**SEGR 5910 Final Exam – Ian Joslin**

Please answer the following 10 questions and return to me, in Word or PDF format, by 22 March at 8:00pm. Late exams will not be accepted. I will send an email when I receive your exam, if you do not hear back from me, assume I did not get the exam.

There's an 11th question that can be used as a bonus. The overall exam is worth 30% of your quarterly grade, so each question is worth 3% of your overall grade. Unlike the mid-term, the bonus question on this exam can be used to bump your overall quarter grade. That is, you can score 33 points toward your grade.

This is an open book exam. You may use any notes that I provided, any presentations, any recommended or assigned reading, and any references you find on the web. However, if you reference material from the web to answer a question you must provide a full reference (author, date accessed, and URL). I will be verifying references.

1. How are machines, virtual machines, and containers related. When and why would a team choose one over the other?

Machines are bare-metal computing hardware consisting of a CPU, motherboard, and memory. Virtual machines (VMs) are a representation of hardware. Multiple may run on the same machine and are allotted resources like CPU, memory, and networks by a Hypervisor (VM manager). Containers are like VMs but use less memory because they require a host operating system (OS). They may be deployed on either a VM or a machine. All may be used to run software applications, isolate address space, and isolate files and networks to some degree. Cost is a big factor in determining what a team may choose. Machines are most costly to maintain, while giving a team the most control over their computing capabilities. Containers are the least costly and lighter-weight (easier to deploy, reuse, and require les memory than VMs). VMs cost more memory space than a container, and they also share CPU/memory with other VMs, but able to run multiple OSs like a machine. Complete control of all resources may be useful for a team providing Infrastructure as a Service (IaaS), or where networking and security are of utmost performance, and so bare-metal could be the way to go. VMs can be called up easily and are less costly depending on the amount of computing resources needed. They may be preferred for control over networking and file structure and to build unique application environments, perhaps deploying software as a service or building platforms for the cloud. Containers are useful for teams looking to quickly spin up multiple applications with the same runtime settings across multiple environments. This is especially useful for building and running tests on a single application. In reality, a team will likely use some combination of the above artifacts in a way that best achieves both cost and resource efficiency, as well as their business goals. [1]

2. How are key performance indicators created for a product, service, or team?

Key performance indicators or KPIs are a type of performance measurement for evaluation of a team or product that are *measurable*. They differ from Service Level Indicators (SLIs) only in that they are considered the most important of the SLIs, which is where the work "key" comes in. They are created with a few simple steps. Envision the ideal: Here a team or product own must ask, what is the best possible performance or service we can provide? Again, this must be in terms of measurable units. For example, "our product is a Google Maps web API and should meet 99.9% of all geolocation requests with a 95% confidence level". Quantify Distance to the Ideal: "We are currently reaching about 96.7%. Our distance to make up is 3.2%". Imagine how behavior will change: The team or product owner must envision what needs to be done to make up that distance. Where should more dev time be spent? Are there new components needed to supplement our current code base? What other testing, if any, must be completed to ensure this goal? These must be actionable changes that are perceived to affect the performance in a positive way. Revise and Select: This step is an evaluation of the goal and the actions above, used to determine if the goal is achievable. This step creates an iterative process between the first three steps, ensuring that the KPI is useful, measurable, and attainable. For example, "We only were able to increase our geolocation request rate from 95% to 96.7% over the last quarter, but we feel the groundwork we have done will allow us to set a higher goal for change this quarter. Lastly, deploy the KPI: Put the KPI(s) in view of all stakeholders so that everyone agrees and can be held accountable for meeting that goal. It is important at this point to be able to measure the actual progress versus the KPI. The Google method of setting their KPI equivalent, OKRs (Objectives and Key Results), is to set a goal that they can meet about 70% of. They want to meet 100% of that goal, but if all are met at 100% you either did not set enough OKRs or have made them too easy. In the above example the team should be happy with an achieved geolocation request percentage or 99.1%. They have not met the OKR/KPI, but have greatly improved upon last quarter's performance and have results they can learn from and take into setting next quarter's goals. [2]

3. What are the common SRE engagement models and how do they relate to a production readiness review?

Common SRE engagement models include a Simple Engagement, Early Engagement, and Frameworks & SRE Platform. A production readiness review (PRR) is the most typical initial step of SRE engagement and seeks to determine whether a service/product meets the standards of reliability, operability, and monitorability. This process may determine whether a given development team is ready for SRE engagement and may implement requirements on that dev team before engagement may begin. In a Simple Engagement PRR analyses a service that is already in place, so must not only determine if the service meets SRE standards but includes any improvements and refactoring by the SRE team as well as cross-team training: the dev team must teach SRE team about the software, and the SRE team must teach the dev team about tooling, on-call processes, event triage and escalation, and platforms. Once onboarding and training is done the dev team can undergo continued improvement using the learned practices. With early engagement the cost of building SRE infrastructure and the best product dev team is incurred up front, and both the dev work and SRE work are completed transparently and must have agreed upon goals. The PRR in this case is simply asking whether a dev team is willing to begin development with SRE before development has begun: does the outline product meet SRE standards? If yes, SREs are involved in the design, build and implementation phases, at launch, and following launch before they disengage. The last model does not involve areal PRR. It involves using SRE tools and frameworks that are already built and tested and using these to guide how development proceeds. Using SRE tools allows for uniform interfaces and production environments across an organization and allows for easier automation and a smarter system. If more teams use the frameworks it can also improve them by requiring change and adaptation to new applications.

4. What approach or approaches did Netflix use to manage container adoption across their development organization(s)?

Netflix went to a completely cloud-based computing model by moving all its' services to AWS. They deploy all their microservices on VMs in the cloud, but they utilize containers to create a more flexible and efficient building and testing of their applications, significantly reducing overhead in these steps of the development lifecycle. The concerns from the beginning were that container adoption would move to fast, and lead to reliability and scale issues or that it would move slow and not warrant wide-spread investment. Netflix's approach was to basically let teams decide. A small set of teams began to see the benefits of adopting containers which provided concrete evidence of how containers could provide measurable benefits and solve problems these teams were facing. Demonstrating the value of containers, the early adopters could then help build more generalized features for other teams, further easing the adoption of container technology across the organization. [3]

5. To what does the phrase "engaged with SRE or not engaged with SRE" refer?

It refers to an old engagement model in which teams had full SRE support or pretty much zero SRE support. The new model of shared responsibility goes away from this, by putting responsibility for platform infrastructure in the SREs hands, and responsibility of on-call support for software service issues in the developer's hands. [4]

6. We talked about *inhibiting* and *silencing* alerts. When and why would you choose one over the other?

Silencing an alert means that the system logs the alert, but nobody is notified about it. This would be used for minor issues that need to be tracked and possibly analyzed later for trends but are not page-worthy because they do not indicate any severe system failure that needs human attention. This is also useful for alerts that are duplicate or uninformative. Google SRE guidelines suggest a 1:1 alert/incident ratio. Inhibiting an alert means you neither log an event nor notify anyone about it, which would be done for events that do not require human intervention and do not need to be tracked, for example, when multiple alerts indicate a much larger issue that would create a separate alert; the individual alerts may be inhibited once the more severe alert has been sent. One may also inhibit alerts on an issue already being triaged. [4]

7. Why would an SRE team be in the business of providing frameworks to development teams?

Providing frameworks for dev teams comes with many benefits. For one, dev teams get the benefit of using built and tested tools before they begin building their product. For SREs, it is beneficial for dev teams to be using common implementations and processes because it can ensure certain quality criteria are met and mitigate any issues caused by dev creating their own tools or implementations. With SRE frameworks, more dev teams may be able to automate certain processes and using them the better they can be improved for continued use, a compounding benefit. This means dev teams can worry about a lot less operational overhead and more on application code, and thus faster development. Having this consistency also allows organization-wide goals and strategies to be built into the support infrastructure, and can minimize continued SRE engagement over time.

8. What is the primary difference between *black box* and *white box* monitoring? What is the relationship of *perspective* to these approaches?

Black box monitoring is done by systems that have no knowledge of the implementation or backbone of a service. For example, monitoring from an end-user standpoint by submitting dummy queries to the service through a web interface. White box monitoring performed by looking at metrics internal to the system, such as logs, http requests handlers, and the like. Black box is related to user perspective or external service, whereas white box is related to an internal perspective of a dev or another internal service, on metrics important to the service owners. [4]

9. According to Google (and other experts by the way) what are some of the common pitfalls of ineffective troubleshooting and how can they be addressed?

As outlined in the Google SRE book, common pitfalls of ineffective troubleshooting include (but are not limited to):
- looking at symptoms that are irrelevant or misunderstanding system metrics
- misunderstanding the system and thus ineffectively testing hypothesis about an issue
- introducing unlikely theories for what is wrong or reasoning that a previous occurrence has occurred again for the same reasons
- mistaking correlation for causation.

A better understanding of the system, either through learned expertise or well-written logs and alerts, can mitigate the first two. This helps the on-call person better assess what is wrong and how to change the system and its' environment. It is also important to know a simpler explanation is more likely than an improbable one, and just because problem B looks like problem A, does not make them equal or mean they are caused by the same thing. The logical fallacies in the third point may be mitigated by remembering these rules. Lastly, correlation does not equal causation, and as systems grow more complex the likelihood of correlation by coincidence also grows. [4]

10. What is the difference between CMM Levels 3 and 4?

The CMM level 3, Defined, differs from level 4, Managed, in that it only lays out regulated, organization-wide standards and guides or best practices, but does not seek to measure actual performance against those practices. Level 4 goes a step further in quantifying actual or needed performance improvement that is predictable, attainable, and aligns with the organizations goals. [5]

11. **(bonus)** At which point does SRE cross from acting as a support organization to operating as a development organization? Defend your answer.

I like the Google SRE book's description of site reliability engineering teams as the result of a software engineer building an Ops team. From this standpoint SRE is development from the beginning. At least in it's ideal case it should be. While SRE teams may use existing products and services to build tooling and frameworks, and thus provide some operational support to those dev teams, I think that the implementation of the tools and frameworks makes them a development organization. The Ops part comes in when they need to teach dev teams to use the tools they have designed and implemented.

Information provided for answers to the above questions was taken from lecture slides or personal lecture notes, except where the answer lists any references.

**References**


[1] Bass, L., Weber, I., Zhu, L. (2015). *DevOps: A Software Architect's Perspective*. Boston, Massachusetts: Addison Wesley.

[2] Klau, R. (2012, October 25). *How Google Sets Goals: OKRs*. Retrieved from https://library.gv.com/how-google-sets-goals-okrs-a1f69b0b72c7. accessed March 20, 2018.

[3] Leung, A., Spyker, A., and Bozarth, T. (2018). Approaching container adoption in an already cloud-native infrastructure. *Communications of the ACM, 61*(2), pp. 38-45. doi: 10.1145/3152529. accessed March 21, 2018.

[4] Edited by Beyer, B., Jones, C., Petoff, J., Murphy, N. (2016) *Site Reliability Engineering: How Google Runs Production Systems.* Retrieved from https://landing.google.com/sre/book/index.html. accessed March 21, 2018.

[5] CMMI Institute. (2018) *What Are CMMI Maturity Levels?.* Retrieved from http://cmmiinstitute.com/capability-maturity-model-integration. accessed March 22, 2018.