Understandability: the most important metric you're not tracking

The software development industry has seen exponential growth over the past few decades, bringing to life ideas and products that were once thought impossible. We have seen a major shift from large scale, tightly coupled, monolithic applications designed to run on-prem to loosely coupled cloud native architectures which allow organizations to build and release software at a faster pace. With these advances in technology and architecture, organizations are finding benefits in cost savings, agility, and superior customer experiences.

While these new advances are creating value for businesses and their customers, it has also caused the complexity of software systems to grow tremendously. Modern architectures tend to contain many moving pieces including microservices which may run in platforms like Kubernetes, serverless components, service meshes, load balancers, web

KEY TAKEAWAYS:

Developing complex applications using next gen deployment methods make it difficult to keep up and understand what's happening in your own code.

Understandability means that a system's information should be given such a manner that it can be easily comprehensible to whoever is receiving that information.

There are four key criteria to understandability: organized, complete, concise, and clear.

When developing software, you need simple, instant, secure access to data. If even one of those criteria are removed, then you won't be able to achieve understandability.

Debugging in production not only generates a lot of stress, but also uses a lot of resources and can cause much annoyance among customers. By gaining better production-level understandability, you are able to get to the root cause faster, thus saving the organization money, risk, stress and reputation.

Rookout helps you achieve understandability by enabling developers to retrieve data about their software or applications behavior in a click, without breaking anything. They can get as much data as they need to achieve full understanding, without having the strain of attempting to extract that much data puts on a dev team. Achieving this level of understandability empowers your devs and optimizes their velocity.

©2020 Rookout

servers, and the list goes on and on. As these software systems continue to grow and developers write more code to support new features required by the business, the understandability of these software systems decreases. In order to properly understand what these systems are doing while they are running in their native environments, having quick and efficient access to the data processed by these systems becomes critical.

In this paper, we will dive deeper into software understandability including why it's important, how it differs from observability, and why quick and simple access to application data across all tiers of the architecture can improve the understability of these systems for the developers that are building and supporting them.

> ***Understandability means that a developer creating an application should be able to easily find or receive data from that application in order to understand its behavior.***

What is Understandability?

Borrowed from the financial industry, understandability requires that a system's information should be given or presented in such a manner that it can be easily comprehended by whoever is receiving that information. Translating this to the world of software development, understandability means that a developer creating an application should be able to easily find or receive data from that application in order to understand its behavior.

How is a developer able to do this, however? There are four key criteria to software understandability. In order to be understandable, an application needs to be:

1. **Organized.** The developer working on the system should be able to easily locate cross-referenced information within the system. This can be done in a variety of ways, such as source code management tools, software documentation, and source code navigation controls like those provided in an IDE.

2. **Complete.** The system has to be presented by the use of a predefined set of sources, such as documentation and code comments, to cover all fundamental information. No important informational items should be left to the developer's imagination.

©2020 Rookout

3. Clear. Developers should make every effort possible to write simple code that is easy to understand. Aspects like code comments, syntax and formatting, among others, can make a significant difference (static analysis tools can help tremendously here). There is nothing worse than running into a bug in a poorly written and poorly documented piece of software which you are not the author of.

4. Concise. The developer reading the system source code shouldn't feel as if they are buried under an extreme amount of detail. Allow engineers to focus on the assignments they have by implementing principles such as abstraction and separation of concerns.

Data Access as a Means to Understandability

In the traditional world of software development, understandability was often achieved using "static" tools and methods that relied on a development team's expertise and discipline. Engineers would be trained to write code that is clear, concise and well organized. Development teams would prioritize having full, up-to-date documentation, and detailed code comments at the cost of meeting deadlines and fixing bugs.

The introduction of DevOps, CICD, Cloud, and SaaS deliveries allowed teams to reach understandability by observing the application as it runs. By collecting data, log lines, and metrics from a running application, one could learn about its expected behavior, map the connections between its components, and visualize the scale and timing of the application.

Reaching understandability in this way, you need to first obtain high quality data. When developing software, you need simple, instant, secure access to data. If even one of those criteria are removed, then you won't be able to achieve understandability.

Simplicity

Obtaining data from running applications is often long and complicated. It involves writing code, getting it approved, testing, sifting through log files, and more. The amount of time spent and the complexity of the process create both technological and company-level friction. Keeping your deployment and monitoring systems simple help reduce friction, and allows you to get the data you need instantly.

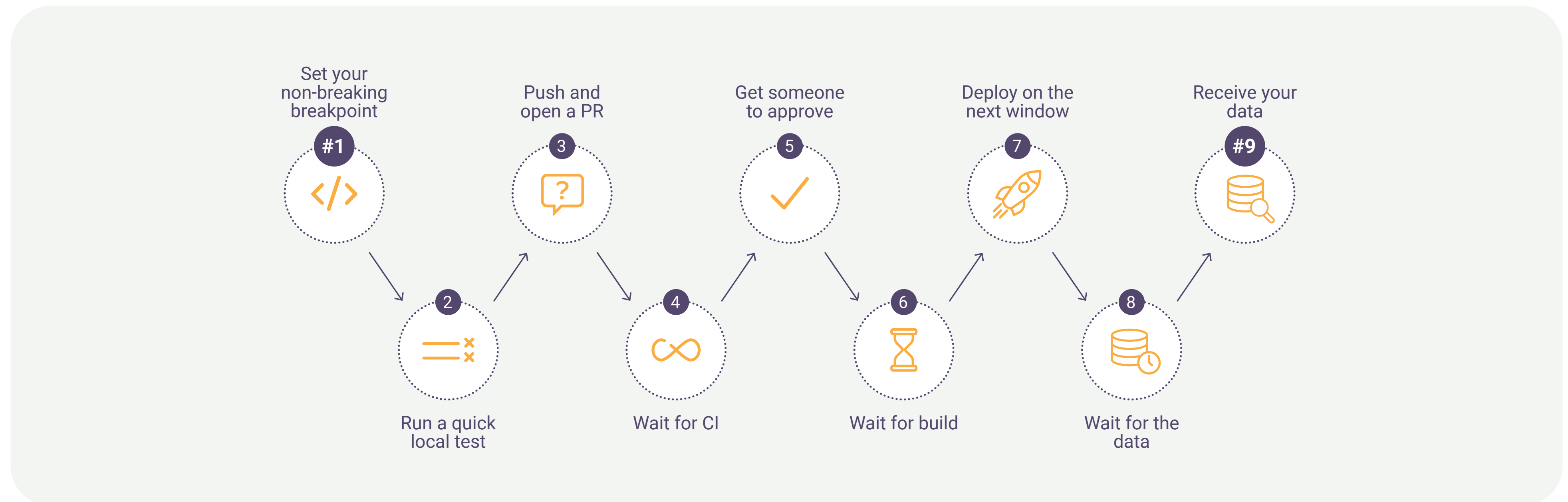
©2020 Rookout

Security

It's critical to consider that accessing your own code generates vulnerability - and no one wants that. Ensuring that your software is secure is of the utmost importance. All different standards and general regulations must be complied with in order to be able to create comprehensible software.

Speed

Quick access to data eradicates a developer's dilemma between needing data to write code, but needing to write code to obtain that data (the 'fly blind or fly slow' dilemma). With slow or little access to data, a developer has two choices: either work slowly, moving through endless deployment cycles to try and obtain data, or hope for the best and work without data, praying that their software will work as they hoped. Data is what provides them deep insight into their code, and without it, there is no understanding.



©2020 Rookout

Practical benefits of Understandability

When implemented and maintained well, understandability should help you meet the challenges of debugging, collaboration and handoffs, technical debt, logging, and monitoring.

Debugging

When debugging, understanding your code is the recipe for success. On the flip side, debugging code that you do not understand is a nearly impossible task. In order to successfully debug, it's important to understand the root cause of the bug, what affects it, and ultimately how to fix it. Without this understanding, debugging can take very long, effectively wasting important time and resources. Making things worse, attempting to fix a bug without full understanding of the application and its expected behavior, will often make things worse and introduce additional bugs in unexpected places.

Ultimately, all challenges encountered when debugging stem from the same source - **access to data that is simple, quick, and safe to capture.** Access to the correct data affords an understanding into your software, of which no debugging process could be successful without.

Technical debt

Technical dept is often manifested in knowledge gaps that limit a developer's ability to understand unmaintained code. Knowledge gaps stem from a disparity between what the developer should know and what they actually know. Quite often developers lack the data needed to have clear knowledge of what is happening in their code when it runs. Some developers choose to continue developing features even with this debt while others stop all feature development until it is settled, as the interdependency of the layers of code has the ability to affect new features. Overcoming this technical debt gap leads to understanding of both your code and your software.

Collaboration and handoffs

A common challenge in developing and maintaining software within a large team, or a team working remotely, is the need to hand off responsibility for code components. This has always been a part of software development - developers handing off to QA, Operations, Support and others.
But code that has been written by another developer can oftentimes be difficult to understand. Many developers have trouble working with other developers' code, or understanding legacy code for applications that they maintain.

All challenges encountered when debugging stem from the same source - access to data that is simple, quick, and safe to capture.

©2020 Rookout

The struggle to understand someone else's code sometimes becomes easier when developers can see the code as it's running. Thus the ability to dynamically get data from a running application makes the onboarding, collaboration and handoff flows easier and more maintainable.

Logging

When looking to understand their software, developers typically use logging to retrieve the data needed. Whether setting more log lines, or not setting enough, logging often doesn't get you the necessary data to understand what is happening. Setting more log lines can increase noise within the code, overhead costs, and wastes invaluable time. On the other hand, setting less log lines isn't the answer either. Gaining understandability doesn't lie within the logs you set, but rather the data you can extract from your code.

Monitoring

A complement to logging, monitoring tools use alerting, error tracking and measuring hardware performance to notify development and operation teams of potential quality issues. Knowing where in the application the problem originates will often point a developer at the impacted component, service, or even the specific line of code that has caused an issue. This is often the first step in trying to understand the source of a problem, or the real behavior of the application.

Observability isn't Understandability

When facing the challenges of software understandability, some organizations look for observability tools as a complement (and sometimes as a replacement) for building a culture that generates organized, complete, clear, and concise code repository. In cases where an engineer cannot understand the code from reading the documentation or code comments, looking at log lines, topology maps, or trace reports is the next best thing to learning how the application actually behaves.

When looking at observability tools, one can see that they were created for purposes that support more traditional IT problems. These are problems such as locating performance bottlenecks and maintaining comprehensive event logs for security or operations. Their capabilities are limited to being able to collect the necessary data developers and operations teams need to ensure smooth running software.

Observability tools were predominantly created for three main use cases:

1. **Maintain production performance.** The most basic illustration of this is to detect service disruptions as fast as possible, alerting the appropriate on-call staff, and ultimately aiding them in understanding the root cause of the issue.

2. **Minimize service disruptions.** The appropriate demonstration of this is when detecting performance bottlenecks and anomalies. These tools allow operations and engineering teams insight into the why and the where of where these are occurring.

©2020 Rookout

3. **Audit and log.** Here, it is meant in terms of providing long-term outputs of the system for consumption by customer-facing representatives, i.e. technical support, along with storing those logs for compliance and security purposes.

None of the above examples allow for data to be collected in order for the developer working on the software to gain understanding of its behavior. While it significantly aids in many other aspects of the software development process, it is simply the wrong tool to achieve understandability.

> *Achieving understandability is at the crux to achieving safe, fast, and efficient software.*

The End Goal

Achieving understandability is at the crux to achieving safe, fast, and efficient software. No matter what you are creating, or what choices you made to get there, being able to understand your software is an absolute must.

Rookout helps organizations achieve understandability by enabling developers to retrieve data from their running applications without affecting the performance of the application or requiring redeployment. Developers can retrieve data on demand in order to achieve full understanding of an application without having to know ahead of time what data they will need to resolve an issue. Achieving this level of understandability empowers developers and optimizes their velocity. Play around in the Rookout sandbox and gain immediate understandability today.

TRY FREE

©2020 Rookout