Chapter 3 - Knowledge Sharing

Contents:

Overview:

This chapter deals with leveraging expertise, creating mechanisms for knowledge distribution, and fostering a culture of learning built on psychological safety.

- Organizations understand their **problem domain** better than outsiders.
- Most questions can be answered **internally** with the right expertise.
- Key requirements:
    - **Experts** who know the answers.
    - **Knowledge-sharing mechanisms** (Q&A, documentation, tutorials, classes).
- Success depends on building a **culture of learning**, supported by **psychological safety** that allows people to admit gaps in knowledge.

Challenges to Learning:

Sharing expertise across an organization is not an easy task. Without a strong culture of learning, challenges can emerge:

- ➢ **Lack of psychological safety**: An environment in which people are afraid to take risks or make mistakes in front of others because they fear being punished for it.
- ➢ **Information islands**: Knowledge fragmentation that occurs in different parts of an organization that don’t communicate with one another or use shared resources. In such an environment, each group develops its own way of doing things.
- ➢ **Single point of failure (SPOF)**: A bottleneck that occurs when critical information is available from only a single person.

Challenges to Learning (cont…):

- **All-or-nothing expertise**: A group of people that is split between people who know “everything” and who are “new”, with little middle ground. This problem often reinforces itself if experts always do everything themselves and don’t take the time to develop new experts through mentoring or documentation.
- **Parroting**: Mimicry without understanding. This is typically characterized by mindlessly copying patterns or code without understanding their purpose.
- **Haunted graveyards**: Places, often in code, that people avoid touching or changing because they are afraid that something might go wrong.

An enormous part of learning is being able to try things and feeling safe to

Haunting Graveyard

Haunted graveyards in code is a powerful metaphor that developers often use when talking about the remnants of old, forgotten, or dangerous code that continue to live in a project, quietly causing problems.

- Dead Code: Functions, classes, or modules no one calls anymore, but still linger in the repo. They waste space, confuse new developers, and sometimes get accidentally revived.
- Zombie Features: Half-built features hidden behind flags or commented-out code, never fully removed. Developers keep tripping over them, unsure if they should be revived or buried.
- Ghost Dependencies: Old libraries or packages still listed in `requirements.txt` or `package.json`, but not really used. These ghosts can bring in security vulnerabilities or bloat builds.

Haunting Graveyard Contd…
Why they’re haunted

- Fear of touching them: “If I delete this, will the whole system break?”
- Knowledge gap: Original authors are long gone.
- Unexpected side effects: Removing one “dead” function sometimes wakes up hidden dependencies.
- Accumulated technical debt: Slows development, increases bugs.

Philosophy of Software Engineering

- Every expert was once a new to organization, and organization must invest in developing their people.
- **One-on-one expert knowledge -** help is valuable but doesn’t scale effectively as if expert is unavailable the knowledge cannot be provided by others, and thus requires different ways to share knowledge among the people of organization.
- **Documented Knowledge** - Written documentation (e.g., team wikis) scales better, but requires maintenance and may lack personalization to one area.
- **Tribal knowledge -** is the gap that exists in between what individual team members know and what’s documented - it's vital to capture and share it.
- Tribal and written documentation **complement each other**; both are necessary.

Psychological Safety and Mentorship in Software Teams

◆ Why Psychological Safety Matters

- Learning begins by acknowledging knowledge gaps - honesty must be welcomed.
- Safe environments encourage risk-taking, asking questions, and admitting mistakes.
- Google research shows psychological safety is the **#1 factor for team effectiveness**.
- Psychological safety is the foundation for fostering a knowledge-sharing environment.

◆ Mentorship as a Foundation

- New members are paired with **non-manager mentors** for guidance and safety.
- Mentors help navigate tech, tools, and company culture without judgment.
- Encourages continual learning across all levels, not just newcomers.

Growing Your Knowledge: A Culture of Continuous Learning

- **Always Be Learning**
  - Learning is ongoing - even senior engineers don’t know everything.
  - Asking questions is essential; don't fear appearing inexperienced.
  - Rely on coworkers as a valuable source of knowledge.
  - Leaders must model curiosity and vulnerability to normalize learning.
- **Create a Safe Space for Questions**
  - Normalize asking “trivial” questions without shame.
  - Kindness and patience when responding foster psychological safety.
  - Targeted help reduces ramp-up time and boosts team productivity.

Growing Your Knowledge: A Culture of Continuous Learning (cont…)

◆ Understand the Context Before Acting

- Learning includes *why* something was built a certain way.
- Don’t rush to change legacy systems — ask: “What problem was this solving?”
- Apply **Chesterton’s Fence**: don’t remove things you don’t yet understand.
- Evaluate changes *after* understanding the historical and technical rationale.

◆ Make Learning Scalable

- Document context and rationale behind decisions, not just the decisions.
- Context-aware engineers contribute more effectively to long-term design and stability.

Scaling Your Questions – Ask the Community

- Write down what you learn - for your future self and others.
- Seek help from the **community**, not just individuals.
- Sharing knowledge makes it accessible to a broader audience.
- Community-Based Learning Tools
    - **Group Chats**
    - **Mailing Lists**
    - **YAQS (Internal Q&A Platform)**

What is Documentation?

- Written knowledge aimed at helping readers learn.
- Not all written content qualifie - e.g., mailing list threads may not count.
- Focus: contributing to and creating **formal documentation**.
- Ranges from fixing typos to documenting complex tribal knowledge.

Creating Documentation:

  - As expertise grows, document new workflows or processes.
  - Share your learning with others through well-written guides.
  - Make docs **discoverable** - unsearchable docs are ineffective.
  - g3doc: documentation lives beside source code, improving visibility.

Documentation (cont…)

Updating Documentation:

- **Best time to improve docs:** when learning something new.
- Fix mistakes or missing steps you encounter.
- Empowered culture: update docs even if owned by another team.
- Google uses **g3doc** to simplify edits and maintain version history.

Feedback Mechanisms:

- Important to have a **way to report outdated or incorrect docs**.
- Encourage feedback to prevent recurring issues for others.
- At Google:
  - Users can file documentation bugs directly.
  - Leave comments on g3doc pages.
  - Comments auto-file bugs to the doc owner - no need to find them.

Documentation (cont…)

Promoting Documentation:

- Engineers often reluctant due to:
    - Time investment
    - Delayed and indirect benefits
- Structural incentives help - but not always sufficient.

Personal Benefits of Documentation:

- Saves time: answer once, then point to the doc later.
- Helps **team scale**:
    - Central reference for repeated questions.
    - Others can update and share the doc.
    - May benefit other teams with similar needs.

Code as a Knowledge - Sharing Tool

- **Code = Transcribed Knowledge**: Even without intent, writing code often shares insights.
- **Readable Code Enhances Knowledge Transfer**: Benefits both **consumers** and **future maintainers**.

Implementation Comments:

- Share reasoning and intent with future readers (including “Future You”).
- Risk: Become outdated if not maintained.

Code Reviews as Learning Platforms:

- Authors and reviewers gain new techniques and tools.
- Review suggestions can reveal new patterns or libraries.
- At Google, the **readability process** standardizes mentorship through reviews.

Readability at Google: Mentorship Through Code Review

What Is Readability? A company-wide mentorship program to ensure **clear, idiomatic, and maintainable code** through structured reviews.

The Readability Process:

- Mandatory **code review** for every changelist (CL)
- **Certified engineers** approve CLs or act as reviewers
- Focus on **mentorship**, not gatekeeping

Benefits:

- Promotes **company-wide code consistency**
- Scales **best practices and language idioms**
- Increases **code quality & maintainability**
- Encourages **collaboration and learning**
- Engineers report **higher code quality** and **learning value**

Readability at Google: Mentorship Through Code Review

Challenges:

- Requires external reviewers if team lacks readability-certified members.
- **Human-driven** → scales linearly
- **Short-term latency** vs. **long-term quality gains**