Report: Motivations, benefits, and issues for adopting Micro-Frontends

Summary:

This study is a survey that aims to explore the motivations behind the adoption of Micro-Frontends by diverse companies. It investigates the benefits they offer and the challenges they pose, based on numerous sources. The study refrains from prescribing solutions or implementation strategies, it offers a basic explanation of Micro-frontends architecture and briefly compares it with Microservice. Additionally, it provides a short overview and comparison of alternative options for developing front-end applications. In conclusion, the article serves as a great starting point for companies considering the adoption of Micro-Frontends, presenting valuable insights for decision-making. Moreover, it raises implications and issues for further research in this area.

Keywords:

MF motivations · MF benefits · MF issues · MF compositions · FE architectures · micro-services

Research Questions:

1. Why are practitioners adopting Micro-Frontends?
2. What benefits are achieved by using Micro-Frontends?
3. Do Micro-Frontends introduce any issues?

Key Points:

➔ **Traditional frontend architectures often result in the development of monolithic projects, characterised by several drawbacks:**
- Increased complexity, interdependent changes
- Extensive codebase, numerous dependencies
- Tight coupling of components
- Impaired team cooperation, slower development *[3-4]*

➔ **The core concept of Micro-Frontends involves breaking down entire frontend applications into a mix of sub-domains**
- Each team can concentrate solely on a specific domain

- Sub-domain applications operate, develop, and deploy independently as isolated and loosely connected services
- This extends the principles of Microservices from the backend to the frontend [4]

➔ **Micro-frontends can be composed through various methods:**

- Client-side composition
  - Application shell dynamically loads Micro-Frontends, appending them as needed.
  - Utilises iframes and transclusion mechanism via client-side include.
  - Lazy loading components using a placeholder tag, replaced with corresponding components.
- Edge-side composition
  - Web page assembly occurs at the CDN level.
  - Leveraging Edge Side Include (ESI), an XML-based markup language.
- Server-side composition
  - The server composes the view by collecting various Micro-Frontends and assembling the final page. [4-6]

➔ **Primary motivations for adopting Micro-frontends include:**

- Frontend Growth
  - Large Codebase - difficulty in understanding the entire application, hindering scalability with numerous dependencies.
  - Increased complexity - maintenance challenges due to tightly interdependent functionalities.
  - Organisational problems - micro-frontends facilitate cross-functional teams, supporting collaboration among different technology stacks (Angular/React team, Java team, DB team, etc.).
- Scalability
  - Scale of development teams - Micro-frontends align better with diverse business needs.
  - Independent deployments - Enables updates to specific parts of the application.
- Code-base rules
  - More flexibility - allows experimentation with new code-base rules.
- Slow on-boarding

    - Large and Complex Application – leads to prolonged on-boarding for new developers.
- Killing innovation
    - Experimentation with New Technologies – Micro-frontends enable the evolution of specific application parts without affecting the entire system.
- Avoid Hasty Abstractions
    - Monolithic Abstractions – more abstraction layers in monolithic applications result in complex and messy systems. *[8-12]*

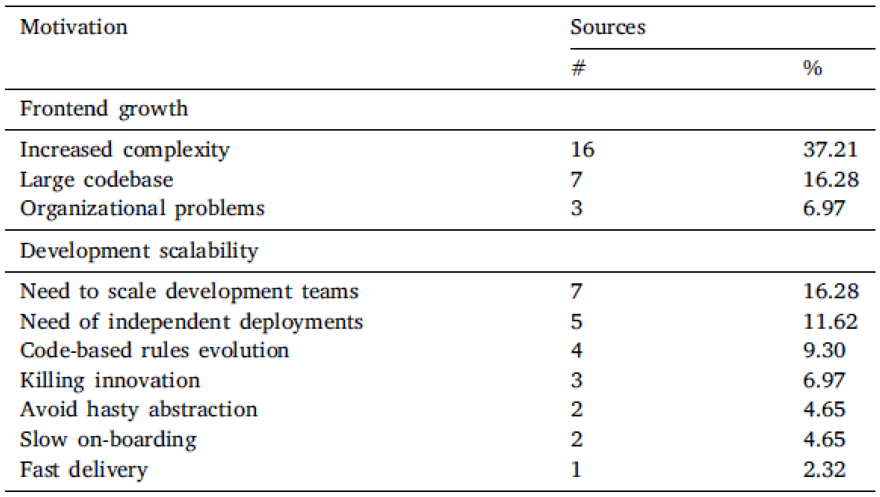
➔ **Adopting Micro-frontends offers a range of benefits:**

- Support for multiple technologies
    - Enables diverse technology stacks within each team.
- Autonomous cross-functional teams
    - Teams can work on different parts of the application independently, without impacting others.
- Independent development, deployment, managing and running
    - Each micro-frontend operates as an independent unit, leading to faster deployment, testing, and supporting CI/CD.
- Better testability
    - Testing specific micro-frontends eliminates the need to run the entire test suite.
- Improved fault isolation, resiliation
    - Failures in one part don't necessitate shutting down the entire application; only the affected part needs attention.
- Highly scalable
    - No coupling between frontends, allowing infinite scalability without increasing complexity.
- Faster on-boarding
    - New developers comprehend the system more rapidly.
- Fast initial load
    - The application shell loads micro-frontends based on user routes, enhancing initial load speed.
- Improved performance
    - Slow features don't impact the entire app; users can interact with faster-loaded features while the entire application loads.
- Future proof
    - Easily integrates new frameworks, avoiding the need to stick to a single framework. *[12-14]*

➔ **Challenges associated with Micro-frontends**

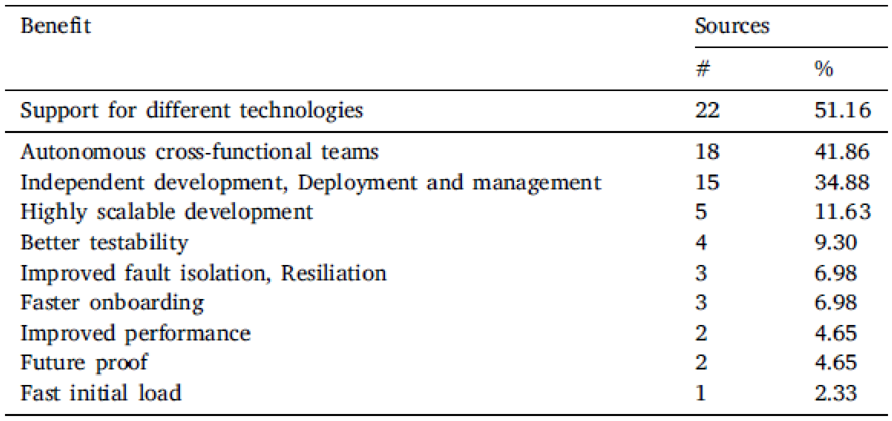
- Increased payload size
    - Slower loading times due to the browser fetching a substantial amount of data, if multiple JS frameworks are used.
- Code duplication
    - Bundlers from independent builds may lead to duplication of dependencies, increasing download size.
- Shared dependencies
    - Complex management due to redundancy of dependencies across sub-projects.
- UX consistency
    - Difficulty in maintaining consistent user experience across sub-projects.
- Monitoring
    - Challenges in tracking and debugging across the entire system.
- Increased complexity
    - Elevated technical and organisational complexity, supporting different sub-projects with varied technologies.
- Administration
    - Collaboration difficulties, with cross-functional teams working on the same product but different code-bases.
- Repetition
    - Repetitive implementation, with the same functionality written multiple times
- Environment differences
    - Risks associated with developing in an environment significantly different from production.
    - Can be avoided by testing the application in production with none or small live traffic
- Higher Risk in Releasing Updates
    - Potential bugs and errors emerging at application run-time during updates.
- Accessibility challenges
    - Certain implementations, such as iFrames, can cause huge accessibility challenges. *[14-15]*

Infographics:



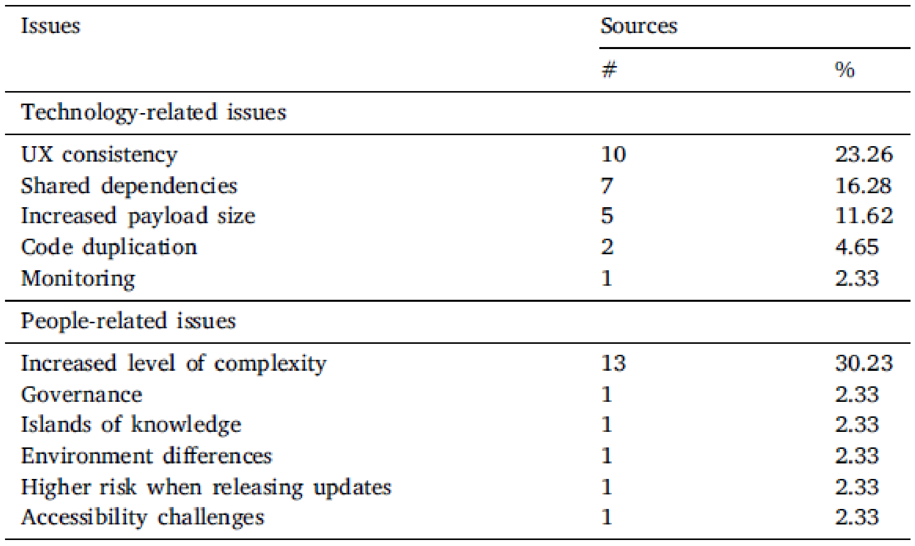
Motivation	Sources	
	#	%
Frontend growth		
Increased complexity	16	37.21
Large codebase	7	16.28
Organizational problems	3	6.97
Development scalability		
Need to scale development teams	7	16.28
Need of independent deployments	5	11.62
Code-based rules evolution	4	9.30
Killing innovation	3	6.97
Avoid hasty abstraction	2	4.65
Slow on-boarding	2	4.65
Fast delivery	1	2.32

Fig. 1: Motivations for the adoption of Micro-Frontends [11]



Benefit	Sources	
	#	%
Support for different technologies	22	51.16
Autonomous cross-functional teams	18	41.86
Independent development, Deployment and management	15	34.88
Highly scalable development	5	11.63
Better testability	4	9.30
Improved fault isolation, Resiliation	3	6.98
Faster onboarding	3	6.98
Improved performance	2	4.65
Future proof	2	4.65
Fast initial load	1	2.33

Fig. 2: Micro-Frontends benefits [12]



Issues	Sources	
	#	%
Technology-related issues		
UX consistency	10	23.26
Shared dependencies	7	16.28
Increased payload size	5	11.62
Code duplication	2	4.65
Monitoring	1	2.33
People-related issues		
Increased level of complexity	13	30.23
Governance	1	2.33
Islands of knowledge	1	2.33
Environment differences	1	2.33
Higher risk when releasing updates	1	2.33
Accessibility challenges	1	2.33

Fig. 3: Figure caption [14]