SKILUP SM DAYS

by: DevOps Institute
ADVANCING THE HUMANS OF DEVOPS

SRE Patterns & Anti-Patterns

Name: Shivagami Gugan
DevOps Institute Ambassador

Agenda

Technology Transformation Leader, Aviation Technologist, CDTO delivering Digital initiatives from the Middle East, Head of Engineering, Site Reliability Engineering & Cloud

Shivagami Gugan

SRE Patterns and Anti-Patterns

Hope you take away from this presentation:

- More clarity on SRE patterns
- Few traps to avoid along the journey

SKILUP DAYS

SRE is about making Machines work for Humans

Old World

a machine, called a pager, wakes you up at 3:00 in the morning because some other machine is having a hard time.
In this world, you work for the computers.

The SRE World

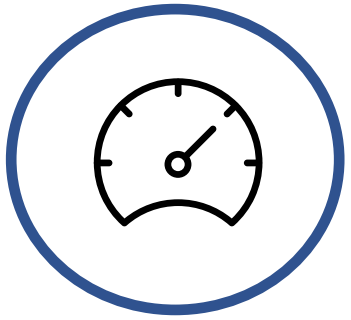
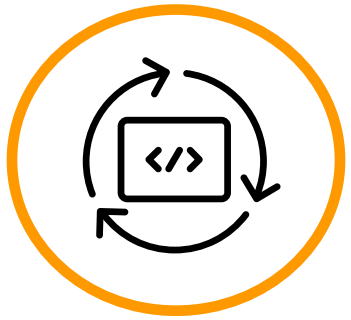
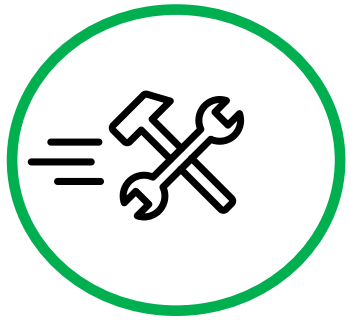
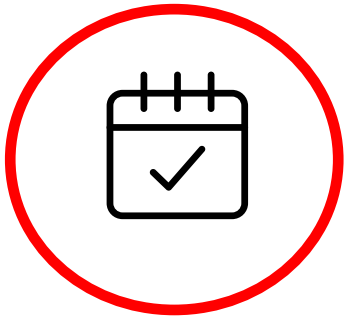
some system you're responsible for is having a problem, it mitigates itself, and then it writes a bunch of information out for you to debug the next morning.
This's a world where the machines work for you.

SRE is a world where the machines work for you. …

"That's the difference between staring at a monitor for alerts, versus trying to write software or implement systems that fix themselves."

Metrics for Success

Software delivery and operations performance correlate to business goals

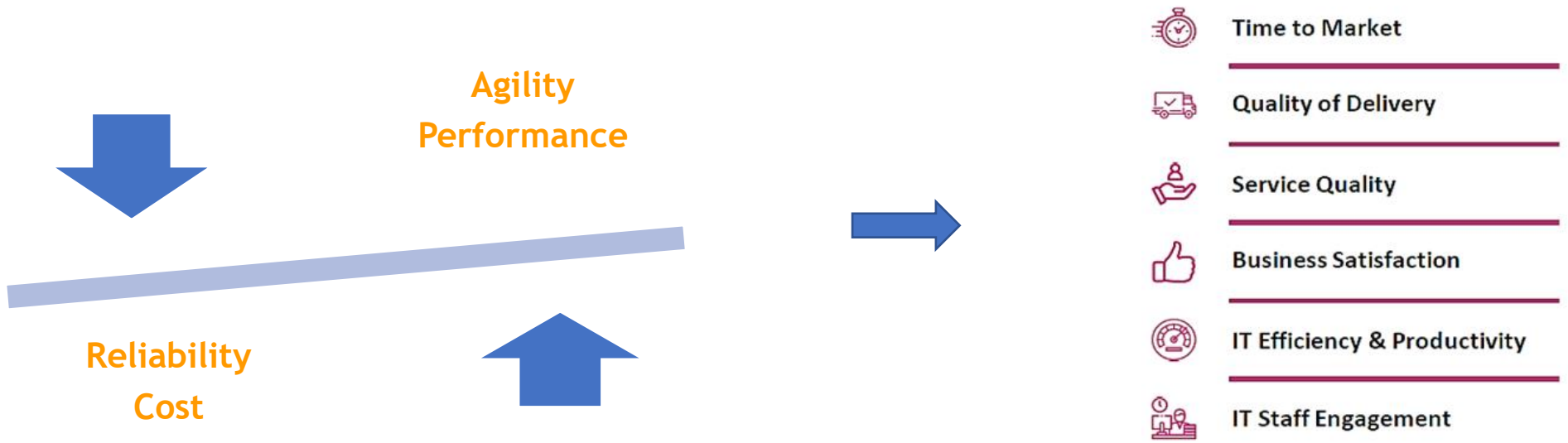
Deployment Frequency	Lead Time for Changes	Time to Restore Service	Change Failure Rate
			
On-demand ↑ Once every 1-6 months	< 1 day ↑ 1-6 months	< 1 hour ↑ 1-4 days	0-5% ↑ 46-60%

Deliver Business

SAFER. FASTER. CHEAPER. BETTER.

SRE Barriers

- **Reliability is everyone's problem**
- SRE is the **purest form of the implementation of DevOps**. SRE is about removing the silos in a Product lifecycle and building reliability across the spectrum (design, dev, deploy, operate, etc.)
- SRE's constantly live in the **Conflation** of Systems thinking and Software Engineering
- SREs constantly play the **balancing act**, and intelligently trade off options to create pragmatic and cost effective decisions across the entire Product life-cycle.

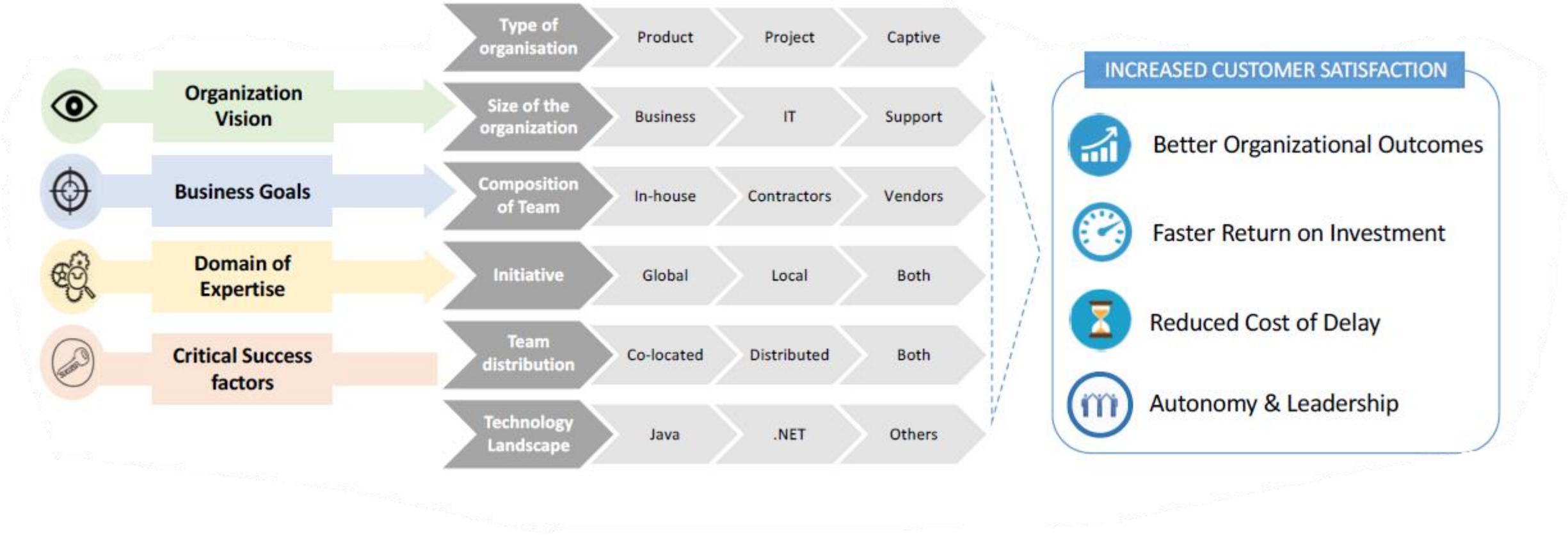


Remember SREs are huge Collaborators

- SREs bring **Singularity** between Dev and Ops, between Software and it's Ecosystem, Systems Engineering and Software Engineering
- An SRE is expected not to "run" **systems** but rather to create environments and automation that will self-enable systems to run
- They work on the entire continuum between Software Design, Development, Testing, Deployment, Operations and Customer Experience

Step 1: Understand Customer Expectations

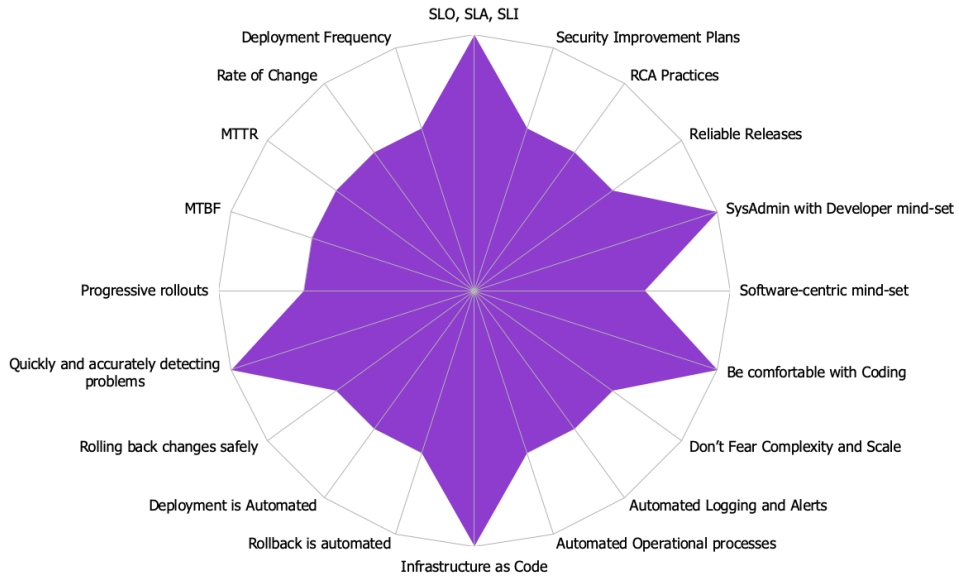
Understand Customer Expectations → Assess Org context and Maturity → Create Solution Strategy and Roadmap → Transform, Inspect & Adapt → Sustain Continuous improvement



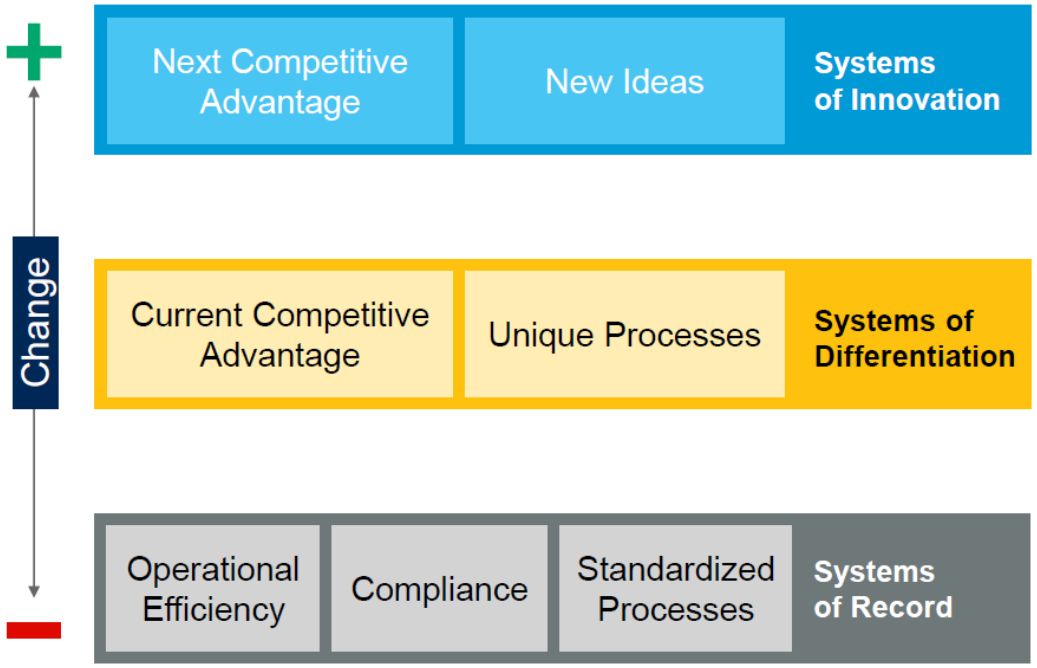
Step 2: Don’t Boil the Ocean

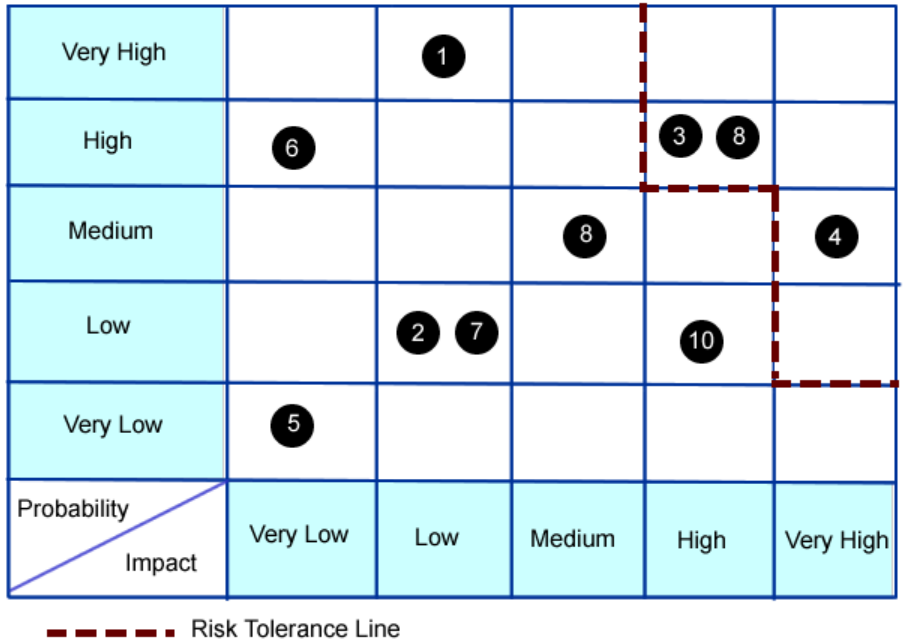
- Start small in obvious areas and scale out….
- Broad benchmarking could help …. Automate the measurements and aim to eliminate toil

	Continuous Integration	Automated Deployment	Test Automation	Automated Provisioning	Architecture	Agile
Level 5 FLYING	Automated feature driven delivery	Deployment automated for advanced “feature go live” scenarios	Teams have adopted business driven, feature based advanced test capabilities	PaaS is innovation accerlator	Teams are free to accelerate and innovate without Constraints	Team are self-organized, self managed, Committed work is getting delivered
Level 4 RUNNING	Pipeline enabled for delivery validation	End-to-end AD, including most infrastructure components	Teams apply test automation for non functional requirements	PaaS with on –demand environments	Support feature driven delivery and evaluation	Retrospectives are effective. Refinement sessions are adding value
Level 3 WALKING	Single build for all environments	Standardized deployments for all environments	Test Driven Development / Behavior Driven Development	Systems are delivered within one day, using self-service tools	Supports autonomous delivery	DoR / DoD are respected. Metrics are collected
Level 2 STANDING	End-to-end build and packaging from CI server	AD for application binaries , configuration and data	Automated feature tests and stakeholder demo’s	Systems are as similar as possible	Feature delivery in small iterations	Agile ceremonies are performed
Level 1 CRAWLING	Central build server and version control system	AD for application binaries (code)	Static code analysis and automated unit tests	Basic provisioning	CD compliant application architecture	Basic trainings on agile done, but projects still in waterfall mode
Level 0 Not Started	Limited central CI capabilities	No deployment automation	No test automation	No provisioning, (partially) manual process	Technical issues prevent increasing Release Frequency	No knowledge of Agile practices



Step 3a: Easier ways to Select Target areas

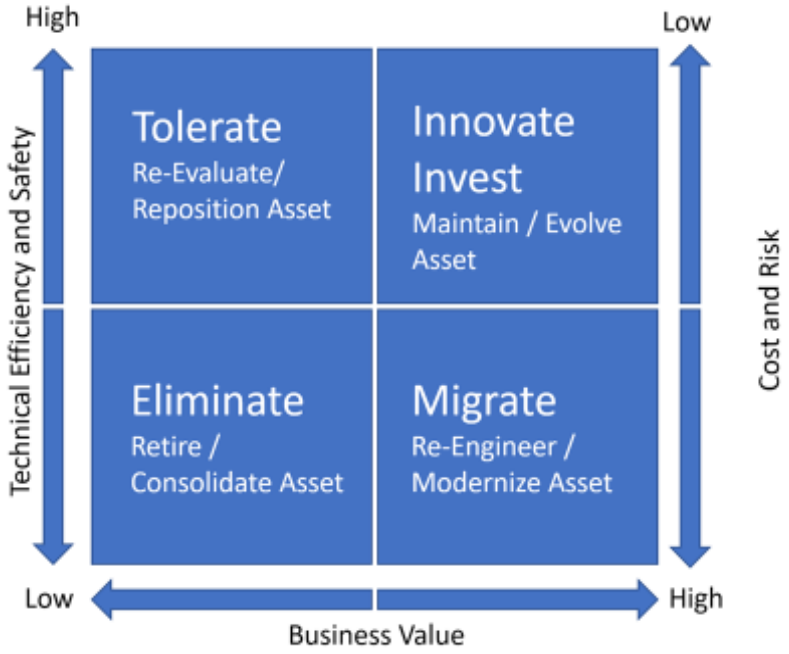


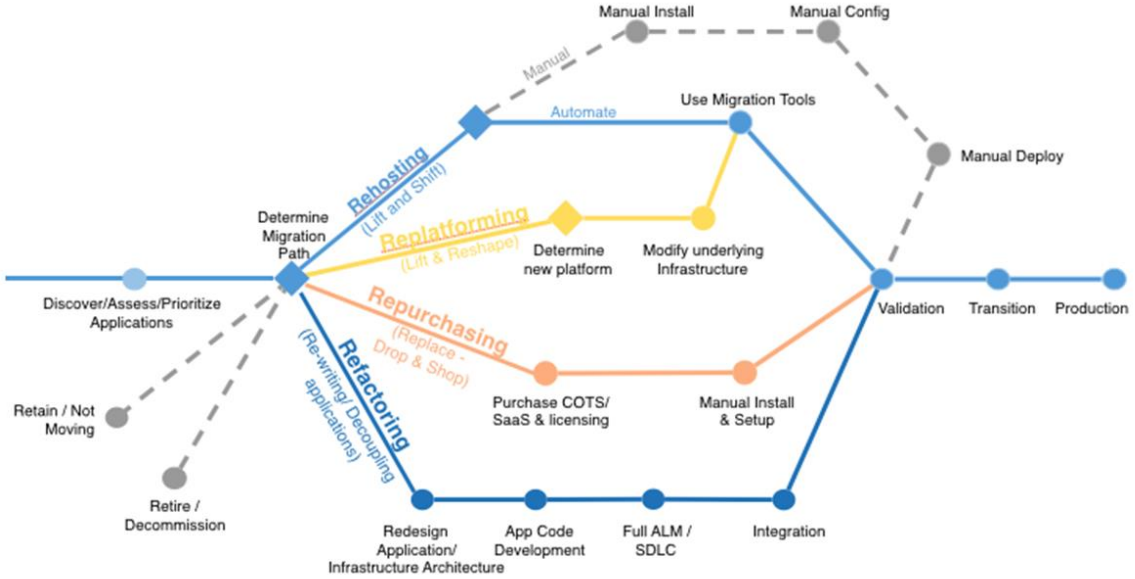


Probability and Impact of Service Failures

Source: Gartner Research Report
https://www.gartner.com/binaries/content/assets/events/keywords/applications/apn30/pace-layered-applications-research-report.pdf

Step 3b: Easier ways to Select Target areas

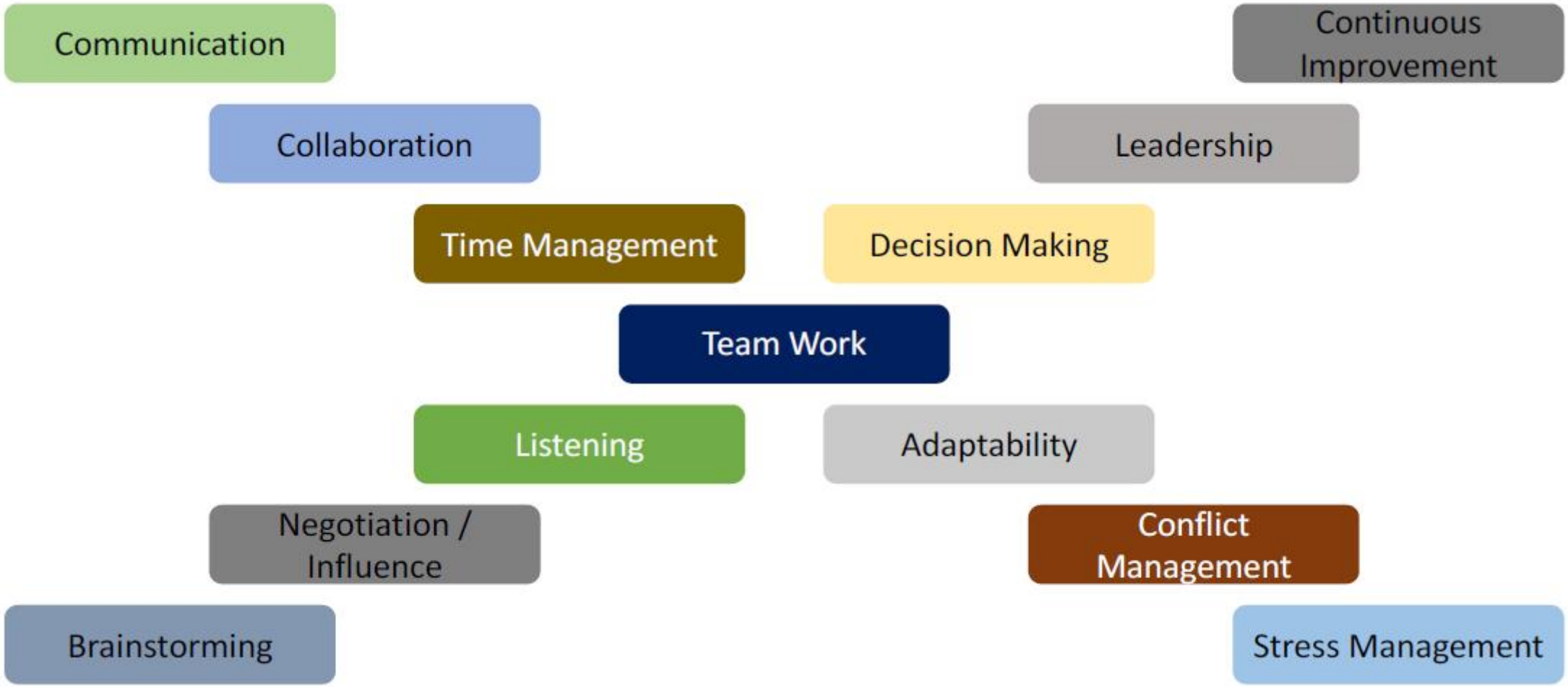




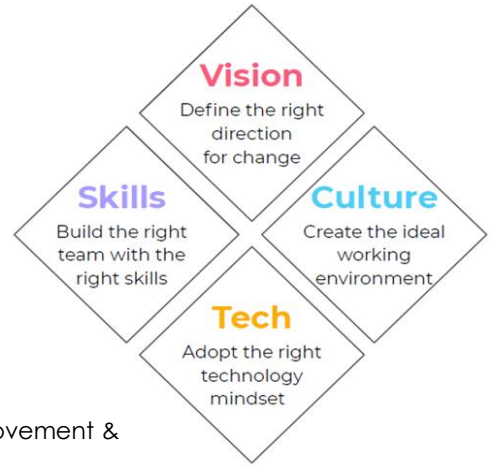
- Is Business Innovation is a Priority
- Is Technical Debt is a Priority
- Is Cloud Migration is a Priority

Cloud Migration, DevOps CI/CD Automation, Observability, Automating Ops, Dev, Deploymet are all intertwined Strategies
Selection of Target areas can be based on a combination of multitude of factors

Behavioral Skills and Culture Assessment is Super Important



Implementation Roadmap



TARGET AREAS

ESTABLISH

1 Blueprint for SRE

- Assess Business expectations, Align with Org goals & establish current state
- Align strategic goals to actionable initiatives
- Identify quick wins & create an actionable plan

2 Assess/ Maturity/ Roadmap

- Determine existing technical and software landscape
- Identify bottlenecks in flow of value and create optimisation roadmap
- Create reference architectures and implementation plan for SRE
- Metrics defined and prioritized

OPERATIONALIZE

3 Best Practices

- Enterprise reference pipeline is defined
- SRE Practices defined
- Best practices documented & socialized

4 Enablement

- Training strategy defined
- Training paths initiated
- Knowledge repository/ scaffolding consulted
- Hands-on coaching to Teams

5 Governance

- Enterprise performance dashboards
- Stage gate governance through pipelines
- Metrics automation

6 Evangelism

- Cross – pollinate learnings
- Adopt best practices
- Metrics baselined, and measured

MATURE

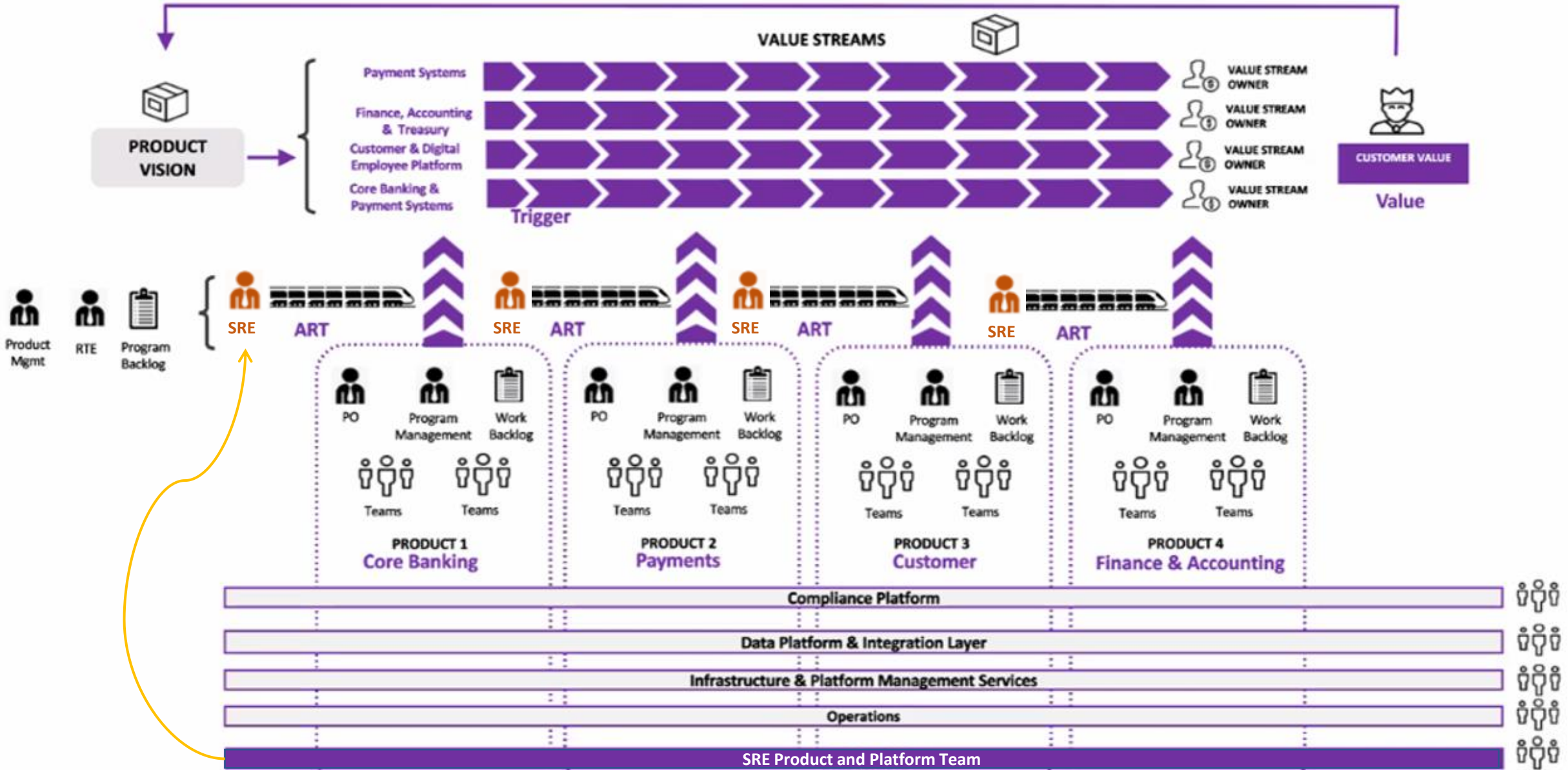
7

- Refine & Review
- Continuous Improvement & Streamlining
- Evolve best practices & plan
- Steering
- Metrics Reviews

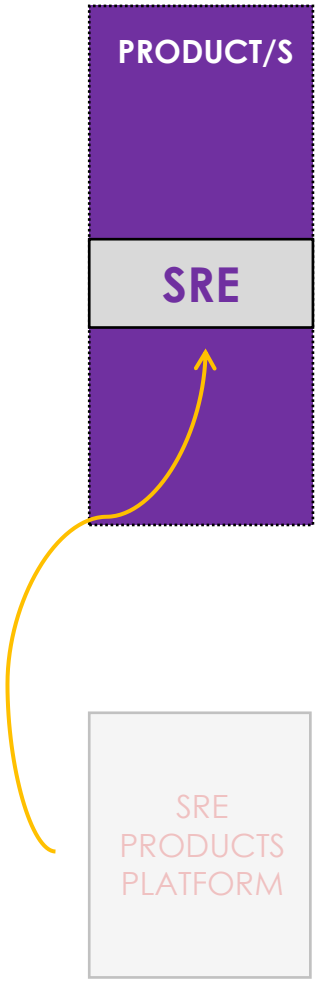
8 Rapid SRE Transformation

Continuous Improvement

SRE Possible Implementation Pattern

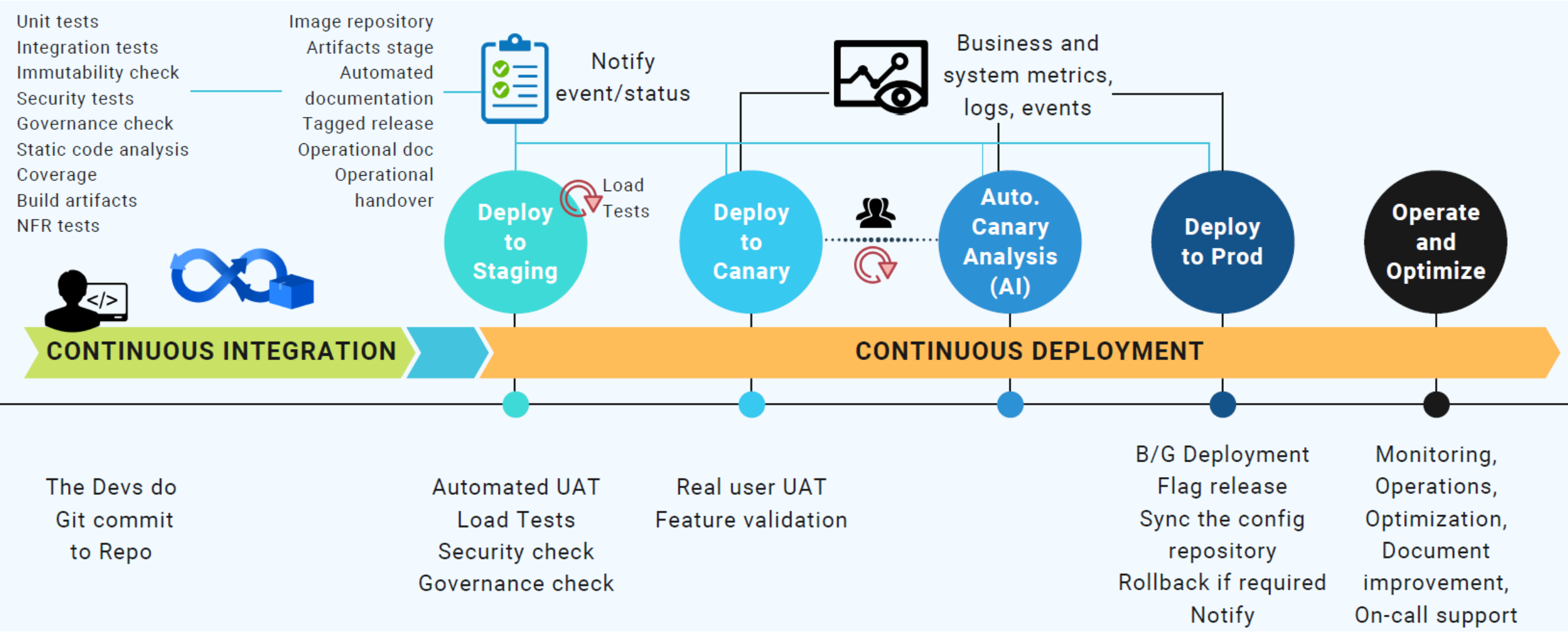


SRE Possible Implementation Pattern



- SREs are **embedded within the** product domains (Release Trains), and will develop deep knowledge **of the Product and it's ecosystem.**
- SREs drive **Automation (illustrative but not exhaustive examples: CI/CD flows, infra as code, operations automation, self-healing, cloud migration to improve reliability across the value chain)**
- SREs **collaborate** with other engineers, product owners, and business to drive automation and remove toil, **aligning with stakeholder roadmaps and priorities.**
- SREs coach developers and product owners on **SRE and DevOps practices** to enable then deliver business and IT objectives.
- SREs **consume and contribute** to the central Platform Products, to ensure the application service **is aligned with the company** development and operational **guidelines.**
- SREs collaborate with other departments to ensure high **reliability** of business services, ensuring early discovery of problems to **reduce the cost of failure**
- SREs **implement application monitoring and observability**, consuming the central platform product, to ensure end to end telemetry
- SREs will participate with Product folks to define **SLAs** for business services
- SREs will translate SLAs to SLOs and **SLIs** and will ensure observability to measure SLI
- SREs will collaborate on **Triage, RCA and blameless post-mortem** towards incident resolution.
- SREs will ensure adequate **capacity planning, optimisation**

Build Stage participation



SREs enable Developers to consume Enterprise CI/CD pipelines, Observability and ensure full visibility of DevOps pipeline and activities such as application deployments, continuous delivery, testing and correlated with data from tools such as Jenkins, JIRA, Confluence and GitHub etc. to optimize Engineering and Operational Processes. They will strive to remove Toil across Product lifecycle

Build Stage Participation from Observability

Measure and optimize DevOps performance to maximize ROI

- Optimize software delivery with goal-based KPIs
- Support decisions with data-driven recommendations
- Identify trends and predict release risk

Examples

Code Coverage from Sonar

Unit Tests passed from Sonar

Code cyclo-complexity and vulnerabilities from Sonar and Shiftleft.io/ Fortify

Number of Functional Tests passed

Stress/Load Tests passed

Build Success Average

Average time of Commit

Number for Commits Rolled back

Run Stage Observability

- Server
- Infrastructure
- Application Ecosystem
- Application
- Business Parameters
- Security Analytics

SRE Execution

- SRE are coders. They know the toolset of the Product thoroughly.
- If coming from the Dev side, they are programmers who understand infrastructure, can shell script and write interpreter code with ease. If coming from Ops side, they are the people who understand application design and development.
- They ensure SLOs are set at correct boundaries of service, they define alerts to detect SLI thresholds
- They measure and report performance against the SLI –Availability (Up time, Error Ratio –5xx/Total Requests) Performance (RPS, Latency)
- Their Operation load is capped at ~50 percent
- They enable developers on CI/CD automation, quality thresholds and deployment automation using infrastructure as code
- They enable developers to understand how their applications are performing in production building observability, using distributed tracing and APM tools
- They thoroughly understand deployment, fail-safe strategies - Rollback, Canary and Feature Flags.
- They influence in building fault-tolerant, autoscaling, cost efficient, highly performing design and architecture.
- SRE should ensure consumption of platform standards, should raise pull requests to enhance SRE Product/ Tool chain features.
- SREs ensure consistency of tooling - All lower environments use consistent methodology and same tooling as used in higher environments.
- SREs handle on-call events and do post mortems (For e.g. They are adept with Memory dump analysis, Thread dump analysis, OS level diagnostics, Functional diagnostics)
- SRE ensures error budgets are followed, they ensure self-regulation of velocity and stability and ensure excess Ops work overflows to the Dev team

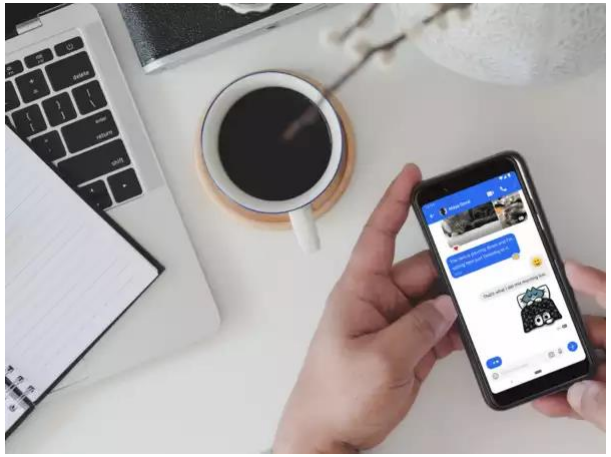
#1 Rebranding Ops

- SRE is not about keeping the systems that have already been built running at all costs.
- SREs build systems to require less human intervention and to fail less often, and they modify existing systems to remove emergent failure modes. They do engineering work.
- Don’t bring Ops closer to machines, the key is to build right alerts and have a distributed sharing and collaboration from anywhere, so that just the engineers who should actually be on call is alerted.

SREs spend more than half their time building better systems, rather than conducting or documenting operational tasks

#2 Users notice an issue before you do

- The basic tenet is to turn your SLOs into actionable alerts that are grounded on the customer's path
- Use key attributes *(Precision, Recall, Detection time, Reset time)* to build alerting strategy
- Build solid fail-over, fault tolerance to resolve within SLO



SREs are a direct contributor to getting the customer what they want: reliable, performant access to the services that make their lives better

#3 Measuring until My Edge

SLO is what is **perceived** by your customer/user, not what your SLOs read

a. Interdependencies and integration with outside partners overall matter

b. Shared institutions, Shared integrations, Shared communications key to success

Meeting your SLO is meaningless if customer does not have the experience

#4 False Positives are worse than No Alerts

- Do most of us operate this way? NOC escalate outages to the SRE, who in turn calls your Dev and Deployment Teams?
- Individual host alerts and False Positives are worse than No alerts
- Response Fatigue and Information overload of timeseries data is no good
- Alerts should have GREAT diagnostic information

Monitoring is about ensuring the steady flow of traffic, not a steady flow of alerts

#5 Configuration Management trap

- Traditional Infra (Snowflakes) are inefficient from a operational management standpoint
- Even with a good configuration management system, with 100s of services, a disaster is waiting to happen
- SREs spend less time on changes and more on homogenizing ecosystems
- Design for Immutability infrastructure - Pet (*snowflakes*) vs. Cattle (*VMs*) vs. Poultry (*containers*)
- Change is all about replacing, never updating or patching

Use Configuration Management to consolidate and migrate to Immutable Infrastructure

#6 The Dogpile: Mob incident response

- All hands on deck without an incident command framework is disruptive to engineering work, increases time to analyze and resolve incidents

- A good procedural framework for handling such situations is a mandate, that an SRE can't magically substitute

Minimize Damage. Make Outage as short as possible

#7 Point Fixing

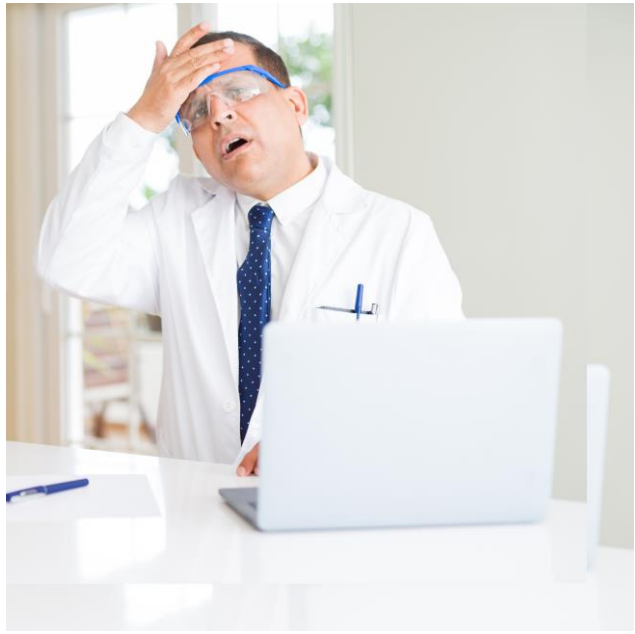
- Minimize Outage with automated alerts and solid paging mechanisms and quick workarounds, Fast rollback, Fail over and Fix Forward
- However, analyze and **eliminate Class of Design Errors**
- Short term fixes followed by Preventive long term fixes leading to Predictive methods
- Auto Remediation and closed loop remediations without human interventions should be the aim of SREs

Kaoru Ishikawa

Reactive to Predictive to Auto Remediation

#8 Human Error

- SREs strive not to have cause of an outage repeated. The desire to prevent such recurrent failures is a very powerful incentive to identify causes.
- The “root cause” is just the place where we decide we know enough to stop analyzing and trying to learn more.
- Instead think of **contributing factors**, if we know what happened and where things “went wrong,” , lets explore the system as a whole and all the events and conditions leading up to the outage.
- **Its never a human error, it’s a system problem - *Jennifer Petoff*, *Google***



If a well-intentioned human can “break” it, it was already broken

THANK YOU!

Meet Me in the Network Chat Lounge for Questions

Shivagami Gugan

SKILUP DAYS