will allow future visualisations of any type to better address these problems.**

***Index Terms*—Microservices, development problems, documentation, microservice visualisation.**

I. Introduction

A microservice architecture is a type of distributed software architecture composed of multiple, small, independent services, called 'microservices', which each have a single responsibility, but which together form a full application. Microservices do this by communicating with each other over lightweight network protocols [1]. Microservice architectures can allow for an agile development strategy and a flexible, scalable system. With only a single responsibility, each microservice can be kept small and relatively simple [2]. However, by simplifying each service, the complexity is moved from the services themselves and into the communication architecture [3]. Understanding the resulting topology of microservice communications can be vital when creating, maintaining, and debugging services [4].

Network visualisations are one way to visualise and document communication topologies [5]. Hence, they can also be used to document the communication topology of a microservice architecture. It stands to reason that other types of visualisation could be viable as well. Visualising microservice architecture topologies is not new, there are various academic and commercial tools that have chosen to do so [6]. However, there is little research into what specific visual elements and interactive features microservice developers would find useful during their development tasks (Section II). Our ultimate goal is to evaluate whether visualisations are an effective way to support microservice development, and if so, which development problems they are well suited to solve. However, before doing so, we want to gain insight into what these problems are, how frequently they are encountered, and how impactful developers find them.

In this paper we present the methodology and results of an online survey where we identify the impact and frequency of various problems faced by developers during their development tasks. We provided this survey to developers at a company called Swordfish Computing. They have multiple teams that develop systems using microservice architectures. These systems primarily use the publish/subscribe communication paradigm [7]. We also asked developers to outline any existing documentation tools they use and what their ideal tool would be.

We found that the most impactful and frequent problems relate to the data structures used by microservices and channels, the communications between microservices and channels, and the causes and effects of microservice faults. For documentation tools, participants reported both textual and graphical tools are used, with markdown and draw.io being the most common. Participants' ideal documentation tools showed several common themes; visualisation and interactivity, minimising effort, and coupling the documentation with code. Based on these results, network and trace visualisations would address the primary concerns of microservice developers, and developing ways to visualise data structures could be beneficial. Additionally, a developer's ideal visualisation would be interactive and automatically generated from the source code of the microservice.

2832-6555/23/$31.00 ©2023 IEEE
DOI 10.1109/VISSOFT60811.2023.00017

II. Related Work

When identifying related work we looked for any research that reported on challenges and problems faced by microservice developers. We also searched for any evaluations of microservice visualisation designs which may have uncovered additional development tasks.

A. Challenges and Problems in Microservice Development

Viggiato et al. [8] did a survey of 122 developers from various microservice related online communities. They performed a mapping study of microservice developers to identify common microservice advantages and challenges. Four advantages and four challenges were identified. Participants were asked to rate the importance of each advantage and challenge from 1 (very important) to 4 (not important). These challenges, along with the mean of their responses, were: *testing the whole system* ($\bar{x} = 2.259$); *complex distributed transactions* ($\bar{x} = 2.301$); *service faults* ($\bar{x} = 2.553$); *expensive remote calls* ($\bar{x} = 2.577$). Additionally, 62% of respondents said they worked with RESTful communication architectures.

Ghofrani and Lübke [9] conducted a survey with 25 microservice developers to identify challenges and concerns in the design and development process. Some of the challenges mentioned were "too many repositories to maintain"; "networking between dockers"; and "debugging a microservice that relies on other services". Another question was "Which notation(s) do you use to describe your architecture?". The answers were *graphical modelling language* (35%); *textual modelling language* (9%); *domain modelling language* (17%); and *none* (39%). Additionally, they stated that "none of the participants named any tool, technique, and [sic] framework for modeling their [architecture]".

Baskarada et al. [10] interviewed 19 developers to identify the opportunities and challenges that come from adopting a microservice architecture. The identified challenges were: *lack of relevant skills*; *existing software-as-a-service and commercial-off-the-shelf software*; *organisational culture*; *governance*; *organisational structure*; *monolith refactoring*; *data management*; *orchestration and choreography*; *testing*; and *performance*.

Zhou et al. [11] conducted a series of interviews with 16 participants across 12 companies to investigate how system faults are handled. Nine participants stated that runtime verification and debugging were the main maintenance challenges they faced, and that they were heavily dependent on monitoring and tracing infrastructure. Debugging was a major challenge for "almost all of the participants". The authors identified numerous types of system faults and debugging approaches. Of these approaches, trace and log visualisations were identified as a particularly useful tool by developers. To test their findings, they presented six graduate students with a series of system faults for a familiar microservice system, presented using text logs, visualised logs, and visualised traces. They found that most faults could benefit from trace visualisations.

Mayer and Weinreich [12] developed a dashboard to display information about a microservice architecture. To develop the features for their dashboard, they ran a study that surveyed and interviewed 15 participants from 12 companies who had experience with microservices. Their study found that the participants considered the most important information to be the following: *service APIs*; *service versions*; *number of service instances*; *system metrics (CPU, memory, error rates, etc)*; and *service interactions and dependencies*.

Engel et al. [13] conducted interviews with participants from five microservice projects using a range of technologies and microservices. The projects ranged from two microservices, to approximately 50. The challenges discussed by experts of each project were: *keeping an overview of the whole architecture* (3 projects); *service boundaries* (2 projects); *refactorings that effect multiple services* (2 projects); *changes to a service that affect other services* (2 projects); *cylcic dependencies* (1 project); *data consistency* (1 project); and *testing* (1 project).

B. Microservice Visualisation Evaluations

In their Masters thesis [14], Frisell outlined an experimental research study where they aimed to evaluate the effectiveness of a microservice visualisation tool for the purposes of debugging system faults. They worked to develop the tool with a company, and described an interview study conducted by the company regarding problems faced by their microservice developers. Their findings were that developers: "wish for a tool to make it easier to find the root cause to service incidents as well as who is responsible"; "have problems understanding who is responsible for bad/broken/wrong data"; and that they "look at graphs from other services to understand the problem".

C. Limitations

These existing studies tend to focus on high level issues such as developer inexperience, deployment co-ordination, and architectural complexity which does not help to determine what specific elements or relationships in a microservice architecture developers find challenging to understand. Additionally, those that do, such as [12], [13], lack details about the frequency or relative importance of problems.

III. Methodology

In this section we will detail how and why participants were recruited, how the survey was designed and distributed, and and how we analysed our results. In this and the following sections we use *italics* to denote preset options we provided or categories we have created, while *"quoted italics"* are used to denote raw user input.

We have made the full survey available online[1].

A. Recruitment & Incentives

Participants for the survey were recruited from the company Swordfish Computing, which is co-funding this research. This cohort was chosen because they develop multiple different microservice systems that have real-world, practical applications,

[1] https://doi.org/10.5281/zenodo.8245342

TABLE I: List of microservice development tasks from part 2 of the survey.

ID	Category	Short Name	Question
T1	Topology	microservices - incoming services	What microservices are sending messages to this microservice?
T2	Topology	microservices - outgoing services	What microservices are receiving messages from this microservice?
T3	Topology	microservices - connected channels	What channels is this microservice published/subscribed to?
T4	Topology	channels - connected microservices	What microservices publish/subscribe to this channel?
T5	Topology	microservices - connected brokers	What message brokers is this microservice communicating with?
T6	Topology	brokers - connected brokers	What message brokers is this message broker communicating with?
T7	Topology	microservices - shortest path	What is the shortest communication path between two microservices?
T8	Topology	microservices - communication type	What type of communication is this microservice using? (pub/sub; REST; etc)
P1	Deployment	microservices - current hardware	What host/hardware is this microservice running on?
P2	Deployment	hardware - running microservices	What microservices are running on this host/hardware?
P3	Deployment	microservices - current instances	How many instances of this microservice are running at any one time?
D1	Data & Definition	microservices - data structures	What microservices use this data structure?
D2	Data & Definition	channels - data structures	What is the data structure of messages in this channel?
D3	Data & Definition	channels - structure of name	How is the channel string structured?
D4	Data & Definition	microservices - purpose	What is the purpose of this microservice?
D5	Data & Definition	channels - purpose	What is the purpose of this channel?
D6	Data & Definition	microservices - type	What type of microservice is this?
V1	Development	microservices - maintainers	Who maintains this microservice?
V2	Development	team - maintained services	What microservices does this person/team maintain?
V3	Development	microservices - languages and libraries	What languages/libraries does this microservice use?
S1	Scenario	microservice faults - affected services	If this microservice goes down what other microservices will be affected?
S2	Scenario	broker faults - affected services	If this message broker goes down what microservices will be affected?
S3	Scenario	microservice faults - causes	If this microservice is not functioning correctly what other microservices could be the cause?

and because our existing association with the company gave us easy access to the participants.

To increase visibility, the survey was advertised to their microservice development teams over Slack, the company's preferred messaging software. The survey was sent to the entire population of microservice developers at Swordfish Computing. At the time of our research in November 2022 this consisted of 29 people.

To encourage participation and improve the external validity of our study, participants were given an indirect monetary incentive to participate because Swordfish Computing allowed the time spent completing the survey (~20 minutes) to be billed as part of their wage. Participants were also told that the results of the survey would be used to develop documentation tools to help them with their development work, which may have also acted as an incentive. Finally, participants were assured that responses would be kept anonymous.

B. Distribution

We chose to distribute the survey online over Google Forms. We concluded this was an effective method of distribution because our potential participants were experienced at computer use, had access to internet devices through their employment, and we had been informed that respondents were interested in the topic of our research [15, Ch.4]. It was also a simple way to meet our privacy and security requirements. The eligible individuals were sent an email with a link to the survey and given two weeks to respond.

TABLE II: List of demographic questions.

Question	Type
How many years have you worked in software development?	short answer
How many years have you worked in microservice development?	short answer
What roles have you had in your team during microservice development?	short answer
What development work do you do in relation to microservices?	checkboxes and short answer

C. Survey Design

1) Demographics: The demographic questions are shown in Table II. The question *What development work do you do in relation to microservices?* took the form of checkboxes where participants could select multiple answers, as well as a short answer box where participants could enter any type of work that was missed by the options. The default options were: *Backend Implementation (programming)*; *API Development*; *Documentation*; *Deployment*; *Orchestration*; *Deployment*; and *Architecture Design*.

2) Problems in Microservice Development: This section attempted to answer the question, *what problems are developers experiencing when developing microservices?*. We presented the participants with a large list of possible tasks (see Table I) and asked participants to rate both the impact of the task on their development time, and how frequently they face the problem along an ordinal scale. The available responses for impact were: 'no impact', 'small impact', 'moderate impact',

'large impact', and 'massive impact'. The responses for frequency were: 'never', 'yearly or less', 'quarterly', 'monthly', 'weekly', and 'daily'. For the rest of the paper we use 'problem' and 'task' interchangeably.

TABLE III: The glossary given to participants.

Term	Definition
Pub/Sub	A communication architecture where microservices publish and subscribe to messages from specific channels rather than messaging each other directly.
RESTful	A communication architecture where microservices communicate directly over HTTP requests.
Channel	Also called topics. Microservices publish or subscribe to messages from specific channels in Pub/Sub architectures.
Message broker	You may also know this as a message bus. The application that handles the routing and forwarding of messages between microservices.
Microservice type	Some projects group or design their microservices to fit into certain categories such as functional; infrastructural; web server; database; etc.
Documentation tool	Any resource that presents information about the microservice architecture you are working on. This can include text-based documentation; visual graphs; interactive software; system logs; etc.

We developed this list based on the existing studies into microservice challenges (Section II) and on feedback from our pilot tests. A glossary (Table III) was included to reduce the risk of participants misunderstanding our terminology. In addition, we gave participants the chance to manually enter any problems they faced which we had missed.

To reduce the risk of fatigue in participants, we split the problems into categories [15, Ch.3]. Each topic was given an individual page in the survey where participants were asked to rate the impact and frequency of each task. For each topic a short answer response box was provided for participants to enter additional problems.

3) Documentation Tools: The final section asked the user to detail any existing documentation tools they use, and to describe what their ideal documentation tool would be. Our definition of documentation tool can be found in Table III. We used an intentionally broad definition because we did not want to risk participants failing to write down a tool in case they were unsure if it met our classification.

D. Data Vetting

As participants have access to sensitive data, Swordfish Computing required that the responses be vetted to ensure nothing was inadvertently leaked through the survey. This process was done by a member of the research team who was also an employee at Swordfish Computing. The only times this vetting occurred was in the short-answer response questions when participants mentioned the name of specific projects or proprietary internal tools. In these cases the names were substituted for generic terms like, 'internal documentation tool'.

E. Data Analysis

1) Problems in Microservice Development: To analyse the results of the tasks in Table I we visualised the answers in the form of a horizontally stacked bar chart. In addition we assigned a numerical value to each answer on the ordinal impact and frequency scales. The impact scale went from 0 (No Impact) - 4 (Massive Impact). The frequency scale went from 0 (Never) - 5 (Daily). Using these numerical values, we could then calculate statistical values such as the mean, median, and mode.

2) Documentation Tools: For the question, *What existing documentation tools do you use?* we went through the responses and counted every tool mentioned. We then categorised these tools as: *textual*; *graphical*; *interface file* (such as an AsyncAPI[2] JSON file); and *debugging tool*. The main reason for this categorisation was to make it easier to compare the number of participants using textual vs graphical/visual tools, which may have implications for future visualisation research. An interface file is technically textual, but we have chosen to categorise them differently because textual documentation (i.e. markdown) is designed to be read by humans, whereas an interface file (i.e. AsyncAPI) is designed to also be read by machines.

The question, *what would be your ideal documentation tool?* was analysed using a thematic analysis approach as described in [16]. For each response we assigned codes that represented the facts and opinions within the answer. This consisted partially of simple codes that directly reflected the contents of the response. For example the response: *"a tool that is completely automatic and does not require any work to maintain"* would be assigned a code "desires automatic generation". However, codes could also represent higher level inferences, for example *"automated, insightful, visual"* implies that the individual sees visualisation as an insightful method of documentation. After refining the codes, we searched for recurring themes in the codes and responses, which we present in Section V-B2.

IV. Demographics

In this section we have included the demographic information of the participants to help characterise our data. The list of demographic questions is presented in Table II. Some data cleaning was necessary for the short answer response questions.

A. Data Cleaning

The 'years of experience' answers needed to be converted into numerical format; any responses such as *"2 years"* were changed to *"2"*. Three responses were in the format *"< 1"* or *"less than a year"*; these were changed to *"0"*.

Responses to the question, *What roles have you had in your team during microservice development?* needed to be cleaned. For example, *software developer*, *developer*, and *software development* were all combined into *developer*. Similarly, *tech*

[2]https://www.asyncapi.com/docs

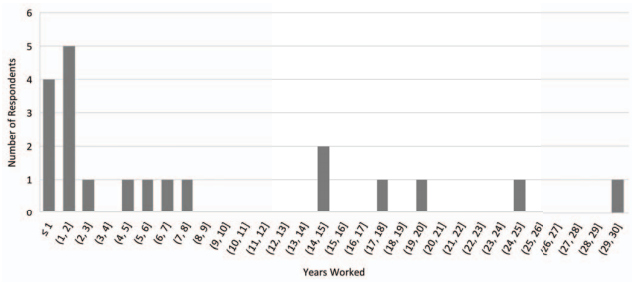

Fig. 1: Years worked in software development.

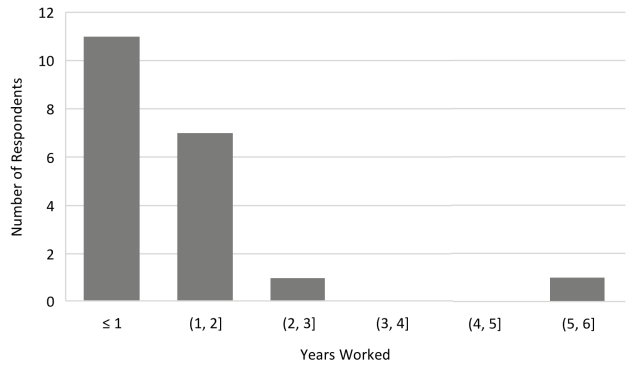

Fig. 2: Years worked in microservice development.

lead and *senior/tech lead* were combined into *tech lead*. Responses that reflected a misunderstanding of the intent of the question and instead listed tasks that they did were instead categorised under *other*. Participants were allowed to list multiple roles.

Responses to the question *What development work do you do in relation to microservices?* did not need to be cleaned as only one participant submitted an additional task this short answer question.

B. Experience

Fig. 1 shows a histogram displaying the years of experience in software development for the participants of the survey. This shows that the company we surveyed has many junior developers working with microservices, and hence that our results will be weighted towards the opinions of junior developers. The mean for this data is 8.15 years, and the median is 5 years.

Fig. 2 shows a histogram displaying the years of experience in microservice development for participants of the survey. This chart is more heavily skewed towards fewer years with 18 (90%) participants reporting ≤ 2 years of experience. Hence, our results are weighted towards the opinions of developers inexperienced in microservice development. The mean for this data is 1.525 years, and the median is 1 year.

C. Roles

Table IV shows the roles that participants reported. Participants were able to specify multiple roles, explaining why the total number of responses for all roles exceeds 100%. The two *other* responses did not state a role, but instead stated specific tasks that they completed in their team, for example one responded with: *"implementation of features for multiple services"*, while the other responded: *"only minor updates to existing services"*. The first individual later stated they were involved in *backend implementation* and *documentation* and had less than a year of experience. The other stated they were involved in *backend implementation*, *documentation*, *deployment*, and *architecture design* and had 20 years of experience in software development.

TABLE IV: Self-described development roles of participants. Each participant can hold multiple roles.

Role	#	of total
Developer	10	0.50
Software Engineer	5	0.25
Tech lead	4	0.20
Junior Software Engineer	3	0.15
Data Scientist	1	0.05
Design and DevOps	1	0.05
Product Manager	1	0.05
Other	2	0.10

TABLE V: Type of development work done by each participant. Each participant can select multiple types. *Frontend Development* is the sole user-submitted work type.

Type of Work	#	of total
Backend Implementation (programming)	19	0.95
Documentation	18	0.90
Deployment	16	0.80
API Development	10	0.50
Architecture Design	13	0.65
Orchestration	11	0.55
Frontend Development (user submitted)	1	0.05

These ambiguous responses were a fault in our question design which did not get flagged in our pilot studies. In hindsight a checkbox+short answer style of question as used by the next question (*What development work do you do in relation to microservices?*) might have been preferable. A list of team roles could have been sourced by our contact at the company prior to the survey distribution.

Table V shows the types of development work undertaken by participants. *Backend Implementation* was the most common response with a 95% positive response rate. Documentation was also very high with a 90% positive response rate, which implies that the documentation process is the responsibility of the whole team rather than just specific individuals. This number is encouraging for us as the widespread documentation experience increases the value of responses to our documentation tool questions. It may be notable that of the two participants who did not report having worked in documentation, one did not respond to the 'ideal documentation' question, and the other focused on the accessibility of the documentation, rather than any aspect of the documentation

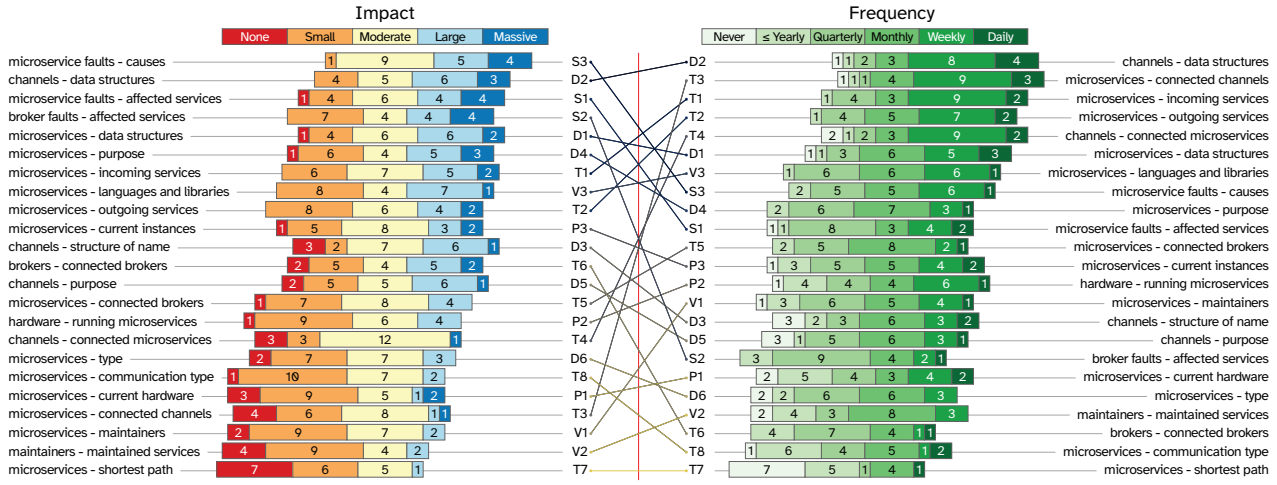

Fig. 3: The impact and frequency responses for the problems in Table I. Each horizontally stacked bar chart shows the number of people who selected each available answer for each problem. Each column is sorted by the mean, with the same problem linked by a line through the centre.

process itself.

V. Results

There were 29 developers who met the requirements to participate in the survey. In total we received responses from 20 individuals, 69% of the full cohort.

A. Problems in Microservice Development

Fig. 3 contains the results for our list of problems in Table I. The bar charts have been translated to create a diverging scale that makes it easier to compare the relative number of responses above and below the centre. The impact column is centred on the *moderate* bar, which represents the middle or 'neutral' option, while frequency is centred on *monthly*.

In Fig. 3, the point along the red separating line that the links cross represents the halfway point between the position of the problem in both columns. We can use this crossing point as a primitive measure of the combined impact and frequency of that problem. Using this measure, links that cross the line closer to the top of the chart have a higher combined impact and frequency than links that cross lower in the chart. We have coloured the links using a $blue \rightarrow yellow$ colour scale to signify the order they cross the line.

According to this metric, problem D2 (*What is the data structure of messages in this channel?*) had the highest combined impact and frequency. Participant responses indicated the need to frequently identify the data structure for channels. Of these, many did so weekly, and all but one response answered that doing so had a moderate or higher impact on their work. D2 was followed by a cluster of S3 (*If this microservice is not functioning correctly what other microservices could be the cause?*), T1 (*What microservices are sending messages to this microservice?*), D1 (*What microservices use this data structure?*), and then S1 (*If this microservice goes down what other microservices will be affected?*) and T2 (*What microservices are sending messages to this microservice?*).

Topology problems, specifically T1–T4 had high frequencies, with nearly 75% of participants selecting *monthly*, *weekly*, or *daily*. For problems T1, T3, and T4, over 50% of participants who rated the problem dealt with the issues weekly or daily.

All of the data & definition problems and scenario problems had medians of at least 2, meaning at least 50% of respondents rated them as moderately impactful or greater. This makes them the most consistently impactful categories in the survey. Scenario problems were especially impactful, all three of which had impact means in the top four. In contrast, development and deployment problems tended to be less impactful, with most problems showing that $\geq 50\%$ of respondents thought the problems had small to no impact. The two exceptions were problem P3 (*How many instances of this microservice are running at any one time?*) and V3 (*What languages/libraries does this microservice use?*), which had a more even spread of responses above and below moderate impact.

The least impactful and frequent problem was T7 (*What is the shortest communication path between two microservices?*), which was rated noticeably lower than any of the other problems. It was the only problem where the mode was 'none' for impact and/or 'never' for frequency. More than any of the other listed problems, it seems like finding the shortest path between two microservices is not something the participants found relevant to their work, though even here there were six people who rated the impact as moderate (5) or large (1), and 5 people who encountered the problem weekly (4) or daily (1).

These results should be interpreted cautiously due to the small sample size and relatively large range and standard

deviation of responses. The variation in the means and medians is also relatively small, so we do not wish to place undue importance on the exact values and orderings of the problems.

TABLE VI: List of additional problems submitted by participants.

No.	Problem
01	Making sure messages are being sent in the correct data format. (weekly)
02	Development across environments with different network access (open; air-gapped). (quarterly)
03	Identifying breaks in a communication chain.
04	Identifying clusters of message brokers.
05	Understanding microservice interfaces.
06	What data flows where and how it is transformed.
07	Identifying bottlenecks in data transformations.
08	Solving connectivity and permission related issues.
09	Will the microservice run/compile on/for specific processor architectures.
10	What is the status of the microservice? (dependency failures; degraded nodes; hardware load; etc)
11	Handling varying message bus overhead of maintaining channels.
12	Debugging and integration testing.
13	Are services stateful and how is state maintained between restarts?
14	How to keep services running reliably and how to idenitfy when they are not.
15	Managing consistant units of measurement for values.
16	How to design a microservice that will tollerate brokers/busses/dependencies going offline.

1) Additional development problems: Table VI contains a list of additional problems identified by participants. Some of these problems were stated by the same individual, but no problem was mentioned by multiple individuals. This may imply that we successfully listed the common problems faced by microservice developers, but may otherwise be due to participants' unwillingness to write more than necessary. Participants were asked to state the impact and frequency with their answers. However, only two participants actually did so, and only the frequency. These two problems have been listed at the top of Table VI and the frequency has been placed in brackets.

Many of these problems focus on live runtime information and debugging, such as problems 03, 08, 10, 12, and 14. Dynamic information problems are perhaps underrepresented in our survey. Problem 01 is marked as *weekly*, making it the only listed problem that we can be confident the respondent encounters frequently. This problem is also very similar to some of the data & definition problems from the main list, especially D2 (*What is the data structure of messages in this channel?*), which was one of the most impactful and frequent problems we listed. However, problem 01 is focused on validating, instead of just identifying, the data format.

This list represents problems that participants did not associate with any of our survey categories. A future study looking to build on this research could benefit from incorporating some of these problems.

B. Documentation Tools

TABLE VII: Documentation tools in use.

Tool	#	Category
markdown	10	textual
draw.io	7	graphical
internal diagram generation tools	4	graphical
asyncapi	3	interface file
confluence	2	textual
IDE debugger	2	debugging tools
logs	2	debugging tools
MS word	2	textual
asyncapi->markdown converter	2	textual and interface file
comments	1	textual
sphinx	1	textual
latex	1	textual
git diffs	1	debugging tools
notepad	1	textual
files	1	textual
none	1	none

TABLE VIII: Documentation tools reported as being used by category.

Category	#
textual	13
graphical	10
interface file	3
debugging tool	2
none	1

1) Existing Documentation Tools: In total, there were 15 different tools reported by participants. These tools are listed in Table VII, along with the category the tool fits within. As most participants listed multiple tools, the total number of responses for each tool sums to greater than 100%. In total 18 (90%) of the participants answered this question.

The tool *"internal diagram generation tools"* is a catch-all for proprietary tools used by participants that, due to confidentiality, cannot be specified in detail. One participant stated that they did not use any documentation tools in spite of their answer earlier in the survey that they participated in documentation work. It is possible that this person creates documentation, but does not use it. IDE debuggers and log files did not receive many responses despite meeting our definition of documentation tool. It is possible that most participants simply did not consider debuggers or system logs to be documentation tools despite our definition.

Table VIII lists the number of responses for each category. These response numbers are often less than the sum of responses for each individual tool in that category, as many participants listed multiple tools from the same category in their answer. Grouping responses in this way shows us that over 50% of respondents utilised textual or graphical documentation, with textual documentation being the most common with 13 (72%) responses, while graphical had 10 (56%) responses. Markdown[3] was the most common textual documentation tool, and the most common tool in general, with 10 (56%) responses. Draw.io,[4] a manual tool for drawing

[3]https://daringfireball.net/projects/markdown/
[4]https://www.diagrams.net/

graphs, was the most common graphical tool, and second most common tool overall, with seven (39%) responses.

Participants seem to predominantly use manual tools for creating documentation (markdown and draw.io), however there were two participants who mentioned a tool that converts asyncAPI files to markdown, and four participants spoke of an internal tool that could generate graphs. It could be that these tools are immature and therefore do not see wide adoption within the company. Alternatively, it could be these tools are managed by very small number of individuals, but the outputs are distributed to other participants as markdown or confluence documents, and who are hence unaware of the source. However, given that 90% of participants answered that they were involved in documentation work (Table V), the majority would presumably still do manual documentation.

2) Ideal Documentation Tools: For this question we used thematic analysis methods [16] to identify three common themes in the responses. Unfortunately, only 13 of the participants elected to answer this question.

Only one of the 13 responses did not fit into any of these three themes. This individual stated they were unsure what their "single" ideal tool would be, as *"its always a balance whether documenting for developers or end users"*. The survey question could perhaps have been improved to clarify to participants that they were allowed to specify various behaviours even if, in practice, these behaviours would not be part of the same tool.

Theme 1: *Visualisations and interactivity* refers to the desire for a new tool for displaying information in the form of visualisations, or some other interactive format. We saw this theme in six (46%) of our responses.

Responses with this theme can generally be split into those which specified the type of data they wanted represented, and those that specified how the data should be represented.

Visualisations were the most common way of representing data, mentioned directly by three responses. One response went into particular detail about why they wanted visualisations: *"Any tool that can automate manual work, and enable easy visualisation of a repository/service gets my vote as visualisation helps me get up to speed with existing services and work out how they work, where data is flowing, and what/where I might need to check when debugging."* Another stated that: *"a visualisation tool similar to those used in manufacturing industries to see the status of the plant, could be beneficial for large scale microservice deployment"*. The third simply said they wished for their ideal tool to be: *"automated, insightful, visual"*.

Well-known visualisations, such as node-link diagrams, can be an effective way to visualise the connections and topology of a network [5]. The high frequency of topology problems that respondents reported (Fig. 3) provides a potential explanation for why several saw visualisation as part of their ideal tool.

Other respondents did not specify visualisation, but still implied a desire for a GUI tool for displaying information. One participant responded with: *"a web based interactive tool that can search for services, check that their interfaces align, and allow one to build a deployment skeleton for a series of brokers and services"*. While another stated that: *"A dashboard which can tap into a microservice architecture deployment and present handy information such as services failing over, using high memory, system logs for each service, attachable shells for debugging, accompanied by plugins which support looking into mqtt messages on a network (topic, sender & receiver of messages, message content)"*.

Theme 2: *Minimising effort and the desire for automation* refers to the desire for a tool to be 'easy to use', and the perception that automation would help to accomplish this goal. This can be found in six (46%) of our responses.

Most participants did not specify the level of automation they desired, simply that the process should be *"easy to use"* and *"enable easy visualisation"*. One expressed a desire for the tool to completely automate the process of automation: *"[a tool that] scans all code and generates all documentation with zero effort from the developer"*. Another was more measured, conceding that the process would be automatic only after the initial setup: *"a tool that is completely automatic and does not require any work to maintain once it understands my project (initial config)"*.

These responses imply that most participants that discussed automation primarily care about reducing the burden of documentation and do not have a strong preference for the specific approach used. Hence, participants see automation as a way, perhaps even *the* way, to reduce their documentation burden.

Theme 3: *Documentation should be tightly coupled with code* refers to the idea that documentation should be stored with, written as, or generated from, the source code of a microservice. We saw this theme across four (30%) of our responses.

This concept is not new; documentation approaches such as *documentation as code* [17], and *continuous documentation* [18] promote the coupling of documentation with code. The claimed benefits of these approaches are that they help to keep documentation up-to-date, allow developers to use their familiar development tools for documentation, and separates the content of the documentation from the presentation [18].

One response directly stated a desire for continuous documentation principles, saying that it: *"would be great if [the automated documentation tool] can be automatically executed as part of the CI/CD pipeline so that documents can also be reviewed, approved and in lockstep with the software release"*. However, most of the responses did not express interest in the full benefits of *documentation as code* or *continuous documentation*, instead focusing on a subset of benefits. One response simply expressed the desire for documentation to be stored in the same location as the code: *"repository READMEs is probably the most accessible as its it [sic] usually in the same repository as the microservice's source code"*. Another response highlighted version control and document generation: *"[documentation should be] created as plaintext (like markdown), stored in version control with the code so it's easy to update and can generate good looking static documents"*. These developers may only care about these features, or they

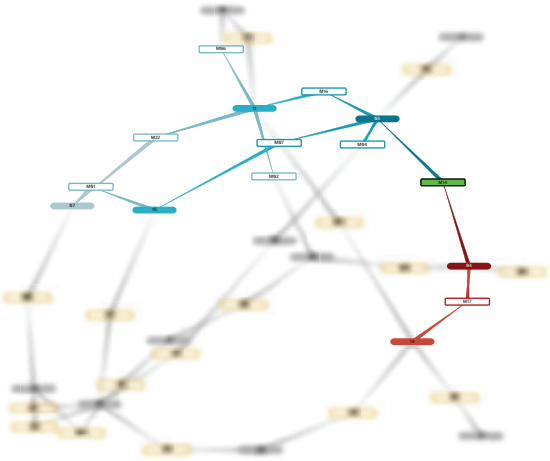

Fig. 4: A network visualisation that shows the connections between microservices (unfilled nodes) and channels (filled nodes). Uses edge highlighting to display the reachability of nodes to (red) and from (blue) the selected (green) node. Unreachable nodes have been blurred. For grayscale reference: the three nodes on the lower right are red, the upper nodes are blue, and the one linking the red and blue is green.

may be unfamiliar with other *documentation as code* and *continuous documentation* concepts.

There is also overlap with theme 2 (minimising effort and the desire for automation) when it comes to automatically generating documentation from code in order to reduce the burden on developers. This can involve converting a plain text artifact like markdown into a rich text document, as shown in one of the responses above. Another respondent, whose response also fell into theme 1, proposes going a step further and *"[scan] all code and [generate] all documentation with zero effort from the developer"*, which removes the separation between code and documentation entirely.

VI. Implications for Visualisation

Half of the participants reported that they use graphical visualisation tools, showing that developers are already actively choosing to use visual forms of documentation. The common theme of *visualisations and interactivity* further supports the idea that developers want to use visualisations to support microservice development. This theme also shows an interest in interactive features, such as the ability to search and filter microservices, and to perform monitoring and debugging tasks. The theme of *minimising effort and the desire for automation* shows that any attempt to visualise microservices should automate as much of the visualisation process as possible. If we also take into account theme 3, *documentation should be tightly coupled with code*, then the ideal microservice visualisation would be automatically generated simply by scanning the source code.

The topology problems that participants encounter most frequently involve identifying the connections between adjacent microservices and channels. This information is the kind that network visualisations are well suited to presenting [5], which supports their use with microservice architectures. However, most of the high frequency topology questions related to connections between adjacent nodes rather than paths through the topology, hence an adjacency matrix may be better suited for these tasks than a node-link diagram. The cohort we surveyed primarily develop microservice architectures that use publish/subscribe communication. However, they still reported frequently needing to identify the connections between microservices, instead of just connections to channels. This implies that network visualisations without channel information would still be useful in publish/subscribe architectures where no direct communication between microservices takes place.

Microservice fault problems were among the most impactful, both identifying the cause of a fault, and identifying which other microservices a fault could affect. Visualisations that support solving these issues would be useful to developers. Zhou et al. [11] found that trace visualisations are effective at helping to find the source of faults, but we are not aware of research into visualising the services affected by a fault. Network visualisations could be used to address this issue by calculating and highlighting the reachability of microservices from the faulty service. In contrast, researchers may not want to devote significant resources into visualising the developers or maintainers of microservices, nor on providing tools to find the shortest path between services. Both of these problems had among the lowest impact and frequencies in our results.

The data formats used by microservices and channels were one of the most frequent and impactful problems in the survey. However, we are not aware of any attempts to incorporate this information into a microservice visualisation. Finding effective ways to integrate data formats into network and trace visualisations could be a valuable avenue of research. Integrating live data such as hardware load and outages would also address problems and requests of developers. This is already a common feature among existing academic [12], [14] and commercial tools[5][6], so researchers have existing designs they can use as inspiration.

Fig. 4 is an example we created of a visualisation that attempts to address some of these implications. It is a network diagram that shows both microservices and channels as different types of nodes, and uses tapered lines as directed edges [19]. It uses highlighting to visualise the reachability of nodes to and from the selected node. This visualisation addresses the frequent topology problems and the impactful fault problems. However, further research is required to evaluate the effectiveness of these visual encodings, as well as to investigate other types of visualisations, including non-network visualisations.

[5]https://www.weave.works/oss/scope/

[6]https://www.dynatrace.com/platform/applications-microservices-monitoring/

VII. Limitations and Threats to Validity

Because the participants of this survey all came from a single company, the results of this survey cannot be used to make generalisations about the wider population of microservice developers. This, combined with the high standard deviations for the problem questions, means we cannot show a high degree of statistical confidence in our results. However, all of our participants have experience developing practical, real-world microservice applications, and the types of problems we asked are general enough to be common across the industry. Our results also showed parallels with the related studies in Section II, such as the impact of system faults, and challenges related to the connections and interactions between microservices.

The demographics for our survey showed that most of our participants had $\leq$ 2 years of experience in microservice development, and half of our participants had $\leq$ 3 years of experience in software development. Hence, our results are biased towards the experience of junior software developers over senior developers. However, as a recently defined software architecture, this level of experience may be common for microservice projects. Additionally, our participants work on microservice projects that primarily use a publish/subscribe messaging paradigm. Developers working on other types of microservice architecture may have a different experience with these problems.

There may be frequent and/or impactful problems that were not included in our problem list. To mitigate this we performed a pilot study with an experienced team lead at Swordfish Computing, but problems may still have been missed. To address the issue, we gave participants the option to manually enter their own problems. The lack of any duplicate problems submitted by participants implies that there were no major problems that we missed, though the possibility does remain that participants were averse to the long-answer question format.

Our results for existing documentation tools may be threatened by participants' definition of 'documentation tool'. While our pilot tester did not flag the definition as ambiguous, it may have been a mistake considering how few stated they used IDE debuggers and logs. It is also possible that some of the less common documentation tools such as MS Word or git diffs are used by more participants, who simply forgot while completing the survey. However, this does not affect our data for other tools, such as Markdown and Draw.io, nor does it affect the overall number of participants who reported using textual vs graphical documentation. It is also possible that some textual documentation like Markdown could contain embedded diagrams, which could lead to an under-representation of graphical tool use. However, this would not impact our conclusion that both textual and graphical tools are widely used by participants.

Our thematic analysis of participants' ideal documentation tools are limited by the low number of participants who chose to answer the question, and by the short nature of many responses. Our inability to ask follow-up questions means our thematic analysis may not reflect the full thoughts and beliefs of the participants. To mitigate this we chose simple themes that would be difficult to misinterpret.

VIII. Conclusion

This study has surveyed a small group of microservice developers at a single company to identify the impact and frequency of problems they face during their development work, as well as the documentation tools they currently use and would like to use in the future. The findings suggest that for the survey participants, the most impactful and frequent problems faced relate to: the data structures used by microservices and channels; the communications between microservices and channels; and the causes and effects of service faults. Additionally, our findings on documentation tools suggest that textual and graphical documentation tools are both widely used by participants, with markdown and draw.io being the most commonly used tools. Our findings on ideal documentation tools showed several common themes: participants want to minimise the effort to document their code; they want the documentation to be tightly coupled with code; and they are interested in visualisation and interactivity.

The results of this study can be used to inform the design of new microservice visualisations that better focus on the problems faced by developers. More generally, the results can be used to guide future research that seeks to develop solutions to microservice development challenges. Based on the results, network and trace visualisations would address the most impactful and frequent issues faced by microservice developers, but our results are applicable to any type of visualisation. Visualisations would also be made more valuable if data structure information could be integrated into the visual data. Additionally, we find that a developer's ideal visualisation would be interactive and automatically generated from the source code of the microservice.

References

[1] L. Baresi and M. Garriga, "Microservices: The Evolution and Extinction of Web Services?" A. Bucchiarone, N. Dragoni, S. Dustdar, P. Lago, M. Mazzara, V. Rivera, and A. Sadovykh, Eds. Cham: Springer International Publishing, 2020, pp. 3–28. [Online]. Available: https://doi.org/10.1007/978-3-030-31646-4_1

[2] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, Today, and Tomorrow," M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_12

[3] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, "Benchmark Requirements for Microservices Architecture Research," in *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*, May 2017, pp. 8–13. [Online]. Available: https://doi.org/10.1109/ECASE.2017.4

[4] S. Silva, J. Correia, A. Bento, F. Araujo, and R. Barbosa, "μ' Viz: Visualization of Microservices," in *2021 25th International Conference Information Visualisation (IV)*, Jul. 2021, pp. 120–128, iSSN: 2375-0138. [Online]. Available: https://doi.org/10.1109/IV53921.2021.00028

[5] T. Munzner, *Visualization Analysis and Design*, 1st ed. Boca Raton: A K Peters/CRC Press, Dec. 2014.

[6] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, and D. Taibi, "Microservice Architecture Reconstruction and Visualization Techniques: A Review," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Aug. 2022, pp. 39–48, iSSN: 2642-6587. [Online]. Available: https://doi.org/10.1109/SOSE55356.2022.00011

[7] I. K. Aksakalli, T. Çelik, A. B. Can, and B. Tekinerdoğan, "Deployment and communication patterns in microservice architectures: A systematic literature review," *Journal of Systems and Software*, vol. 180, no. 111014, Oct. 2021. [Online]. Available: https://doi.org/10.1016/j.jss.2021.111014

[8] M. Viggiato, R. Terra, H. Rocha, M. T. Valente, and E. Figueiredo, "Microservices in Practice: A Survey Study," Tech. Rep., Aug. 2018, arXiv:1808.04836 [cs] type: article. [Online]. Available: https://doi.org/10.48550/arXiv.1808.04836

[9] J. Ghofrani and D. Lübke, "Challenges of Microservices Architecture: A Survey on the State of the Practice," *ZEUS*, pp. 1–8, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4803408

[10] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting Microservices: Practical Opportunities and Challenges," *Journal of Computer Information Systems*, vol. 60, no. 5, pp. 428–436, 2020. [Online]. Available: https://doi.org/10.1080/08874417.2018.1520056

[11] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, Feb. 2021. [Online]. Available: https://doi.org/10.1109/TSE.2018.2887384

[12] B. Mayer and R. Weinreich, "A Dashboard for Microservice Monitoring and Management," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Apr. 2017, pp. 66–69. [Online]. Available: https://doi.org/10.1109/ICSAW.2017.44

[13] T. Engel, M. Langermeier, B. Bauer, and A. Hofmann, "Evaluation of Microservice Architectures: A Metric and Tool-Based Approach," in *Information Systems in the Big Data Era*, ser. Lecture Notes in Business Information Processing, J. Mendling and H. Mouratidis, Eds. Cham: Springer International Publishing, 2018, pp. 74–89. [Online]. Available: https://doi.org/10.1007/978-3-319-92901-9_8

[14] M. Frisell, "Information visualization of microservice architecture relations and system monitoring : A case study on the microservices of a digital rights management company - an observability perspective," Master's thesis, Royal Institute of Technology, Stockholm, Sweden, 2018. [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2:1240044&dswid=-8254

[15] A. G. Fink, *How To Conduct Surveys A Step-By-Step Guide*, 6th ed. Sage Publishing, 2017.

[16] V. Braun and V. Clarke, "Thematic analysis." *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological.*, p. 57, 2012. [Online]. Available: https://doi.org/10.1037/13620-004

[17] E. Holscher, "Docs as Code," 2023. [Online]. Available: https://www.writethedocs.org/guide/docs-as-code/

[18] B. Andel, "Continuous Documentation: Automating Document Preparation with your DevSecOps Pipeline," in *2022 IEEE 29th Annual Software Technology Conference (STC)*, Oct. 2022, pp. 156–165. [Online]. Available: https://doi.org/10.1109/STC55697.2022.00029

[19] D. Holten and J. J. van Wijk, "A user study on visualizing directed edges in graphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09. New York, NY, USA: Association for Computing Machinery, Apr. 2009, pp. 2299–2308. [Online]. Available: https://doi.org/10.1145/1518701.1519054