Alexander Polomodov

Technical Director @ T Tech

Reliability and security: are they additional options or the foundation for modern IT systems?

PHDAYS FEST

OT ∎ positive technologies

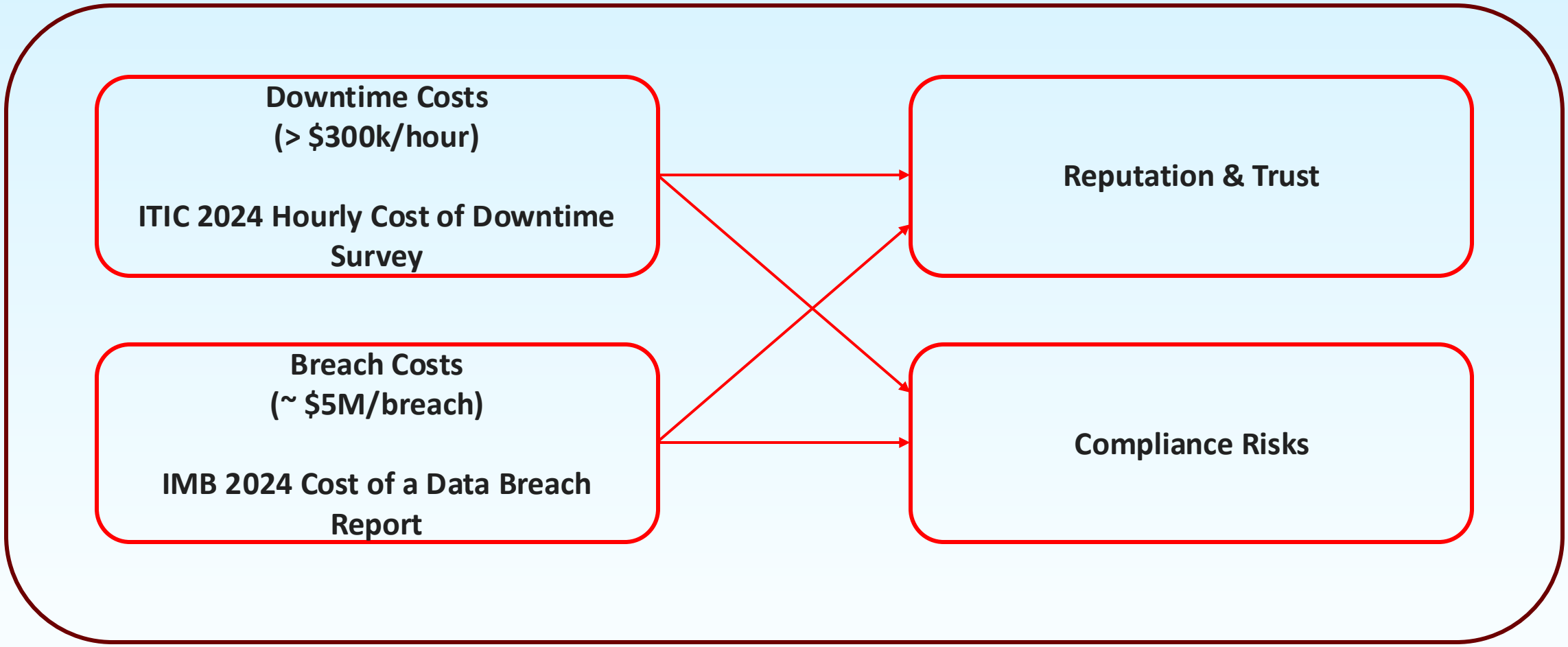
Agenda

- **Framing the Problem:** Are reliability & security optional foundational?
- **Core Concepts:** Risk modeling, ATAM, threat modeling, emergent properties
- **Industry Practices:** “Secure by design”, “Shift left everything”
- **Case studies:** Google, Netflix, AWS
- **Recommendations:** Strategies, cultural shifts, and actionable best practices
- **Conclusion:** Key takeaways, next steps, and discussion questions

01

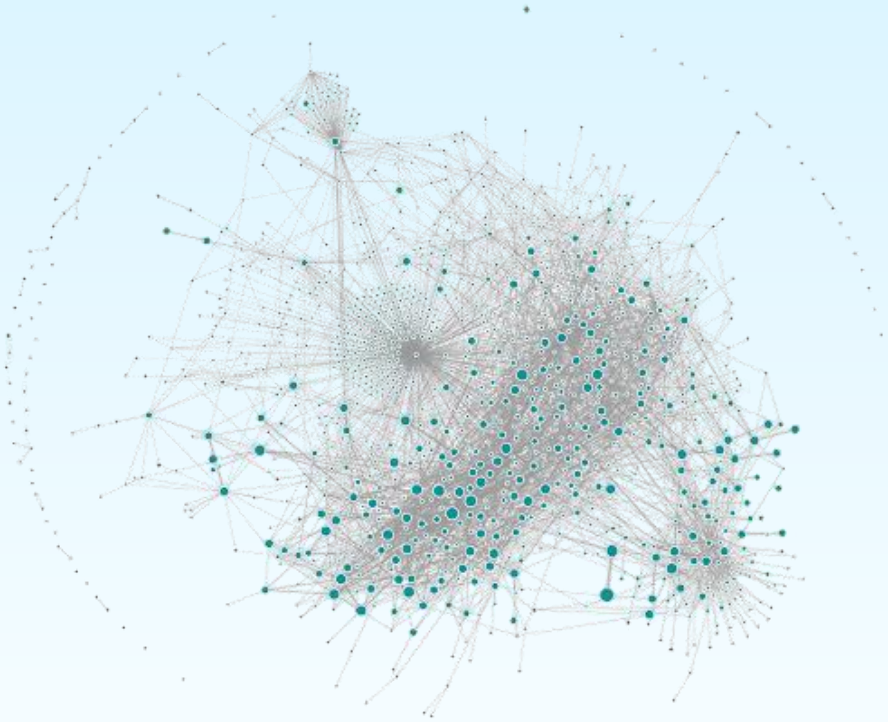
Framing the problem

What's at Stake?



Modern IT Landscape – Complexity & Risk

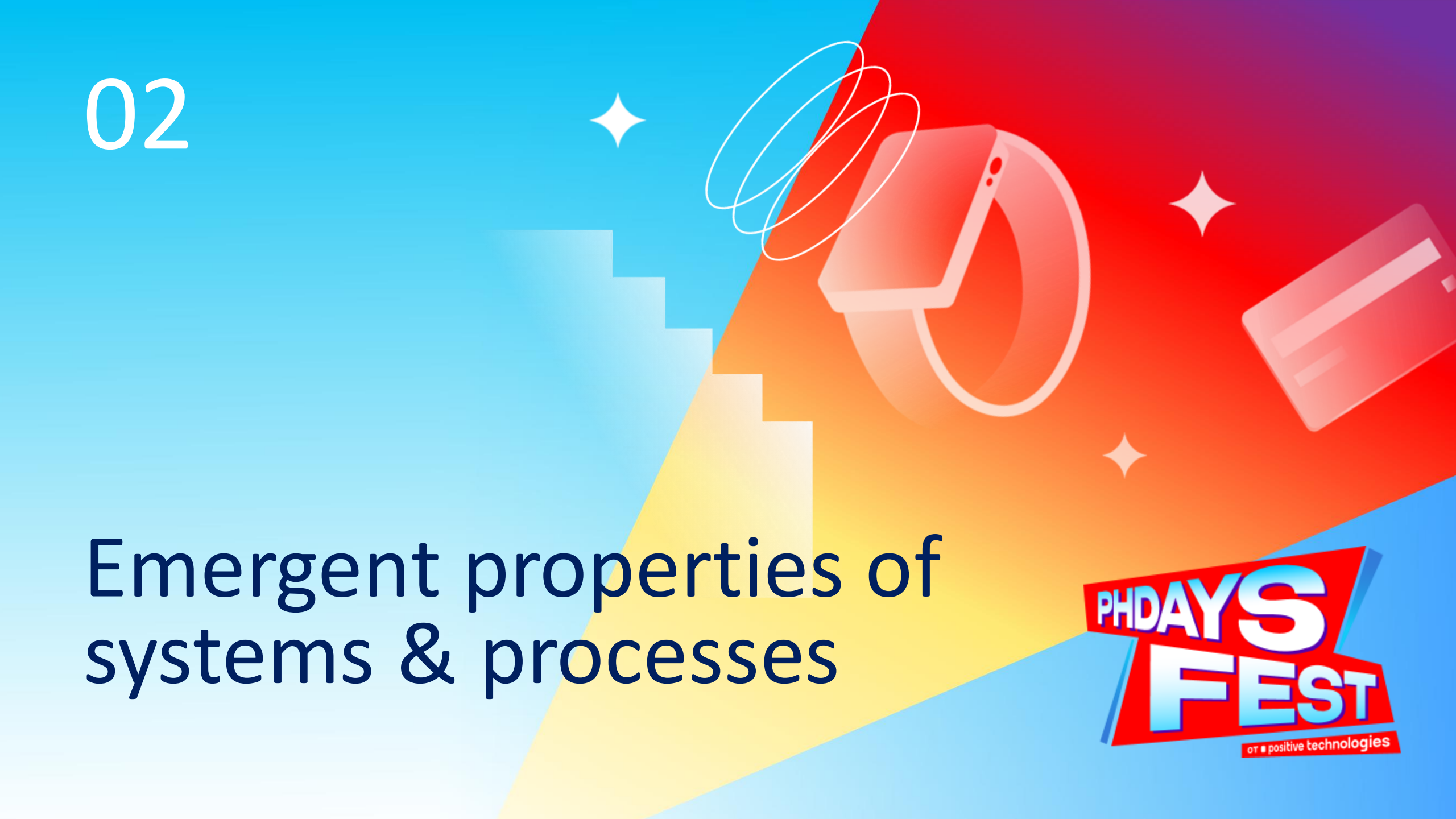
- **Hyper-Connected Systems:** Cloud, microservices, and IoT mean more integration. And **“limitless attack surface**
- **System Complexity:** Distributed architectures have emergent behaviors. Small bugs can cascade into large outages; security gaps in one service can open doors to entire networks.
- **Rapid Change:** Continuous deployment and frequent updates increase speed, but raise the risk of pushing unstable or insecure code if safeguards are lacking.
- **Threat Environment:** Cyber attacks are more sophisticated and frequent. A modern system must assume adversaries and failures will occur, and be prepared for both.



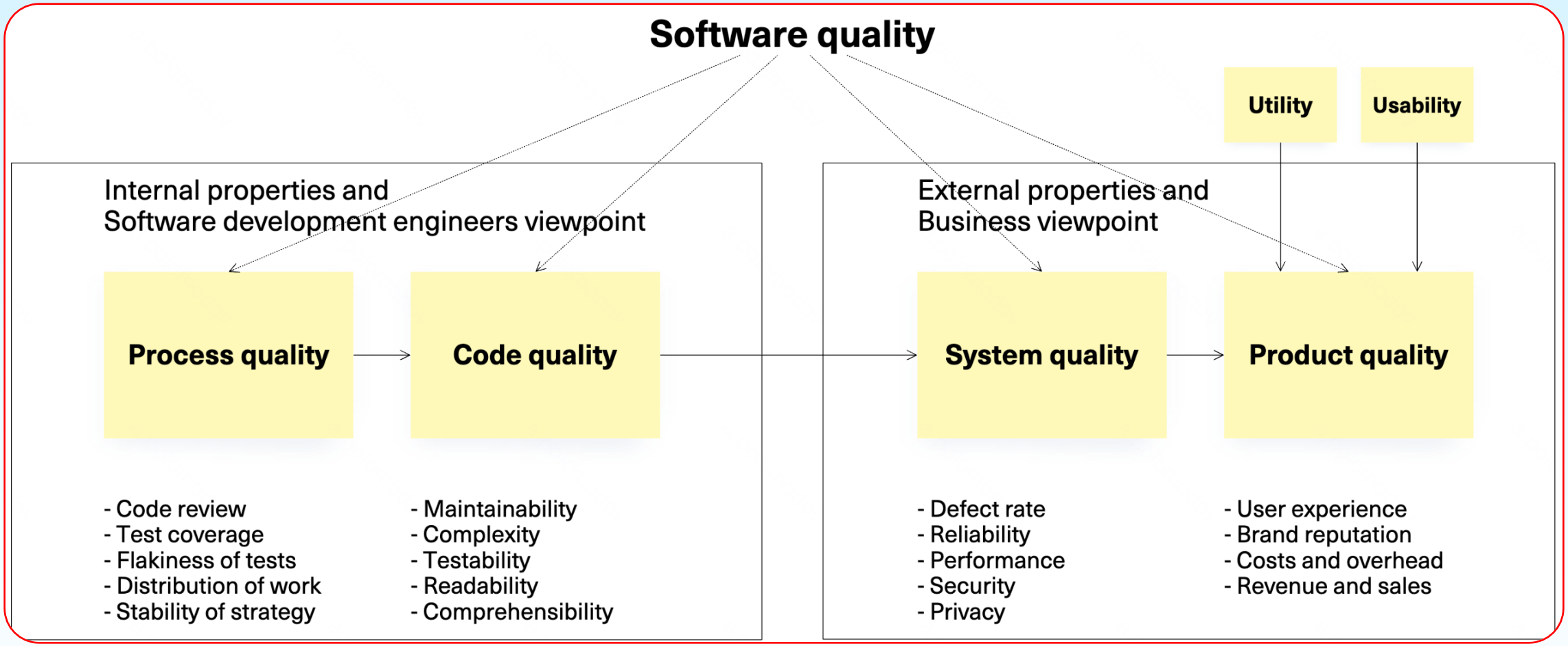
Uber’s microservice architecture circa mid-2018 from Jaeger

02

Emergent properties of systems & processes



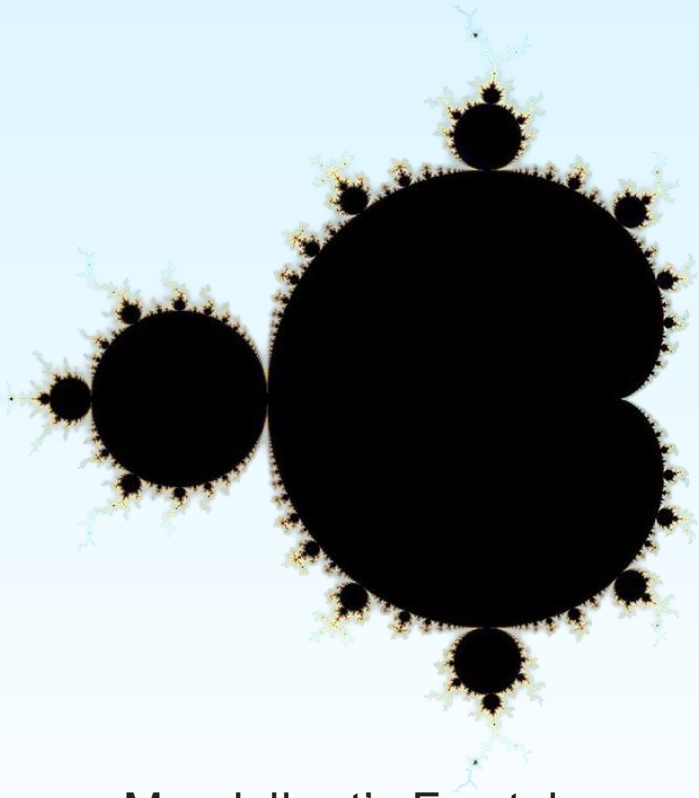
Software Quality Model



Based on whitepaper “Developer productivity for Humans, Part 7: Software Quality” from Google

Reliability & Security as Emergent Properties

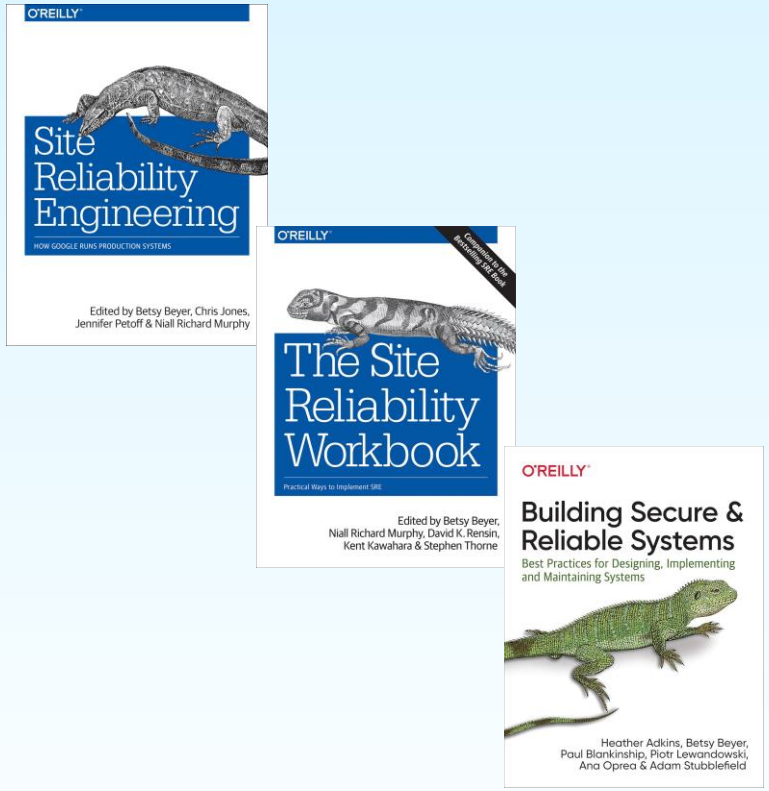
- **Not Plug-and-Play:** You can't buy a "reliability module" or a single "security feature" to guarantee success. Reliability is an emergent property of the entire system's design, development, and operations. Security is similarly emergent – no single component provides it in isolation.
- **Whole-System Design:** These qualities result from many factors working together. *Example:* Reliability emerges from how you partition services, handle dependencies, do testing, and monitor systems. Security emerges from component trust relationships, choice of frameworks, secure coding practices, ...
- **Cross-Cutting Concerns:** Because they are emergent, reliability and security must be woven through every layer – from code to infrastructure to processes. A flaw at any layer (an unreliable network call or an unpatched server) can undermine the whole.
- **Implication: Foundational, not Optional.** If you don't architect and cultivate these properties, you can't simply tack them on later.



Mandelbrot's Fractals

Reliability by Design – Engineering for Failure

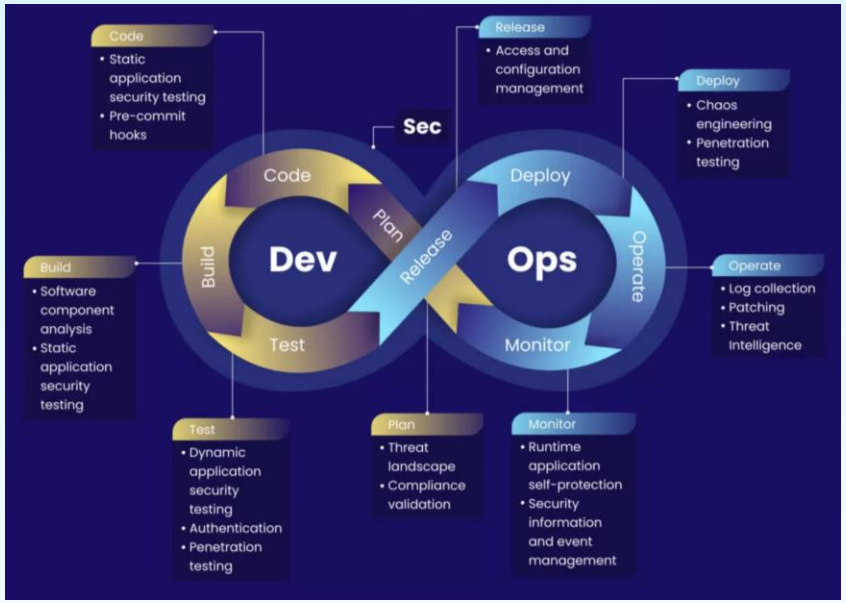
- **Design for Resilience:** Reliability isn't a feature toggle – it must be architected in. Build redundancy (e.g., failover servers, replicated data), graceful degradation (the system limps instead of crashes), and fault tolerance from the start.
- **Anticipate Failure:** Assume components *will* fail. Incorporate patterns like circuit breakers, bulkheads, and backpressure to prevent one failure from cascading. "Always on" means designing for the worst day, not just the best day.
- **Load & Performance Planning:** Include capacity and performance considerations in design. Set Service Level Objectives (SLOs) for uptime/latency early and ensure the architecture can meet them under stress.
- **Integration in Dev Cycle:** Shift reliability testing left – e.g., perform chaos testing in staging, do load tests pre-launch, …



Popularized as SRE by Google

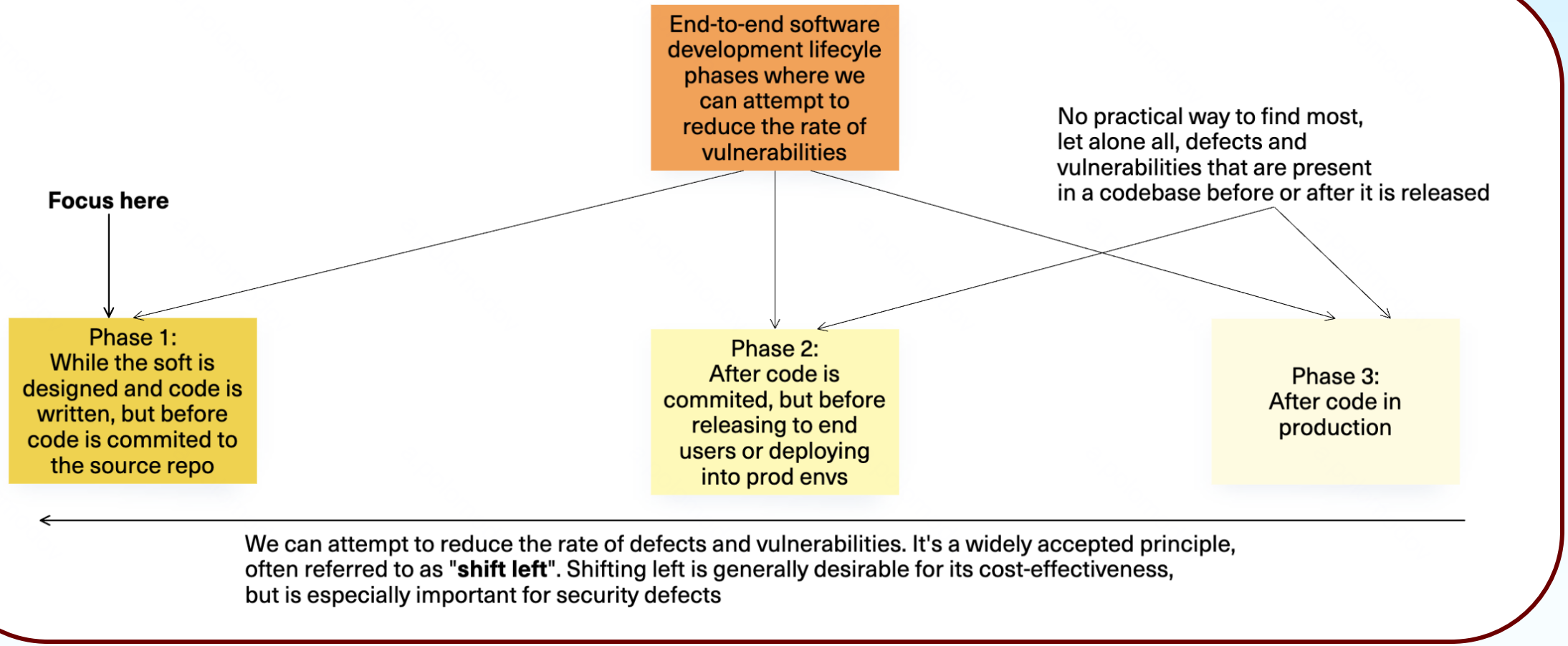
Secure by Design & Shifting Left

- **Secure by Design:** Embed security *from the earliest design phases*. Security requirements are treated as fundamental as functionality. No “afterthought” modules – every design decision considers security implications
- **Shift Left:** Move security and testing activities earlier in the development lifecycle. Incorporate code reviews, threat modeling, and automated security tests in design & development, not just before release.
- **Benefits:** Catching vulnerabilities early saves time and money – addressing flaws during design is far cheaper than post-release. Studies show fixing a design-stage flaw after deployment can require exponentially more effort.
- **Culture Change:** Developers and architects collaborate with security from day one. Security isn’t a “gate” at the end; it’s a continuous process throughout



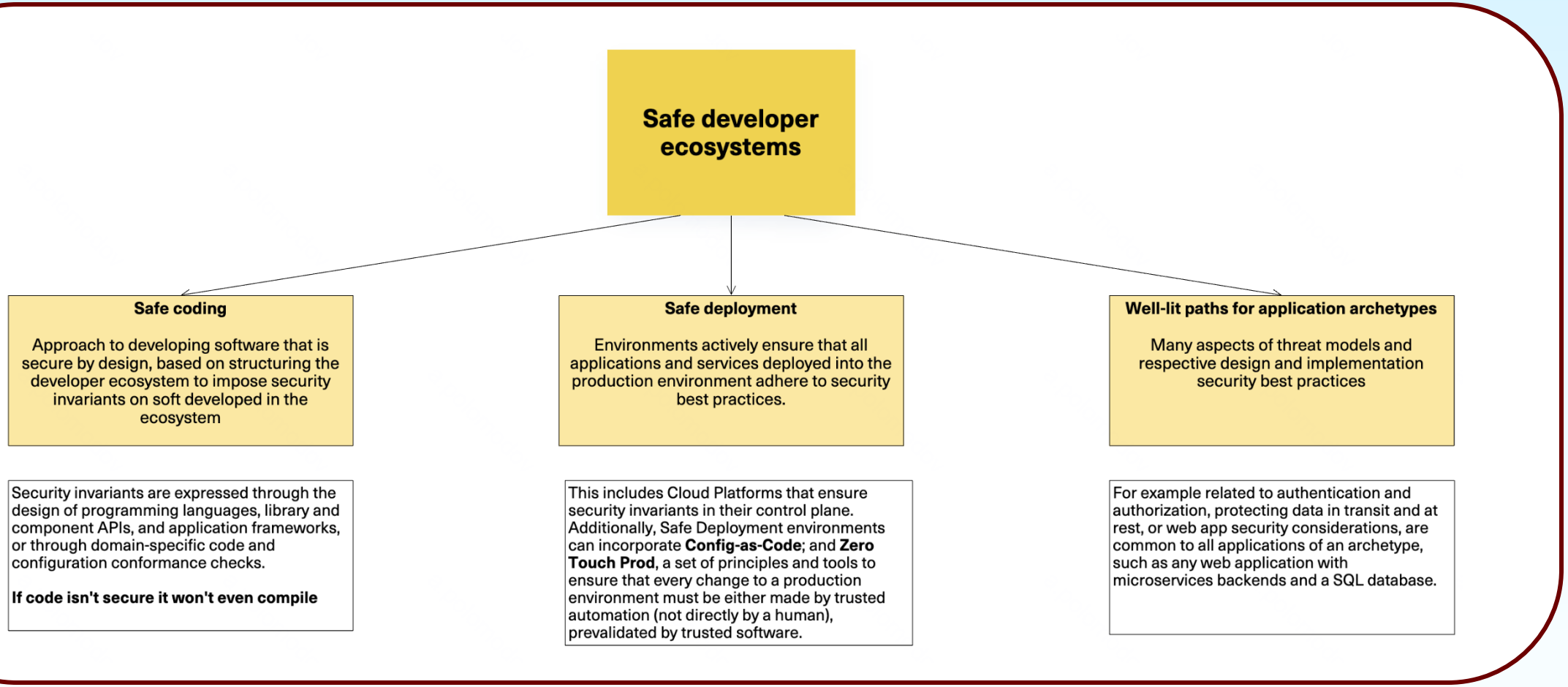
From digitalmara.com

Secure by Design at Google



Based on whitepaper “Secure by Design at Google”

Secure by Design at Google



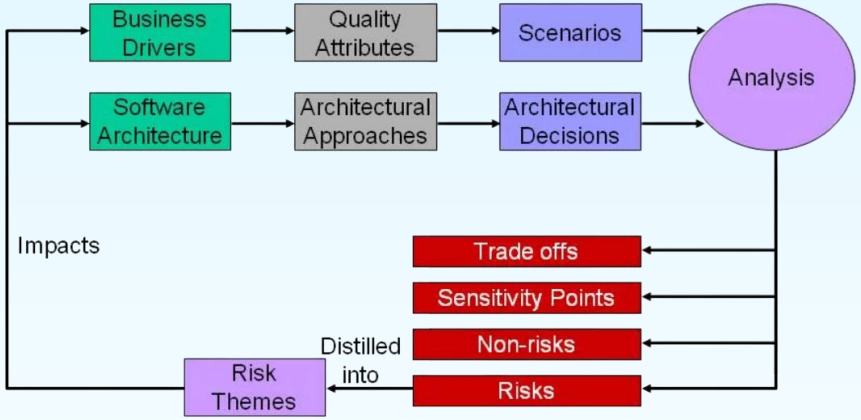
Based on whitepaper “Secure by Design at Google”

03

Architecture processes

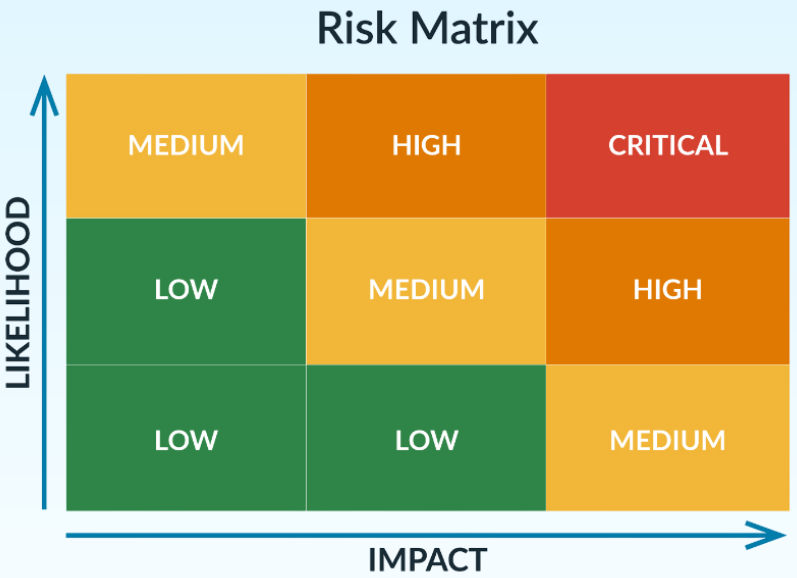
Architecture Trade-off Analysis (ATAM)

- **What is ATAM?** It is a structured technique to evaluate software architecture decisions against multiple quality attributes (performance, **availability/reliability, security**, …)
- **Trade-offs & Risks:** ATAM brings stakeholders together to examine how architectural choices impact quality attributes. It identifies trade-offs (e.g., a design that improves performance may hurt security) and surfaces risks early.
- **Balancing Qualities:** Using ATAM, teams can debate reliability vs. other concerns with data. For example, adding robust failover might slightly increase cost or complexity – ATAM helps justify that investment by clarifying risk impact.
- **Outcome:** The process yields *risk themes* and sensitivity points – guiding where mitigation is needed. ATAM essentially bakes reliability and security into architecture discussions, ensuring they are foundational considerations, not forgotten requirements.



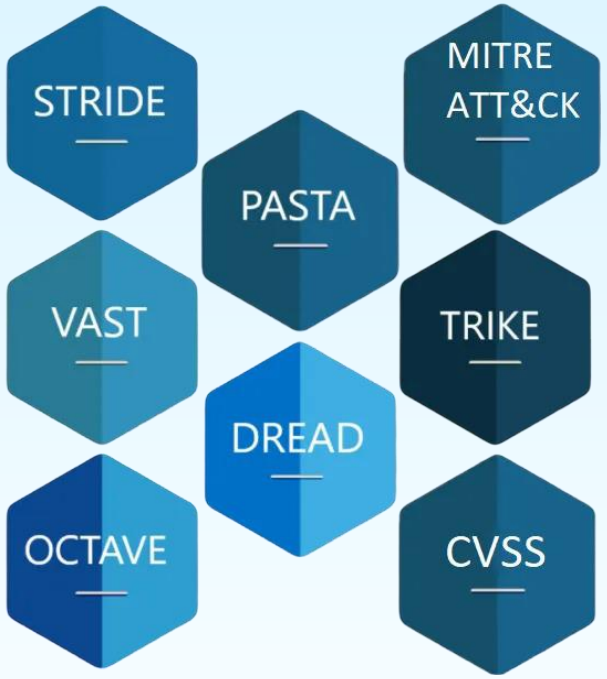
Risk Modeling Guiding Decisions

- **Purpose:** Risk modeling means systematically identifying potential failure scenarios or threats, and estimating their likelihood and impact. This applies to both reliability (e.g., system outages) and security (e.g., data breaches).
- **Approach:** Use tools like risk matrices or FMEA (Failure Mode & Effects Analysis) for reliability, and risk assessments for security. Quantify risk = *Probability × Impact* for each scenario. This helps prioritize what to address first.
- **Prioritization:** Focus on high-impact, high-likelihood risks. For example, if a single point of failure could take down a service, that's a high risk to mitigate; similarly, a common vulnerability in your stack is a high security risk to fix.
- **Outcome:** A risk model informs investment – it provides an executive view of where reliability and security efforts will avert the most damage. It turns abstract "worries" into concrete data for decision-making and resource allocation.



Threat Modeling Foresight in Security

- **What is Threat Modeling?** A structured approach to identify and address potential threats early in design. Teams systematically think through “What can go wrong?” in terms of security, for a given system or feature.
- **Process:** Identify critical assets (e.g., customer data), potential adversaries and attack vectors, and system vulnerabilities. Common frameworks like **STRIDE** (Spoofing, Tampering, Repudiation, Information Disclosure, DoS, Elevation of Privilege) guide brainstorming.
- **Value:** By modeling threats, you can prioritize mitigations before coding. For instance, if threat modeling a web app reveals SQL injection risk, developers can add input validation and parameterized queries from the outset rather than after an attack.
- **Secure Design Integration:** Threat modeling embodies *“secure by design”* – it’s often done alongside design specs. It helps communicate possible damage from flaws and drives teams to build in defenses



From Dark Roast Security

04

Case studies

Google, Netflix, AWS

Google: Reliability at Scale

- **SRE and SLOs:** Google treats reliability as a primary feature. Site Reliability Engineering (SRE) teams set Service Level Objectives (SLOs) for uptime/latency and use error budgets as a tool to balance innovation with reliability
- **Design Reviews:** Google's internal design documents mandate sections on reliability and security for every project. Before launch, teams undergo Production Readiness Reviews (for reliability) and security reviews
- **Emergent Approach:** At Google's scale, it's understood that reliability emerges from design and process. They invest heavily in testing (from unit tests to large-scale load tests) and monitoring/alerting. There is a culture of blameless postmortems – learning from each failure to improve system design continuously.
- **Outcome:** Google can offer products with very high availability because they've made reliability a core engineering requirement, governed by data and disciplined practices.

"Hope is not a strategy"

Benjamin Treynor Sloss, ex VP of Eng at Google

Netflix: Resilience through Chaos

- **Chaos Engineering:** Netflix famously introduced *Chaos Monkey* – a tool that randomly disables servers in production to ensure the system can survive failures. This proactive "failure injection" tests resilience continuously. Netflix designs systems assuming things will break, so services must heal themselves or degrade gracefully.
- **Cultural Mindset:** At Netflix, engineering teams are encouraged to be "highly autonomous" but responsible – the freedom comes with the expectation of building robust, resilient services. They learn from controlled failure: every Chaos Monkey incident is a chance to discover weak points and fix them before real incidents occur.
- **Resilience Patterns:** Netflix's cloud architecture emphasizes redundancy across regions, fallback modes, and quick recovery. For example, they also use load shedding and prioritization to keep core functionality running under strain (as noted in Netflix tech blogs).
- **Outcome:** Despite huge scale (streaming to millions of users), Netflix achieves high uptime and performance. By treating reliability as a continuous experiment and investment, they minimize customer impact from failures.

AWS: Security is Job Zero

- **Security Culture:** Amazon Web Services treats security as the top priority, calling it “Job Zero” – meaning it comes even before the first priority. Every employee, from engineers to leadership, is responsible for security. There are weekly reviews led by the CEO where outstanding security issues are reviewed.
- **Built-In and By Default:** AWS builds security into its services (e.g., encryption is often enabled by default, rigorous identity and access management). New services at AWS do not launch if known security issues remain. Security teams work closely with development from the beginning of a project to avoid last-minute surprises.
- **Operational Excellence:** AWS also emphasizes reliability and operational excellence as foundational pillars (in AWS’s Well-Architected Framework, *Security* and *Reliability* are 2 of the 6 core pillars). They invest in fault-isolated infrastructure (multiple AZs/regions) so that services remain up even if one component fails.
- **Outcome:** AWS’s approach has enabled it to achieve high customer trust in running critical workloads. The strong top-down security focus means major incidents are rare, and when vulnerabilities (e.g., Spectre/Meltdown) arise, AWS can respond rapidly across its massive infrastructure.

“You build it, you run it”

Werner Vogels,
Amazon CTO

Key Principles from Big Tech

- **1. Security is Everyone's Responsibility:** Companies like AWS instill that *every* role has a part in security. Empower your team to take ownership (report issues, fix proactively) – security is not just the security team's job.
- **2. Reliability is a Feature:** At Google, reliability is planned, measured, and managed (via SLOs and error budgets) just like any feature. It's not negotiable – if reliability suffers, feature development stops until it's back on track.
- **3. Design for Failure:** Netflix's principle – assume things will break. Proactively test and prepare. This leads to architectures that can self-heal and degrade gracefully instead of catastrophically.
- **4. Leadership and Culture Matter:** In all examples, leadership set the tone (Google's early design reviews, Amazon's CEO-led security meetings, Facebook's motto change to "stable infra"). Executive support and a learning culture (e.g., blameless postmortems) encourage teams to prioritize these qualities without fear.
- **5. Continuous Improvement:** Big Tech treats reliability/security as a journey, not a one-time project. They invest in ongoing training, tooling, and iterative improvement (patching, monitoring, incident analysis) to stay ahead of new risks.

05

Case study at T Tech

PHDAYS FEST
OT positive technologies

Spirit – Principles of platform development

Repeatability and Transparency of the Code Lifecycle

The code always goes through the same stages and it is crucial that any developer can understand what is happening and which stages they need to complete.

Codebase Management and Quality Assessment at Every Stage

The quality of the codebase is one of the main characteristics of a technical product. The ability to maintain consistently high quality is a sign of mature and resilient development processes.

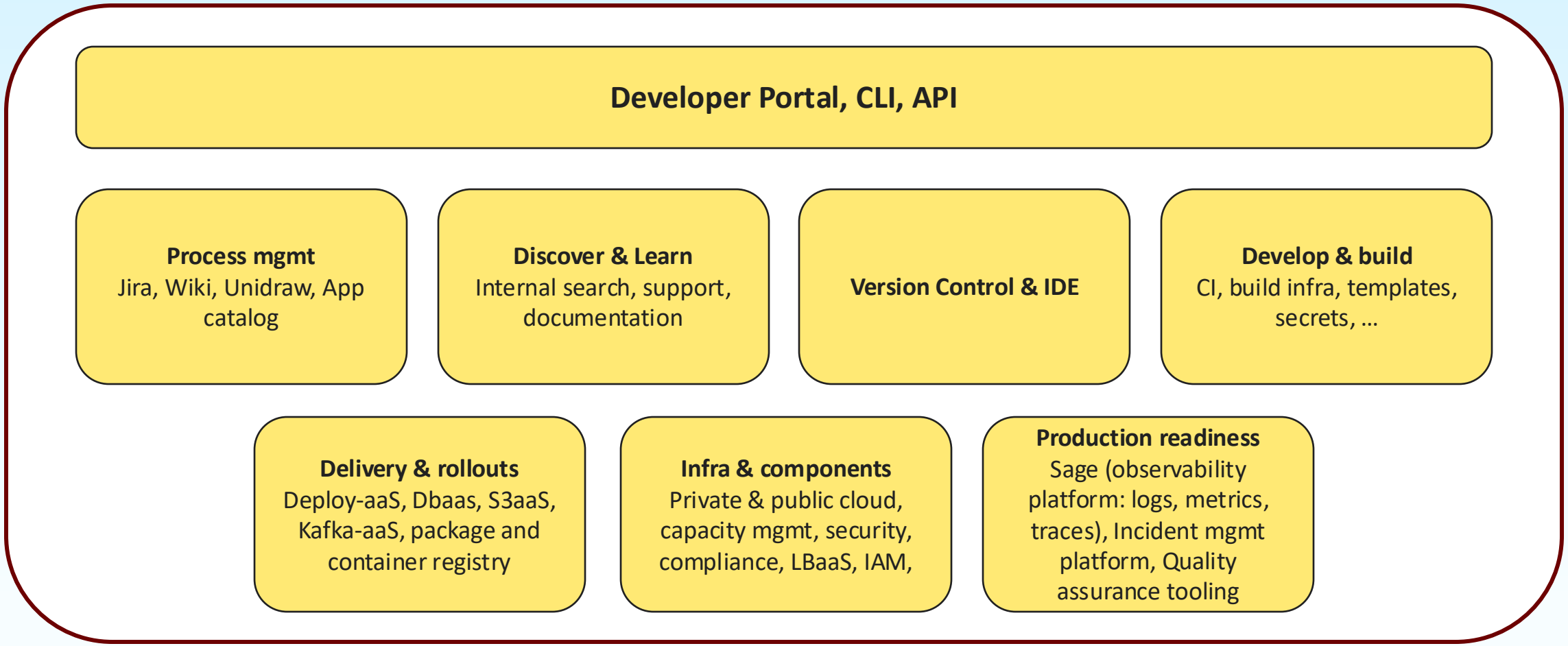
Transition to Inner Source within the Company

Reusing code and colleagues' solutions is a major advantage for the company. It helps reduce development costs and increase the speed of delivering business features. The platform should enable fast and seamless code sharing.

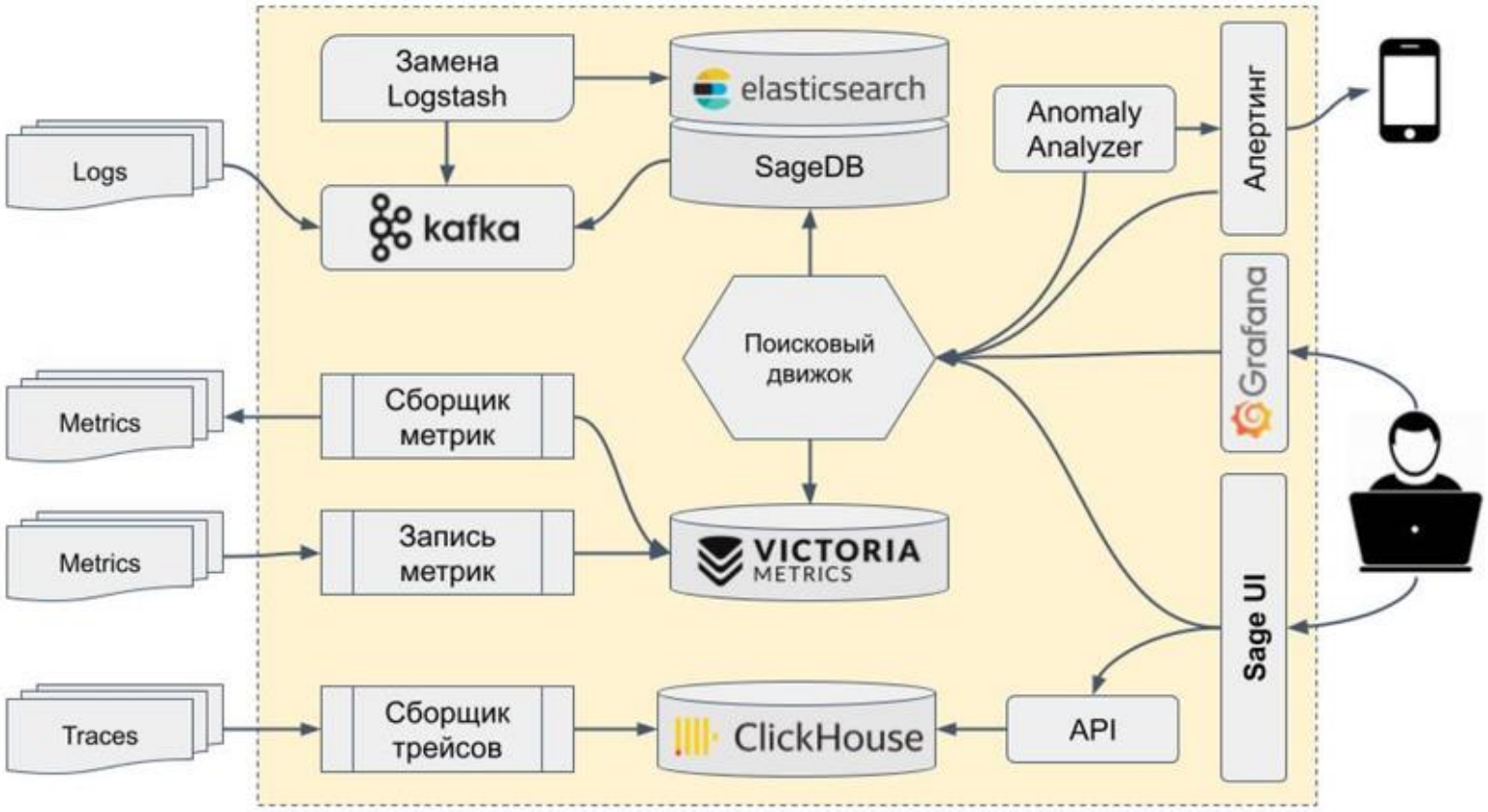
Focus on Results

When developers have all the necessary tools to manage the product lifecycle, it saves a lot of time. Our goal is for developers to spend 80% of their time on business features, without being distracted by routine or infrastructure complexities. Ultimately, this benefits the end users.

Spirit – Internal PaaS



Sage – observability platform



Finedog for incident mgmt

Service Catalog & Teams
Services are linked to others they depend on or influence. This allows owners to be notified of potential issues if an incident occurs in a related service.

Incident Detection

If monitoring detects an SLO violation, the system raises a suspected incident.

Incident Resolution

When a suspicion is confirmed or a problem is noticed manually, an incident is registered.

Subscriptions & Escalation Policies

Users select which events to be notified about and configure who receives alerts in case of incidents or suspicions.

Root Cause Analysis & Postmortems

After an incident is resolved, analysis continues to prevent recurrence.

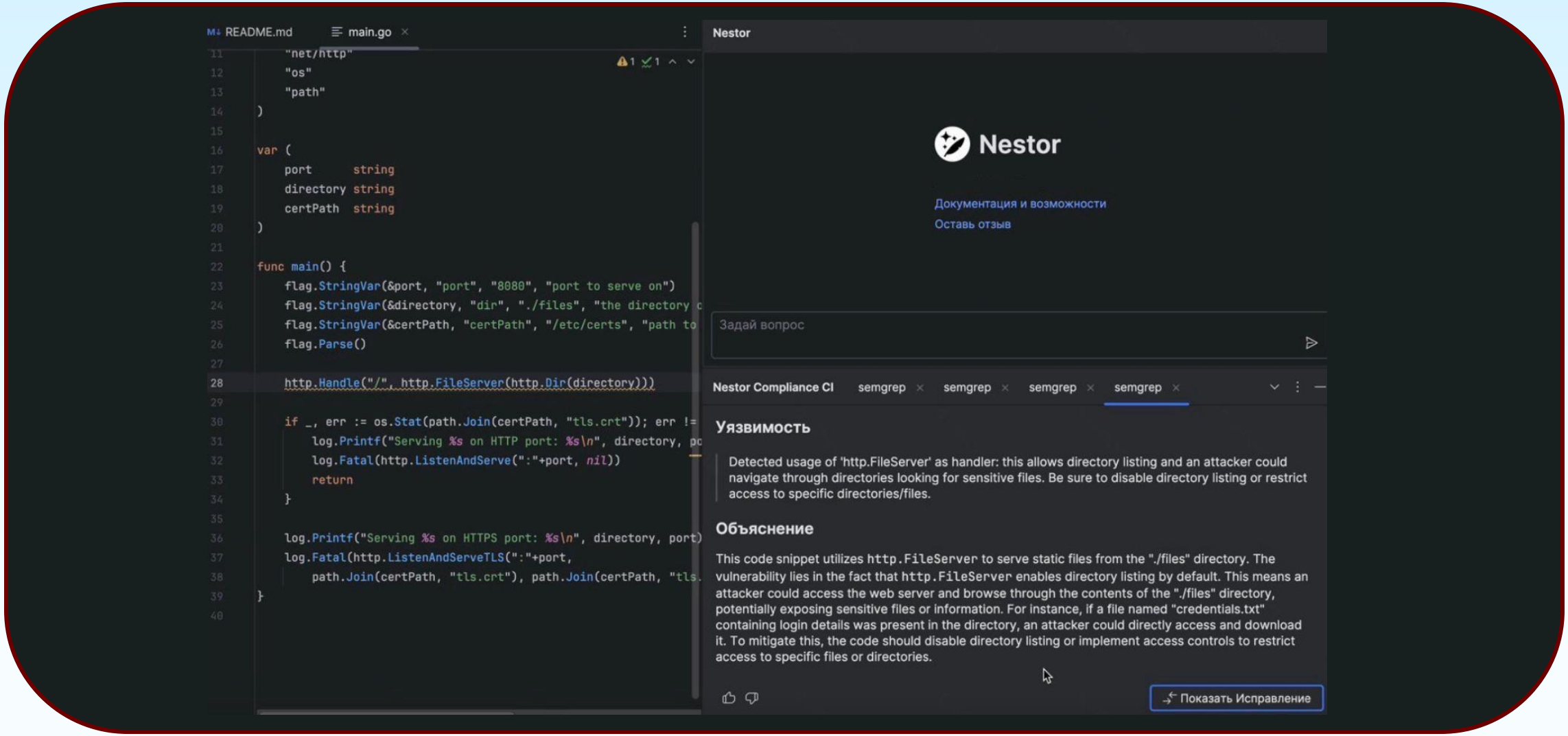
Analytics

FineDog offers dashboards for insight into company health, service status, and team performance.

Releases

Release planning and information help service owners assess incident risks and coordinate their own releases.

Nestor – AI copilot (with Safeliner)



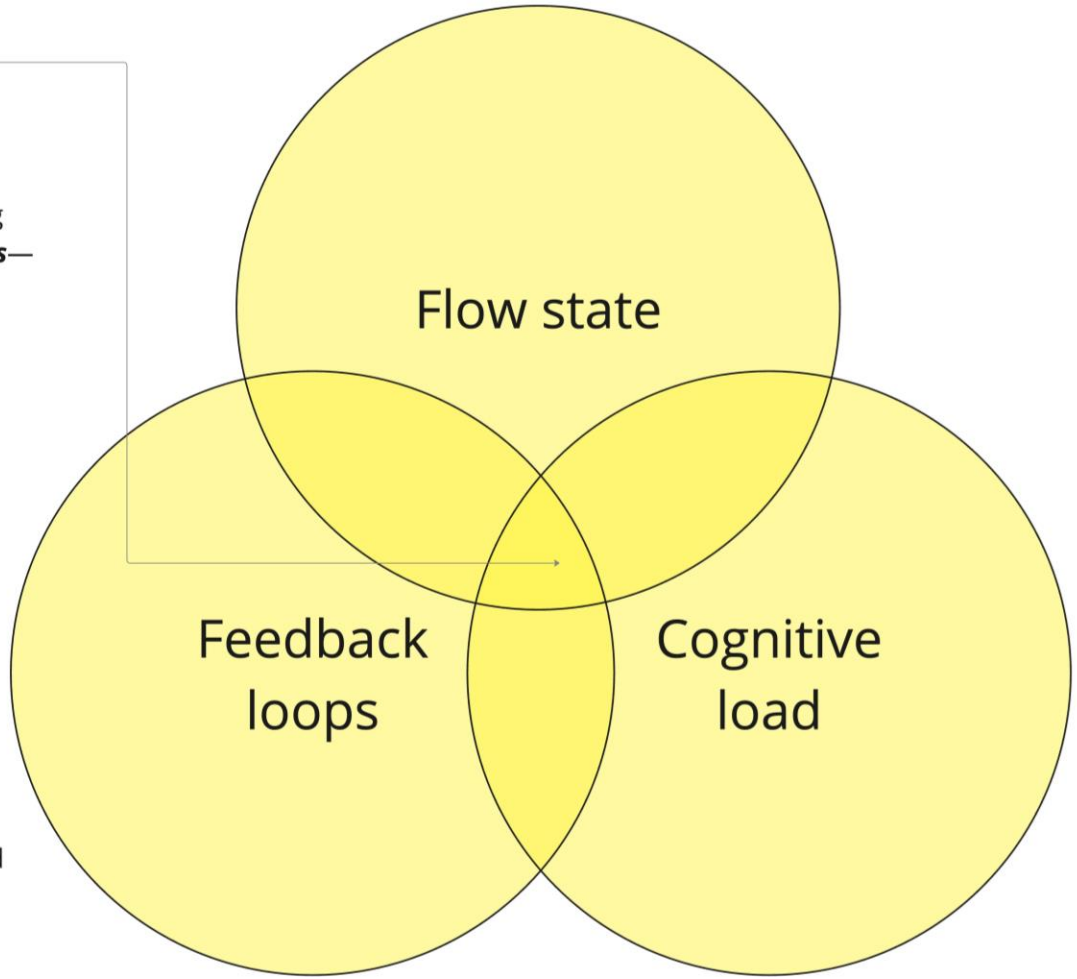
Spirit – Developer Experience

DevEx

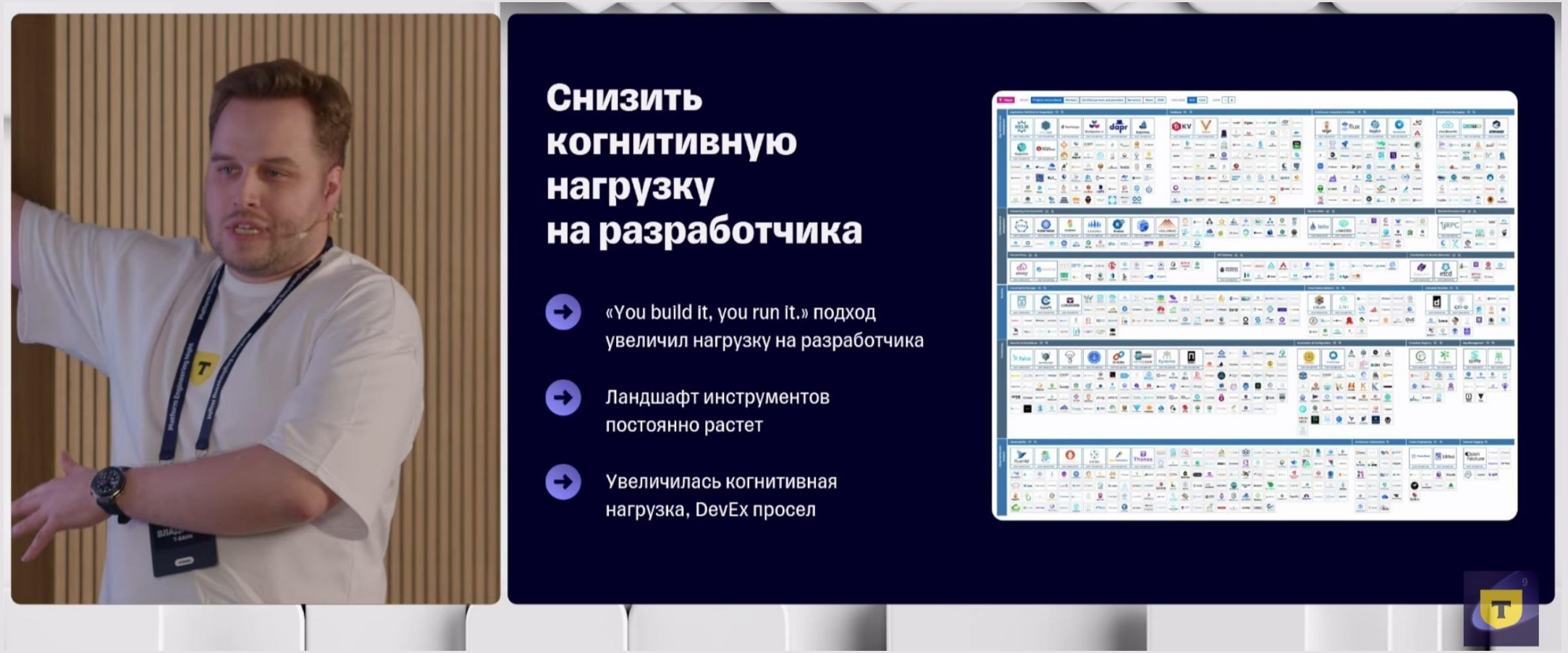
1) Software organizations commonly look for ways to optimize their value stream by reducing or eliminating delays in software delivery. Shortening ***feedback loops***—the speed and quality of responses to actions performed—is equally important to improving DevEx.

2) Software development is inherently complex, and the ever-growing number of tools and technologies is further adding to the cognitive load faced by developers. ***Cognitive load*** encompasses the amount of mental processing required for a developer to perform a task.

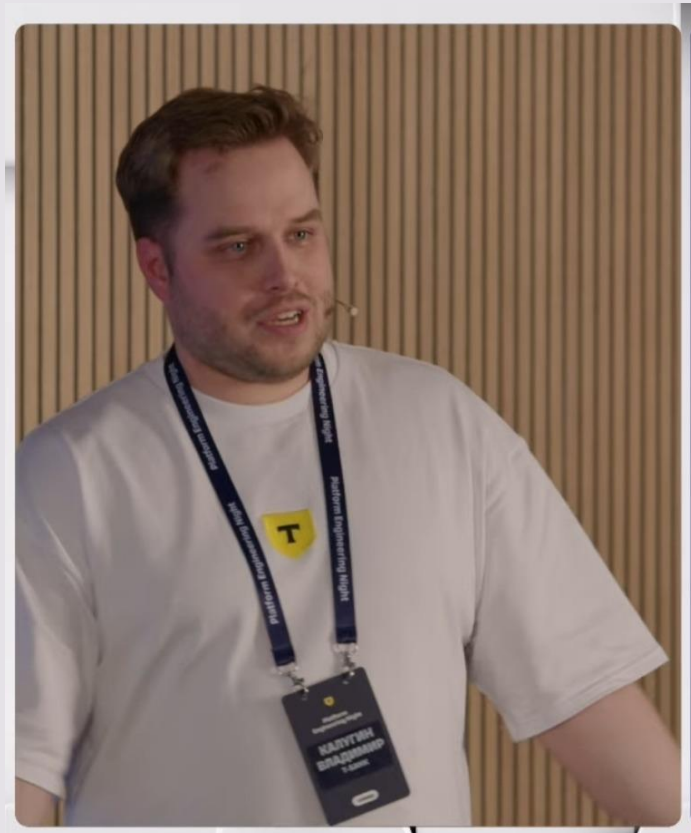
3) Developers often speak of "getting into the flow" or "being in the zone." Such statements colloquially describe the concept of ***flow state***, a mental state in which a person performing an activity is fully immersed in a feeling of energized focus, full involvement, and enjoyment.

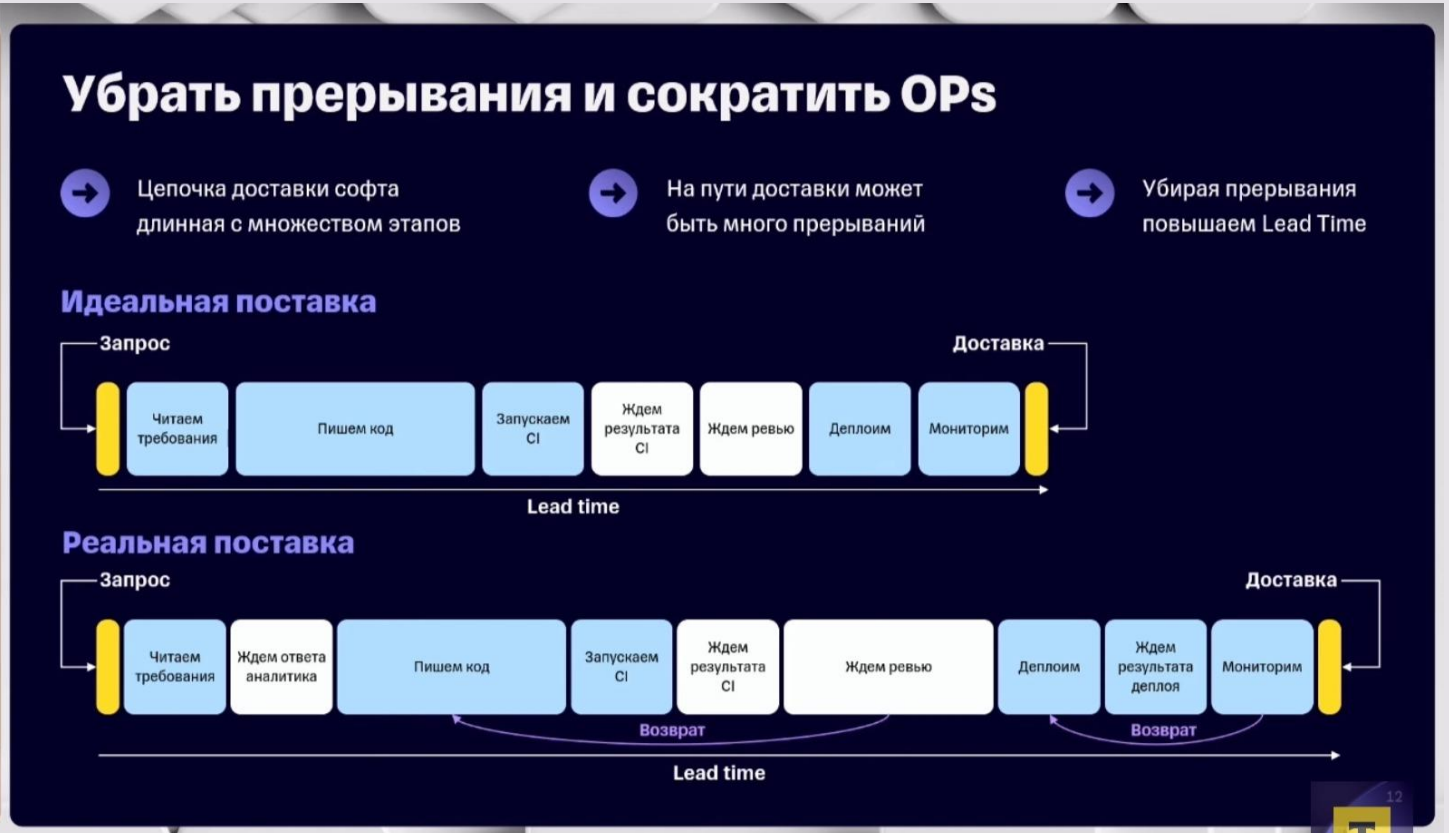


Spirit – Developer Experience

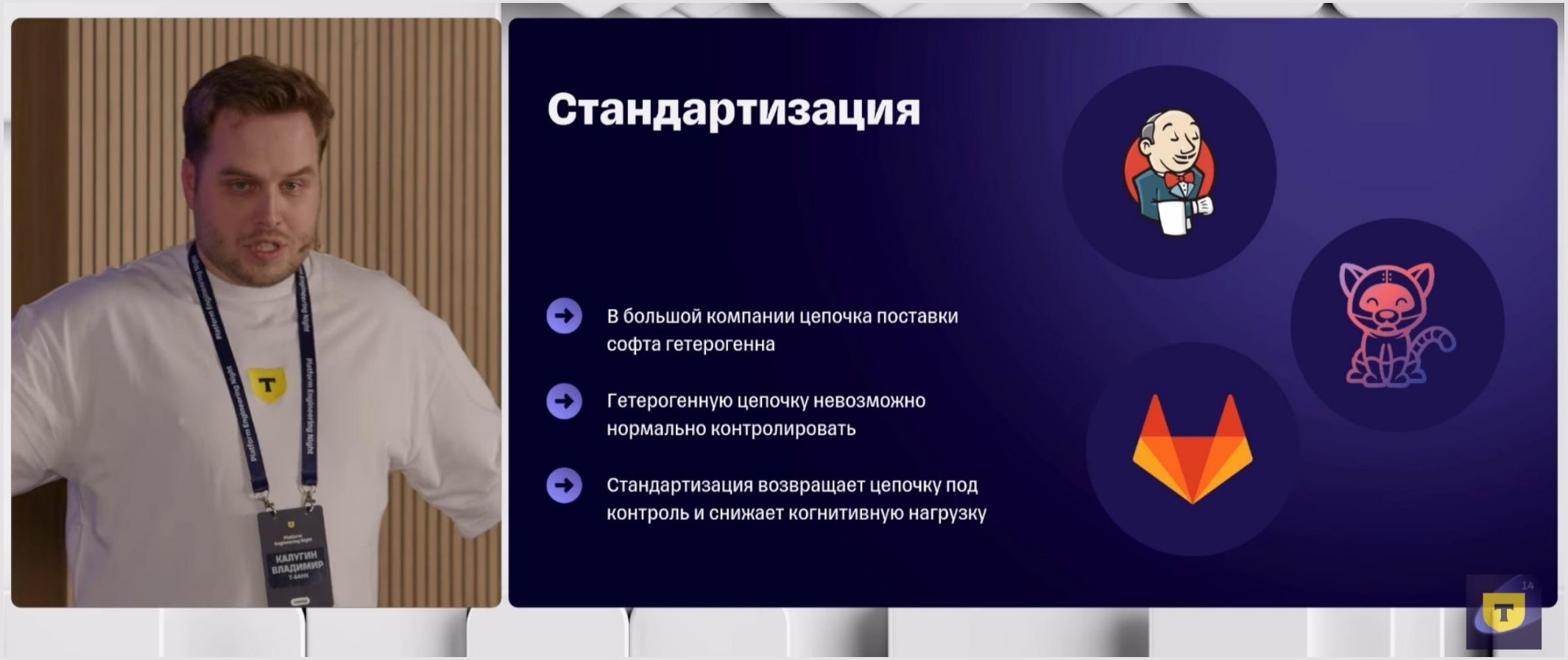


Spirit – Developer Experience

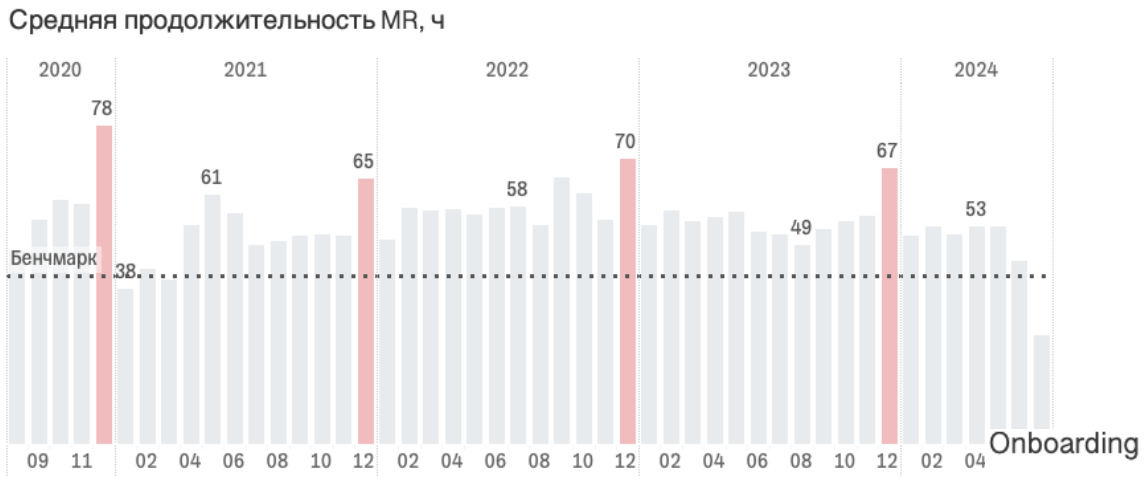




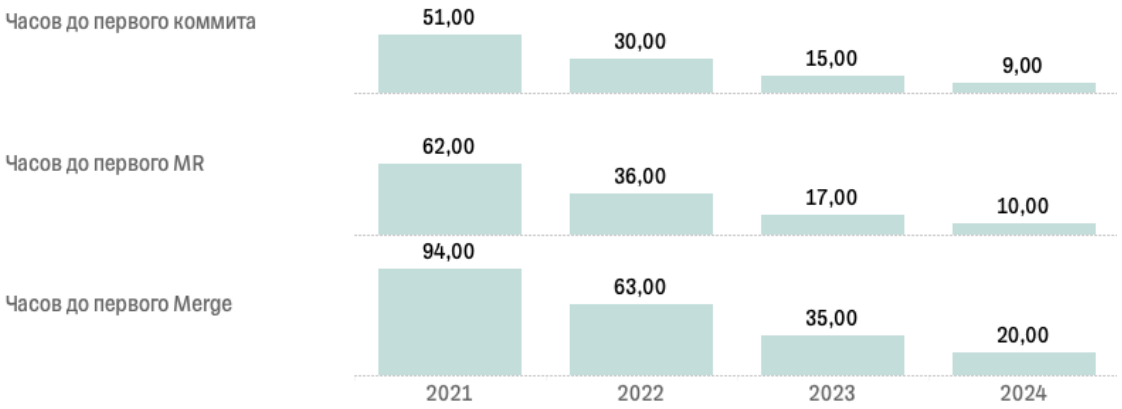
Spirit – Developer Experience



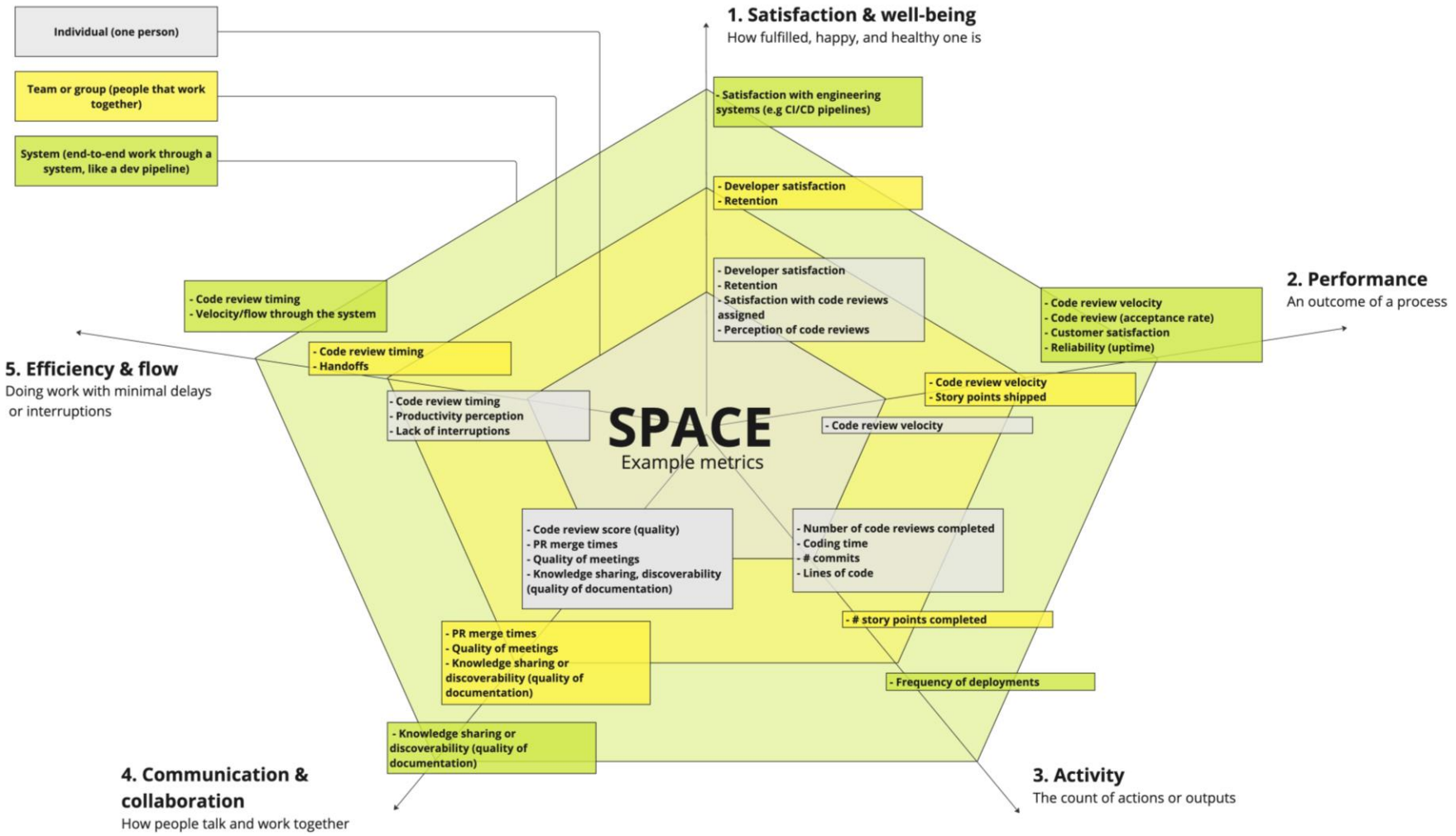
Spirit – DevEx Metrics



Onboarding



Spirit – SPACE



06

Recommendations

Strategy – Making Reliability & Security a Priority

- **Embed in Vision:** Clearly include reliability and security in the product/engineering strategy. For example, set a company vision statement that “We will be trusted for our uptime and data protection,” so it’s top-of-mind for all teams.
- **Goal Setting:** Define explicit goals/OKRs around uptime (e.g., “99.95% uptime for Service X”) and security (e.g., “Zero critical vulnerabilities open”). Tie these goals to business outcomes (customer satisfaction, SLA commitments, etc.).
- **Resource Allocation:** Dedicate budget and time for reliability and security work. This might mean allocating a percentage of each sprint or quarter to reliability improvements, tech debt reduction, and security enhancements. Make it a planned part of projects, not “if time permits.”
- **Risk Governance:** Establish governance where high risks (e.g., single points of failure, known security gaps) are regularly reviewed at the executive level. This mirrors how Big Tech does weekly reviews – it sends the message that leadership cares and progress will be tracked.

Culture & People – Everyone Owns Quality

- **Shared Responsibility:** Promote a culture where developers, ops, QA, and security teams collaborate rather than work in silos. Just as “DevOps” broke barriers between dev and ops, extend to DevSecOps.
- **Training & Awareness:** Invest in training engineers on secure coding, threat modeling, and reliability engineering practices. Empower teams with knowledge to make the right decisions
- **Empowerment:** Give teams the authority to halt a launch if reliability or security isn’t up to par (and back them up when they do). Make it safe for engineers to speak up on risks.
- **Incentives:** Include reliability and security metrics in performance evaluations or team KPIs. If people are only rewarded for feature delivery, guess what they’ll focus on? Instead, recognize those who strengthen the foundation (e.g., rewarded for reducing incident counts or closing security gaps).
- **Blameless Learning:** When incidents do occur, handle them with a blameless post-mortem process. Focus on learning and improving systems, not punishing individuals. This encourages openness about problems – which is crucial for identifying and fixing root causes in complex systems.

Architecture & Design Best Practices

- **Early Consideration:** Incorporate reliability and security criteria in design documents and architecture decisions. For every architecture proposal, include sections like: “How does this design ensure availability? How are we securing data and interfaces?
- **Principles to Apply:** Use proven design principles – *Least Privilege* (each component only accesses what it must), *Defense in Depth* (multiple layers of security controls), *Loose Coupling* (so a failure in one module doesn’t domino), *Redundancy* (no single points of failure).
- **ATAM Workshops:** Consider running Architecture Trade-off Analysis sessions for major systems. Ensure you discuss reliability and security trade-offs explicitly.
- **Reuse Frameworks & Standards:** Leverage industry standards and frameworks. Use security frameworks (like NIST, ISO27001) as a checklist for design. For reliability, reference cloud architecture blueprints or the AWS Well-Architected Reliability pillar which defines best practices (e.g. auto-scaling, backup and restore).
- **Threat Modeling in Design:** Make threat modeling a required step for new architectures. For example, when designing a new API service, model threats (abuse cases, injection points, data leakage paths) and adjust the design to mitigate them (e.g., adding authN/Z, input sanitization layers).

DevSecOps & Shift-Left in Practice

- **Integrated Pipelines:** Incorporate security and reliability checks into CI/CD pipelines. e.g., automated static code analysis (SAST) to catch security bugs, dependency vulnerability scans, and automated test suites for reliability (unit, integration, and stress tests) on every build.
- **Infrastructure as Code & Policy as Code:** Manage infrastructure via code (Terraform, CloudFormation) and encode security policies (e.g., linting for misconfigured cloud resources). This ensures environments are reproducible and config drift (a common source of reliability issues) is minimized.
- **Continuous Testing:** Do continuous integration not just of features but of failure scenarios. For instance, run nightly chaos tests in a staging environment, or simulate high-load scenarios regularly.
- **Feedback Loops:** Use telemetry from operations to feed back into development. If monitoring shows frequent near-misses or performance bottlenecks, feed that info to the backlog. Developers should treat ops data as essential as user stories.
- **Tooling:** Provide easy-to-use tools for developers to do the right thing. For example, provide libraries that handle encryption correctly so devs don’t roll their own. Automation and tools make doing the secure/reliable thing the path of least resistance.

Common Pitfalls to Avoid

- **Treating It as One-Time Project:** Don’t assume a single “hardening sprint” or one reliability initiative means you’re done. Security and reliability require ongoing effort.
- **Overlooking Human Factors:** Many failures come from process or people (misconfigurations, poor handoffs) as much as tech. Not investing in training or not addressing burnout (tired engineers make mistakes) can undermine technical safeguards. Maintain processes that account for human error (peer reviews, automated checks to catch mistakes).
- **All or Nothing Approach:** While these are foundational, avoid going to the other extreme of *“gold-plating”* beyond necessity. Use risk-based prioritization. Not every system needs five 9s of uptime or military-grade security – define levels appropriate to business value.
- **Siloing Reliability and Security Teams:** If you have dedicated teams (SRE team, Security team), great – but don’t let others off the hook. A silo mindset like “the security team will handle it” or “SRE will fix our outages” breeds lack of ownership in feature teams.
- **Ignoring Early Warning Signs:** Small outages or minor security alerts often precede major incidents. Don’t ignore them. If a service has frequent minor downtime, treat it as a big red flag and investigate before a massive outage occurs.

Future Trends & Considerations

- **AI and Automation:** As systems incorporate AI (e.g., AI ops, automated threat detection), expect both improvements and new challenges. AI can enhance monitoring and predict failures, but also introduces new risks (e.g., ML models that need security and reliability evaluations themselves).
- **Zero Trust Architectures:** The industry is moving toward “zero trust” security models (treat no network segment as trusted by default). This will become foundational for security in distributed systems – designing authentication and authorization deeply into every interaction
- **Infrastructure Evolution:** Serverless and containerization abstract infrastructure, but reliability and security still need attention (cold starts, multi-tenant security issues, etc.). New architecture paradigms will require rethinking how we ensure reliability and security – e.g., chaos engineering might shift to chaos experiments at the application level since the infra is abstracted.
- **Regulatory Pressure:** Expect more regulations mandating “secure by design” and resilience, especially in critical sectors. Governments are recognizing cyber resilience as national security.
- **Cyber-Physical Convergence:** With IoT and cyber-physical systems, reliability and security have real-world safety implications (think autonomous vehicles or smart grids). The bar for “foundational” in such systems is even higher – a crash or breach can affect human lives

07

Key takeaways

And next steps

Key takeways

- **Foundational, Not Optional:** Modern IT systems must treat reliability and security as *built-in qualities*, as essential as the features themselves. In an “always on” digital world, nothing less is acceptable to users or stakeholders.
- **Holistic Approach:** We saw that a mix of sound architecture, early risk identification (risk modeling, ATAM, threat modeling), and cultural practices yields robust systems. It’s not one silver bullet but a comprehensive strategy and mindset.
- **Learning from the Best:** Industry leaders (Google, Netflix, AWS, etc.) have demonstrated that prioritizing these areas pays off in scalability, trust, and agility. Emulating their principles in a way that fits your context can drastically improve outcomes.
- **Action & Continuous Improvement:** Making reliability and security foundational is an ongoing journey. It requires initial commitment and continuous adaptation. But the payoff is huge: fewer emergencies, more user trust, and freedom to innovate safely.
- **Final Thought:** By proactively building secure and reliable systems, we aren’t just preventing negatives; we are enabling the business to move faster and more confidently. It’s an investment in long-term excellence and resilience.

Next Steps – From Theory to Practice

- **Assess Current State:** Take stock of your organization's reliability and security posture. Where are the biggest gaps? Perform an audit or maturity assessment against best practices discussed (e.g., do we threat model? do we have SLOs?).
- **Quick Wins:** Identify 2-3 immediate actions. For example, *establish a regular threat modeling meeting* for new features, or *set up a basic chaos test in staging next sprint*. Early visible improvements build momentum.
- **Set Targets:** Define reliability SLOs and security objectives for critical systems if not already in place. Communicate these targets to all stakeholders and make them a part of project definitions moving forward.
- **Invest in Training/Tools:** Allocate budget for team training and for tooling (monitoring systems, security testing tools). Equip your people for success.
- **Plan & Iterate:** Develop a roadmap for the next 12-18 months for larger changes – e.g., implementing an SRE program, adopting DevSecOps pipeline fully, refactoring a risky legacy module for reliability. Break it into phases and iterate. Revisit progress each quarter.

ТЕХНОЛОГИИ В ТВОИХ РУКАХ

phd X pt

@BOOK_CUBE

@apolomodov

@book_cube

QR for voting