Bus Factor

It’s the number of people on a project who would need to be hit by a bus (or otherwise unavailable) before the project stalls or becomes unmaintainable. In other words, it’s about how much knowledge concentration there is in a team or project.-

Low bus factor (e.g., 1–2) means that only one or two developers understand critical parts of the system. If they leave, get sick, or are unavailable, the project is in serious trouble. Example: A legacy system only one senior developer knows how to deploy.

High bus factor (e.g., 5–6): knowledge is well-distributed across the team.

Documentation, code reviews, pair programming, and knowledge-sharing practices are in place.

The project can continue smoothly even if some team members leave.

Low bus factor is a red flag for project continuity.

Ways to Improve Bus Factor:

1. Code Reviews & Pair Programming – Spread knowledge across multiple developers.

2. Good Documentation – Architecture, design decisions, deployment steps, etc.

3. Cross-Training – Rotate developers through different modules.

4. Shared Ownership – Avoid “my code” mentality.

5. Onboarding Practices – Ensure new developers can ramp up quickly.

Example: If Alice is the only person who knows how the payment system works → Bus factor = 1.

If 4 developers can fix and deploy it → Bus factor = 4.

Below is a real-world case study that illustrate the bus factor problem clearly:

Case Study 1: Open-Source Software Library (Heartbleed Bug, 2014): OpenSSL was (and still is) a critical open-source library used to secure most of the world’s internet traffic.

Bus Factor: Shockingly low — only two core maintainers were working full-time on it, with little funding and limited external review.

Impact: A critical bug called **Heartbleed** went unnoticed for over 2 years. When it was discovered in 2014, it affected millions of servers worldwide. Highlighted the dangers of depending on a project with a bus factor of 2 despite its global importance.

(The Heartbleed bug was a severe OpenSSL vulnerability, publicly disclosed in April 2014, that allowed attackers to read a system's sensitive memory contents, potentially exposing private keys, passwords, and other confidential data without detection. The flaw was in a misimplementation of the transport layer security within OpenSSL versions 1.0.1 through 1.0.1f, allowing a buffer over-read of up to 64KB of memory with each request. Correcting the issue involved patching the affected OpenSSL versions (to 1.0.1g or later), revoking and reissuing compromised SSL/TLS certificates, and users changing their passwords.)

⍰ Case Study 2: NASA’s Space Shuttle Software (1980s–2000s)

* **Background:** The Space Shuttle’s flight control software was one of the most complex systems ever built.

* **Bus Factor Concern:** Initially, knowledge of some subsystems was **concentrated in small groups of specialists**. If they left, critical expertise would vanish.

* **Mitigation:**

  * NASA enforced **extreme documentation standards**, **redundant code reviews**, and **team rotations**.

  * This dramatically raised the bus factor, making the system maintainable across decades and multiple shuttle missions.

⍰ Case Study 3: Open-Source Developer Burnout

* Many popular open-source projects rely on a **single maintainer** (e.g., `left-pad` incident in 2016).

* When maintainers step away, projects can break for **millions of downstream developers**.

* Example: The maintainer of `event-stream` (a widely used npm package) handed control to a malicious actor because of burnout → resulted in a **supply-chain attack**.

* This is essentially a **bus factor = 1 risk**.

✅**Lesson Across All Cases:**

A low bus factor = fragility.

A high bus factor = resilience, continuity, and lower risk.

Do you want me to also explain **how companies actually *measure* bus factor** in practice (there are some research papers and industry techniques for this)?