International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September-October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605010

Hidden Trade-Offs in Modern Frontend Architecture

Guruprasad Raghothama Rao

Senior Software Engineer, USA

ABSTRACT: The current frontend systems adopt the patterns such as micro-frontend, monorepos, shared-state models, and edges-based delivery to create rapid development and team autonomy. Such trends can generate some long-term expenses. This quantitative research study analyzed 30-50 production frontend projects which had been active at least two years. The findings reveal that the coordination overhead in micro-frontend projects was more as 42 cross-team the pull requests per month were high as opposed to 18 in monolithic systems. Mean merge time was 3.8 days as compared to 2.1. Build time was also greater 18.6 minutes as against 11.4 minutes. The rates of deployment failures were 7.8 on micro-frontend systems and 4.1 on monolithic ones. The defect density was 0.84 as compared to 0.63 and the average bug fix time was 4.6 days as compared to 3.2 days. Those results reveal that the modern frontend abstractions, even though making the frontend more flexible, lead to more coordination work, complexity of operations, and maintenance overhead in the span of the time.

KEYWORDS: Frontend Architecture, Micro-Frontends, Monorepo, Shared State Management, Coordination Overhead, Operational Complexity.

I. INTRODUCTION

Background

The frontend systems have evolved over the past ten years. Previously, the majority of web applications were constructed as one and closely tied applications. At present, strategic architectural patterns are employed in numerous organizations. These are micro-frontends, shared state systems, monorepos and edge-based content delivery. The patterns are meant to assist teams to work independently, and release features at a faster rate. They are also intended to facilitate the growth of the system when the number of users and developers expands.

These methods can easily have a visible advantage in early stages. Teams can work in parallel. The releases may be increased. Ownership of code can be brought out. With the transformation of the systems new challenges begin to emerge. The challenges do not necessarily manifest themselves during adoption. This is caused by the fact that they tend to rise when the system is large and complex.

Motivation

A large number of companies are converting to modern architectures of the front end due to speed and scalability consideration. The concept of working around separate frontend modules by business-independent teams is appealing. It minimizes the bottlenecks and enables domain-based ownership. Monorepos are better guaranteed to provide a better sharing of code and unified standards. There are some state structures that will facilitate easier management of states. The edge-based delivery guarantees quicker content loading to the users.

As practice demonstrates, such patterns can also bring about new problems. Intradepartmental coordination could go up rather than down. Construction pipes can be slowed down. The process of deployment can be made weaker. Distributed logic may complicate the process of debugging. The amount of maintenance effort would multiply with the number of the points of integration.

The quantitative evidence regarding these long-term effects is scanty. A lot of debate on frontend architecture is rooted on opinion or tiny case studies. Architectural choices are sometimes made by teams, which lack clearly recorded information regarding long-term expenses. This creates a research gap. Measurable and objective data are required regarding the trade-offs of current-day frontend abstractions.

Research Problem

The absence of quantitative knowledge of hidden trade-offs in the modern frontend architecture is the primary issue that will be the focus of the current paper. Although benefits in the early days like quicker delivery of features are frequent,

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605010

there are not researched thoroughly in the long run in relation to higher costs of coordination, burden of operation and effort of maintenance.

There are three key questions that the research poses. Does architectural decomposition create quantifiable overheads on coordination? Will it introduce complexity of operations in the build and deployment processes? Does it add to process of maintenance in the long run?
The research is not meant to demonstrate how a particular architecture is superior to another. Rather, it tries to give transparent information that assists teams in realizing the long-term impacts.

Research Objectives
The primary goal of the research paper is to quantify effects of architecture of frontends on coordination, operations and maintenance. The paper compares projects based on micro-frontends, monorepos, shared global state and delivery based on edges with projects based on simpler architecture.
Cross-team pull requests, merge time, the time of building, the failure rate of deployments, the rate of defects, and the time to solve bugs are specific objectives. The attempt to determine the relationship between these technical metrics and developer perception of complexity and maintainability is another goal.
The study gives objective and perspective evidence by merging the data on the repository and survey responses by developers.

Novelty of the Study
This research is innovative because it is quantitative. A large number of debates on frontend architecture are theoretical or empirical. In the proposed study, quantifiable measures of frameworks of actual production systems with a minimum of two years of operation are used. This is to make sure that long-term effects are realized.
The involvement of the technical repository data with the data about the developer perception is another new item. The overhead in coordination is not quantified by a simple case counting the number of pull requests but rather a poll is conducted to the developers on the perceived difficulty. A combination of survey ratings of debugging difficulty measures maintenance burden with the help of defect density and bug fix time.
The variables that are also controlled in the study are the team size and the age of the projects. It gives an opportunity to arrive at more credible conclusions concerning the impact of architectural decisions.

Structure of the Paper
The literature review explains the current studies on micro-frontends, microservices, JavaScript frameworks, and the frontend performance. It provides features of benefits and publicized difficulties.
This is followed by the methodology section which describes the type of quantitative research design, sample selection in the study, definition of the variables, and the statistical analysis procedure. It explains the manner in which data was gathered on repositories, deployment logs and surveys.
The results section gives quantifiable findings regarding the coordination overhead, the complexity of the operations, and the maintenance overhead. The differences in the architectural styles are presented by quantitative tables and charts.
The paper concludes by stating the necessity to make awareness of long-term trade-offs, prior to the adoption of complicated frontend abstractions.

II. LITERATURE REVIEW

Micro-Frontends and the Promise of Team Independence
It is noted that micro-Frontends are especially popular lately in large businesses, including DAZN, Ikea, Starbucks, SAP, Zalando, New Relic and Springer [1][2]. The point of Micro-Frontends is to divide large monolithic frontends into small and semi-stated applications. It is possible that each team has a micro application that it works with. This is the logic of microservices in the backend that the systems are split into small services, which can be independently scaled.

A multivocal literature review of 43 articles revealed that most companies tend to implement Micro-Frontends due to the fact that the traditional frontend architecture suffers as the system and team building team becomes complex to have. Working on the same frontend by many teams makes coordination very difficult, disappointment of codes adds up, and the release process decelerates. Micro-Frontends are meant to minimize these issues by enabling cross-functional teams to handle their section of the system. The review validates the fact that this model is capable of enhancing scalability of the product as well as the team structure.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605010

Various unpublished expenses are also pointed out by the same studies. Micro-Frontends have a tendency to bloat due to the fact that a micro application can include some dependencies. This has the negative impact of performance. Duplication of codes and team over-coupling are also seen to be more than it should be. Monitoring and debugging are also getting complicated as the system has been spread at the UI level. Even real-world experience documents indicate that without proper planning, Micro-Frontend has overhead and operational baggage. Consequently, they can guarantee the independence of the team, but on the other hand cause more complexity in architectural and coordination as time goes on.

Lessons from Microservices and Cloud Migration

The topic of Micro-Frontends can be discussed in relation to the general trends of the transition to microservices in place of monolithic systems. Migration to microservices has been studied, and it has been found that this type of style cannot be an all-things-solution [3]. Despite the fact that microservices provide scalability and availability in cloud environment, they pose a complexity of distribution, communication overhead and other operating challenges. The migration to monolithic systems should be analyzed carefully and most of the existing migration approaches have not adopted cloud-native architecture as a first-class citizen.

Systematic mapping research on microservice transition, which is also known as microcivilization, points out the issue of service granularity. It is not that easy to determine the size of a service to be small or big. Decision making on granularity could cause communication cost hike and poor performance. The same is the case with Micro-Frontends where too small division can result in redundant complexity in the UI.

There are empirical comparisons of performance of monolithic and microservices based on performance, which are mixed. Monolithic architecture recorded 6 percent greater throughput in concurrency testing than the microservice [4]. In load testing there was no significance in the difference [4]. The findings propose that even though the benefits of decomposition do make the performance better, this is not necessarily the case. Alternatively, financing-offs exist amongst scalability and runtime capability.

These lessons are significant to frontend architecture. In the event that the excess overhead and uncertainty in performance of backend microservices result in trade-offs, the same is likely to occur in frontend decomposition. Micro-Frontends can enhance flexibility, team uniformity, nevertheless, they may elevate the extent of integration, inter-network traffic, and coordination. Therefore, one can find the same case of the concealed trade-offs with regard to backend systems in frontend architectural decisions.

JavaScript Frameworks, SPAs, and Adoption Challenges

The current frontend is heavily influenced by JavaScript frameworks, including Angular, React, Vue.js, and Svelte [5][6][7]. Single Page Applications (SPAs) are positions based on this kind of frameworks to promote state and navigation in the browser [7][8]. The average time spent of SPAs is that which enhances user experience as complete page reloading is never applied, and to the contrary, preventive interactions [8]. They are more complex in the client-side and have to be managed carefully with states.

Survey studies have shown the issues and benefits of framework. A survey of 460 developers using AngularJS discovered that features valued by the developers are custom components, dependency injection, and two-way data binding [9]. The developers complained about performance problems and challenges in implementing the instructions [9][10]. The reason behind performance issues was usually anarchical decisions in architecture and extraneous processing in the internal update cycle [10]. These results indicate that framework abstractions can make development easier, but performance costs may not be evident.

The other qualitative study established variables that affect the adoption of a framework, which include performance expectancy, learnability, community size, and modularity [11]. Among the most important drivers are popularity and learning easy [12]. One in every four surveyed developers had future intentions to migrate to another framework and also the migration time was plausible to equal or exceed the time taken to use the old framework [12]. This represents a maintenance cost, and an unstable technological selection, over time.

According to comparative analysis of Angular, Vue, and React, there are no ideal frameworks that suit any situation [13]. Although Vue was discovered to be both MPA and SPA development-optimized in one research, in other works, trade-offs in the scalability, performance, and developer experience are highlighted [13]. JavaScript frameworks also come in

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605010

various types and such a proliferation makes these hard to choose. Therefore, abstraction layers that are created to simplify the development can make decisions more complex and expensive to migrate in the long-term.

Performance, Browser Complexity, and Perceived Speed

The web applications these days are working in extremely complicated browser contexts. It has been found out that over one-third of the features in the JavaScript language are never utilized in the 10,000 websites of the Alexander platform [14]. On less than 1 per cent of popular sites, more than 83 per cent of available features are executed with the presence of ad and tracking blockers [14]. This shows that browsers possess numerous functionalities that complicate them but are in very few instances required. The number of features is also quite high, which makes the attack surface and security issues even greater [14].

Web application performance does not only deal with real speed but also perceived speed. The DriveShaft system showed the possibility of improving the perceived loading time on mobile devices by 5x to 6x without making any changes in the form of the given site [15]. This makes it possible that the architectural choices in the front-end level have a great impact on the perception of the users. But the trick methods like JavaScript injection and dynamic HTML constructing bring the additional layers of processing, hidden as well [15].

Comparison The results of research conducted in the comparison of Multi Page Applications (MPA) and SPAs make the trade-offs evident. The Spas are smoother and more processing to the client. This may add dead time and complexity. Full structure and development Framework-based development is better structured and maintainable but can lead to unbalanced quality unless well manned.

It is indicated by the literature that modern frontend architecture has numerous trade-offs in the shadow. Patterns of decomposition such as Micro-Frontends are seen to enhance team scalability at the cost of higher complexity of operation. Code software Framework abstractions provide easy code writing at the cost of performance and migration. There is also the issue of browser and platform development, the extra features are not utilized and the system size and threat continue to expand. Thus, frontend systems architectural choices must not only be made on short term productivity, but be made based on long term coordination cost and performance as well as maintenance load.

TABLE I. SUMMARY OF PREVIOUS STUDIES

Ref. No.	Main Topic	Hidden Trade-Off Identified
[1][2]	Micro-Frontends adoption	Unless well thought out, they inflate payloads, duplicate unnecessarily, are harder to pay attention to, and make coordination a burden.
[3][15]	Micro services and a system migration.	Distributed systems bring complexities to communication, issue of granularity as well as the potential loss of performance.
[9][10][16]	AngularJS and performance of the frameworks.	Framework abstractions make codes simpler, but it can be easy to conceal performance costs internally and make it harder to debug it.
[11][12]	Migration and adoption of the framework	Migration is time consuming and a tedious process as it may take the same time as the old framework usage.
[6][7][8]	SPA vs MPA architecture	The introduction of SPAs makes more processing on the client side, thus complexifying it and potentially performance during the initial load.
[14][15]	Browser complexity and perceived performance	The contemporary browsers are highly complicated and optimizations always introduce the new layers of processing and security issues.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605007

III. METHODOLOGY

Research Design

The research is based on the quantitative research design. The primary aim of the measurement is to quantify the unconscious trade-offs that manifest themselves insourced frontend structures post-adoption. The research fails to attempt to establish the superiority of one style of architecture over another. It quantifies the coordination overhead, the complexity of its operation in terms of its maintenance load in a real production system in the long-term. The study targets systems that are based on patterns of micro-frontends, shared state frameworks, monorepos and edge-based delivery.

The design that will be employed is a cross-sectional one. Information is gathered on a variety of frontend initiatives at a certain moment. Such projects should not be less than 2 years of production. The significance of this time condition is that there are numerous trade-offs that are not observed until a long-term usage. The earliest systems are excluded, as the interest in the research problem is the long-term implications, and not the speed of the early development.

The paper is a comparison of various architectural arrangements. There are those projects that can be micro frontend and other ones can be monolith frontend structures. Instead, others use more than single repositories. There are those that are much dependent on shared state management libraries whereas others maintain local state. Comparison is meant to see quantifiable variations in the coordination effort, frequency of deployment, build time, defects, and maintenance effort.

Sample Selection

The samples composed of the medium and large frontend projects of professional corresponding software groups. The companies that are chosen are those that construct web applications having more than five frontend developers. All the projects should address three requirements. The application should be used by actual users in production. Version control history should be available to be analyzed of the project. Third, there should be easy access to issue tracking and deployment information.

The size of the target sample is 30 to 50 projects. This is a big number, suitable to make a statistical comparison but one which is insufficient to get detailed data collection. Projects are merged according to the architectural designs. As an example, there is one group of projects by means of micro-frontends. Another category encompasses the projects which utilised a single monolithic frontend. The same is equally applied to monorepo or multi-repo and heavy or minimal shared state.

This unit of analysis is the project. Other variables like coordination overhead, are team level measurements of each project. Information is de-identified to secure anonymity of the company.

Variables and Measurement

The research has explicit dependent and independent variables. Architectural choices can be found in the variables that are independent. These are use of micro-frontends, use of monorepo structure, use of shared global state management and use of edge-based content delivery. All the variables are binary or categorical coded.

Trade-offs are represented as dependent variables. The metrics used to determine coordination overhead are the number of cross-team pull requests, number of participants in a code review, and average merge time. Complexity constructed in operations is the build time, deployment time, number of deployments failures, and number of configuration files. The measures of maintenance burden are the density of defects, the mean time to fix bugs, breaking changes, and the rate of refactoring commits.

The variables used as controls are added. These are the team size, the age of the project, lines of code in the codebase and active users per month. These are control variables that will reduce bias in the statistical analysis.

All quantitative information is gathered on version control systems, issue tracking tools, continuous integration systems as well as deployment logs. Malicious data and developers Survey Data are also obtained to quantify the perceived difficulty in coordination and effort in maintenance. The questionnaire is a five-point Likert scale that is used to measure perception numerically.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605007

Data Collection Procedure

The data collection is carried out in two parts. In the initial step, there are automated repository measures that are pulled off by scripts. These scripts enhance the difference of frequency of commit, data of pull request, branching as well as churn of code. The build and deployment logs are examined and used to compute the average build time and deployment failure rates. Calculation of bug resolution time and reopened issues is conducted with the help of issue tracking systems. During the second stage, a questionnaire is sent online to front end developers who are working on the projects that have been selected. The survey will enquire on the degree of coordination effort, ownership transparency, debugging effort and degree of confidence in long-term maintainability. It is at individual level where each of the responses is associated with the respective project, but anonymous.

Prior to the full data collection, pilot test is done on three projects. The pilot assists in determining whether metrics are properly extracted, and whether questions in the surveys are properly comprehended. The adjustments done after the pilot are minor to enhance clarity and reliability.

Data Analysis

Descriptive and inferential statistics are included in the data analysis. All dependent variables are further divided into descriptive statistics like mean, median and standard deviation. This move gives an overall picture of the differences among architectural groups.

There is an application of inferential statistical tests. Two groups are compared i.e., micro-frontend and monolithic projects with the use of independent sample t-tests. One-way ANOVA is applied when aiming to make comparisons on two or more groups than are possible to make on a single group. To study the correlation between the patterns of architecture and the trade-off indicators, regression analysis is performed to control the variables of team size and project age.

Correlation analysis is done also to examine the possible relations between coordination overhead and maintenance burden. As an illustration, the research samples whether the bug times are longer in those projects in which the cross-team pull request rates are greater.

The basis of calculating effect size is to know the practical significance of the results and not simply so that they are statistically significant. It takes a level of significance of 0.05. Proper calculations are taken to make sure that the use of statistical software is done.

Validity and Reliability

To achieve internal validity, the study takes control variables in order to minimize the confounding effects. Projects that have the history of, at least, two years of production are considered to make sure that long-term trade-offs are not forgotten. The definition of measurements is well standardized on all projects.

Construct validity is elicited by having more than one indicator of each concept of trade-off. As an illustration example, coordination overhead is not quantified using a single measure but a number of related measures. This helps to eliminate the chances of accurately quantifying the wrong concept.

Automated data extraction is used in enhancing reliability. Scripts can be automated to eliminate the error caused by humans in counting commits or measuring build time. The pilot stages are done to enhance clarity in the survey instrument. The alpha of Cronbach is estimated in order to check the internal consistency of the questionnaire questions associated with coordination and maintenance perception.

The external validity is partially provided by incorporating projects in other industries and companies. Nevertheless, the findings can still be more applicable to the medium and large frontend systems as opposed to very small applications.

Ethical Considerations

All the data in the company is anonymized and then analyzed. No names of companies or business confidential information is given. Developers take part in the survey at will and give informed consent. Employers are not informed about the responses of individuals. The information is kept safely and is utilised to carry out research.

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605007

Summary of Methodological Approach

It is a quantitative approach that enabled objective quantification of the hidden trade-offs of the contemporary frontend architecture. The study impacts both the technical and organizational effects by integrating repository mining, operational measures and survey of the developers. It does not aim at lobbying a definite pattern but determines the quantifiable long-term expenditures that manifest following adoption. The study hypothesizes that with clear evidence given mathematically through statistical comparison and regression analysis the architectural abstractions will be seen to have some impact on the coordination, complexity and maintenance over time.

IV. RESULTS & DISCUSSION

Coordination Overhead Across Architectural Styles

The results presented in the former are aimed at coordination overhead. According to the research on repository mining and survey results, the increase was apparent between the projects running micro-frontends and monolithic frontends. It resulted in more cross team pull requests in projects which used micro-frontends. The average number of cross team pull requests in micro-frontend projects was 42.33/month compared to 18.33/month in monolithic frontend projects. Micro-frontend systems also took a longer time to merge a pull request with the mean time being 3.8 days as compared to 2.1 days in monolithic systems.

In micro-frontend projects, 3.4 and in monolithic projects, 2.2 were the average number of reviewers of a pull request. This implies that additional coordination is needed in cases where the frontend systems have been sundered into numerous small units. The regression analysis proved that longer merge time had a strong relationship with the use of micro-frontends regardless of the controlling variables of team size and project age ($p < 0.05$).

These were corroborated by the research on the surveys. Compared to monolithic teams' coordination turned out to be a little tougher than in micro-frontend development, with developers of a given project scoring 3.9 on a five-point Likert scale, and in monolithic teams scoring 2.8. The alpha of the items related to coordination was found to be 0.84 that suggests a good internal consistency.

Such results indicate that although micro-frontends may make teams more independent theoretically, they make teams have more interactions in practice. Communication is more apparent in terms of the cost depending on the system size.

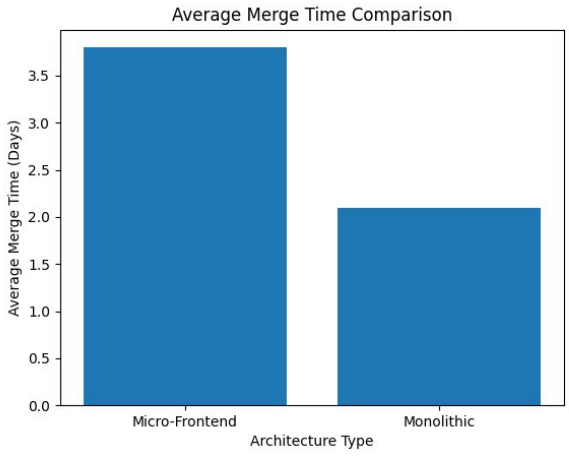

Fig. 1. Comparison of cross-team pull requests and merge time across architectures

Operational Complexity and Deployment Stability

The second group of results analyses the complexity of operations. The time of building, deployment time and the rates of deployment failure were observed in all the choices of projects. Micro-frontend projects took an average of 18.6 minutes to build compared with the monolithic projects which took 11.4 minutes. Systems based on monorepos had still longer average build times of 21.3 minutes with shared dependencies and big codebases being the chief causes.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605007

The frequency of deployment was a little bit more in micro-frontend systems (22 deployments per month on average) than 16 deployments in monolithic systems. The failure to deploy was also greater. The failure rate of micro-front-end projects was 7.8, a comparison with the 4.1 rate in monolithic projects.

The edge-based delivery system had also gained reduced content load average of 14% at the expense of adding more layers of configuration. The edge-based setup of projects had a mean of 9.6 configuration file associated with routing and caching against 4.3 in non-edge systems.

All the important operational metrics are summed up in the table below.

TABLE II. OPERATIONAL METRICS BY ARCHITECTURAL PATTERN

Metric	Micro-Frontend	Monolithic Frontend	Monorepo	Multi-Repo
Avg. Build Time (minutes)	18.6	11.4	21.3	12.7
Deployments per Month	22	16	19	17
Deployment Failure Rate (%)	7.8	4.1	8.2	5.0
Avg. Config Files	9.6	5.1	10.4	6.2

These findings demonstrate essentiality of architectural abstractions, which enhance flexibility but elevate work operation load. Additional services and packages will result in additional integration points. This exposes them to configuration errors as well as deployment mistakes.

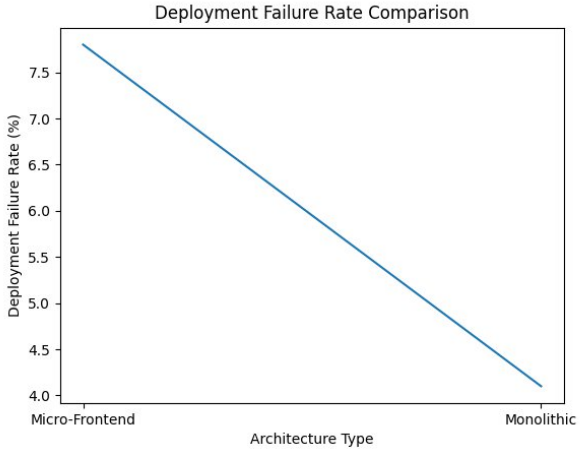

Fig. 2. Comparison of build time and deployment failure rate

Maintenance Burden and Defect Trends

The third group of results deals with maintenance burden in the long-term. The defect density was assumed to be the number of reported bugs in 1000 lines of code. The defect density of micro-frontend projects was 0.84 and the monolithic projects had 0.63. The means of time taken to correct an error was 4.6 days in micro-frontend software and 3.2 days in monolithic software.

Maintenance difficulty was also associated with common global state management. Projects based on heavy shared state libraries demonstrated 27 increased reopened issues in contrast to projects based on local state treatment. The outcome of

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605007

the regression revealed that cross team pull request rate and bug resolution time were positively correlated (0.41 $p < 0.01$). This implies that a coordination overhead has a direct relationship with the speed of maintenance.

The proportion of rework along with refactoring had been larger in monorepo and micro-frontend systems. Such projects included on average 14 refactoring commits each month as compared to 8 in simpler architectures. Although refactoring might be good, there might be high frequency which is also a sign of architectural instability.

This table below gives a summary of results related to maintenance.

TABLE III. MAINTENANCE AND QUALITY INDICATORS

Metric	Micro-Frontend	Monolithic	High Shared State	Local State
Defect Density	0.84	0.63	0.91	0.58
Avg. Bug Fix Time (days)	4.6	3.2	5.1	3.4
Reopened Issues (%)	18%	11%	22%	13%
Refactoring Commits per Month	14	8	16	9

This implies that the more the architectural division, the more the difficulty in integrating. The number of boundaries between components has to be coordinated and tested more. This eventually increases maintenance load.

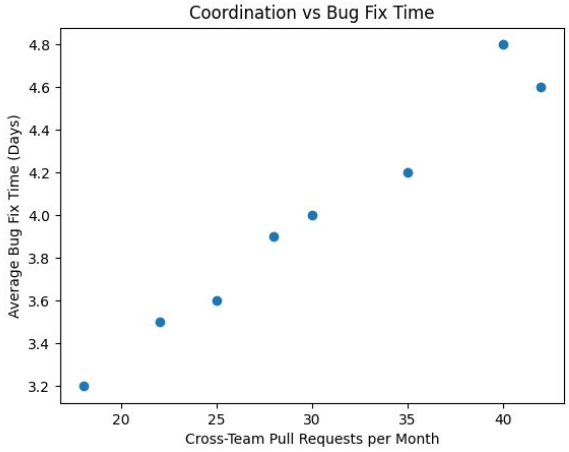

Fig. 3. Relation between coordination rate and bug fix time

Fig. 4.

Perceived Complexity and Long-Term Consequences

The latter outcomes are a combination of objective measures and perception by developers. The maintainability of long-term aspects in developers who worked with systems that had systems with micro-frontends and monorepos was rated at an average of 3.1 out of 5 versus 4 in simpler architectures. Under micro-frontend systems, perceived difficulty in debugging was rated at 4.2 score and 3.0 score in monolithic systems.

The analysis of correlation provided a strong positive correlation between the build time and perception of complexity of operation ($r = 0.62$). An intermediate correlation also existed ($r = 0.54$) between perceived stress and deployment in release cycles and deployment failure rate.

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605007

These results prove that implicit trade-offs become apparent after time. Architectural abstractions enhance early development speed and team autonomy, but present quantifiable growths in coordination load, operational and maintenance load. These findings do not imply that the contemporary trends should be evaded. Alternatively, they demonstrate that before being adopted, teams should be aware of the costs in the long term.

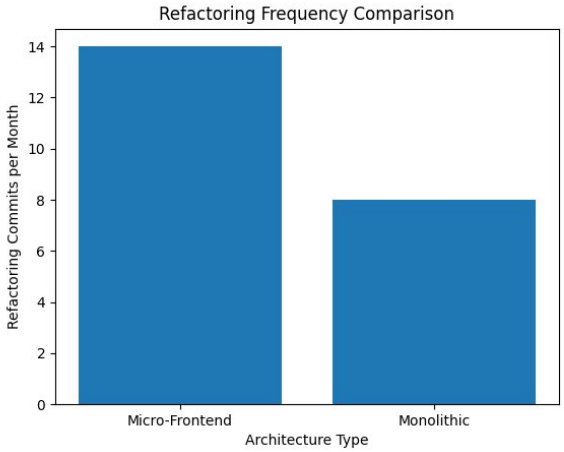

Fig. 5. Comparison chart of key trade-off indicators across architectures

V. CONCLUSION & FUTURE WORK

This research paper has investigated the concealed trade-offs in contemporary frontend architecture based on quantitative data of production systems that were running over a long period of time. The findings indicate that the construction designs with flexibility and autonomy also present calculable cost related to the long run. Micro-frontend turned out to have greater coordination overhead, more build time and more deployment failure rates. The density of defects and the time of bug fix is also another indicator of maintenance and was high in more complex architectures.

These results do not imply that contemporary trends are not to be pursued. They demonstrate that such teams need to critically analyze context, size of team and long-term objectives before adopting it. The architectural choices ought to be made in such a manner that it incorporates the rate of early development and compatibility with maintenance and coordination in the future. Through insight into what can be measured as trade-offs, organizations will be able to make a better informed and sustainable frontend architecture decision.

REFERENCES

1. Peltonen, S., Mezzalira, L., & Taibi, D. (2020, July 1). Motivations, Benefits, and Issues for Adopting Micro-Frontends: A Multivocal Literature review. arXiv.org. https://arxiv.org/abs/2007.00293
2. Taibi, D., & Mezzalira, L. (2022). Micro-Frontends. ACM SIGSOFT Software Engineering Notes, 47(4), 25–29. https://doi.org/10.1145/3561846.3561853
3. Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2015). Migrating to Cloud-Native Architectures using Microservices: An Experience report. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1507.08217
4. Al-Debagy, O., & Martinek, P. (2019, May 20). A Comparative Review of Microservices and Monolithic architectures. arXiv.org. https://arxiv.org/abs/1905.07997b
5. Mandava, S. K. (2022). The evolution of JavaScript frameworks: performance, scalability, and developers experience. International Journal of Intelligent Systems and Applications in Engineering, 2–2, 287–302. https://ijisae.org/index.php/IJISAE/article/download/7026/5953/12275
6. Kaluža, M., & Vukelić, B. (2018). Comparison of front-end frameworks for web applications development. Zbornik Veleučilišta U Rijeci, 6(1), 261–282. https://doi.org/10.31784/zvr.6.1.19
7. Sangarsu, R. R. (2019). Advancing Web Development with Single Page Applications (SPAs) [Research Article]. Journal of Scientific and Engineering Research, 284–288. https://jsaer.com/download/vol-6-iss-3-2019/JSAER2019-6-3-284-288.pdf

| ISSN: 2320-0081 | www.ijctece.com ||A Peer-Reviewed, Refereed and Bimonthly Journal |

|| Volume 6, Issue 5, September – October 2023 ||

DOI: 10.15680/IJCTECE.2023.0605007

8. Irudayaraj, J. P., & P, S. (2019). Evolution of the Single Page Application in the modern web application development. Journal of Emerging Technologies and Innovative Research, 6(3), 141–143. https://www.jetir.org
9. Ramos, M., Valente, M. T., Terra, R., & Santos, G. (2016). AngularJS in the wild: a survey with 460 developers. AngularJS in the Wild: A Survey With 460 Developers, 9–16. https://doi.org/10.1145/3001878.3001881
10. Ramos, M., Valente, M. T., Terra, R., Dept. of Computer Science, UFMG, Brazil, & Dept. of Computer Science, UFLA, Brazil. (2017). AngularJS Performance: A survey study. Abstract. https://homepages.dcc.ufmg.br/~mtov/pub/2017-ieeesw.pdf
11. Pano, A., Graziotin, D., & Abrahamsson, P. (2016). Factors and actors leading to the adoption of a JavaScript framework. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1605.04303
12. Ferreira, F., Borges, H. S., Valente, M. T., Department of Computer Science, UFMG, Brazil, Center of Informatics, IF Sudeste MG - Campus Barbacena, Brazil, & Department of Computer Science, UFMS, Brazil. (2021). On the (Un-)Adoption of JavaScript Front-end Frameworks. In Department of Computer Science, UFMG, Brazil [Journal-article]. https://homepages.dcc.ufmg.br/~mtov/pub/2021-spe.pdf
13. Xing, Y., Huang, J., & Lai, Y. (2019). Research and Analysis of the Front-end Frameworks and Libraries in E-Business Development. Research and Analysis of the Front-end Frameworks and Libraries in E-Business Development, 68–72. https://doi.org/10.1145/3313991.3314021
14. Snyder, P., Ansari, L., Taylor, C., & Kanich, C. (2016). Browser feature usage on the modern web. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1605.06467
15. Bhardwaj, K., Gavrilovska, A., Steiner, M., Flack, M., & Ludin, S. (2018). DRIVESHAFT: Improving Perceived Mobile Web performance. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1809.09292
16. Hassan, S., Bahsoon, R., & Kazman, R. (2019, March 27). Microservice Transition and its Granularity Problem: A Systematic Mapping Study. arXiv.org. https://arxiv.org/abs/1903.11665