AppMap

Document and analyze legacy code with new reverse-engineering technology

Runtime code analysis is a critical missing piece in understanding the behavior of legacy systems.

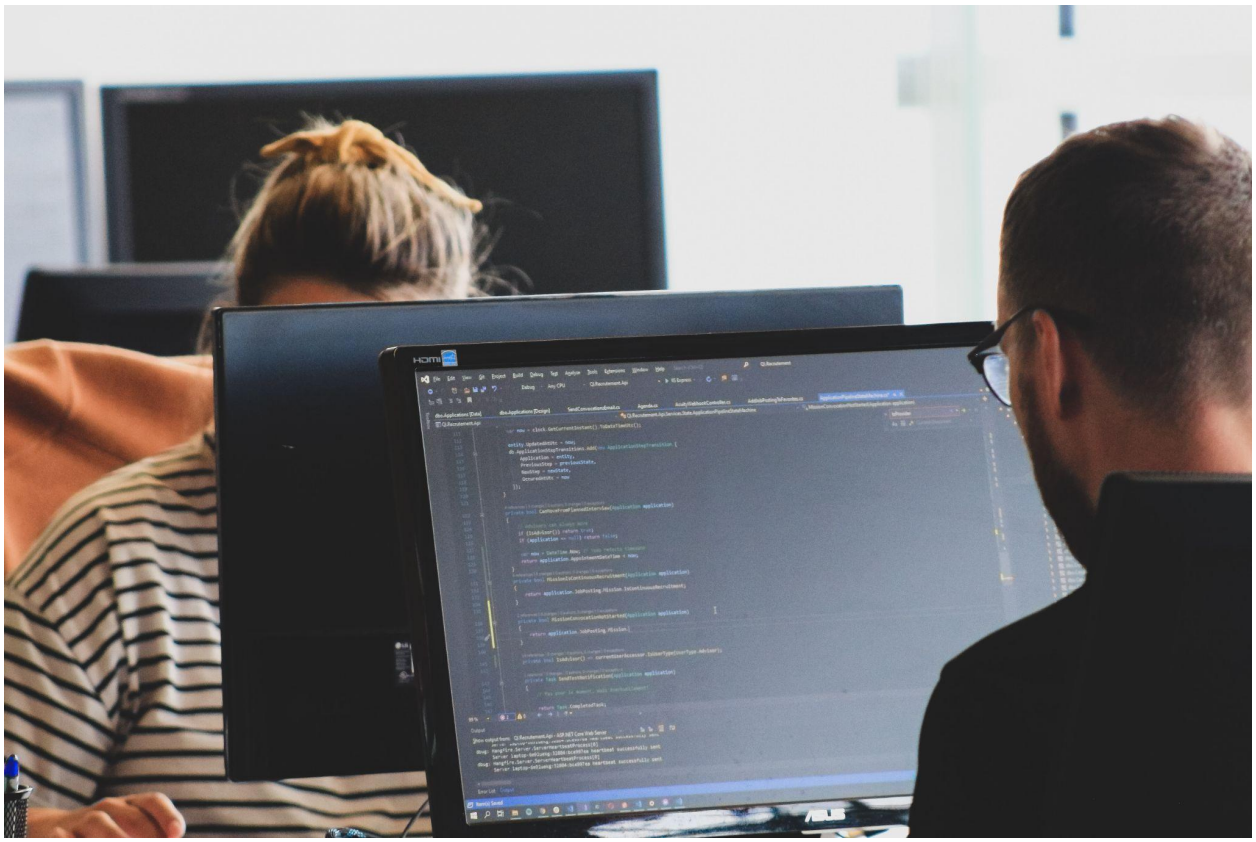


Photo by Sigmund on Unsplash

Modernizing a legacy application is a complex process that requires careful planning and execution. One of the critical challenges in modernizing a legacy application is the lack of documentation for codebases.

This prevents new project architects from identifying what code should be addressed first and understanding the entanglement of existing code behavior. This complexity of poorly documented and tested codebases brings an added risk to the application modernization team tasked with improving it.

Runtime code analysis is a critical missing piece in understanding the behavior of legacy systems.

Common approaches to modernization

The Strangler Pattern is a common approach to application modernization that involves gradually replacing parts of monolithic applications with microservices. As a first pass, developers identify areas where the application is tightly coupled and identify potential services that can be extracted into microservices. This process can reduce the risk of introducing new services into the system and ensure that the overall system remains cohesive.

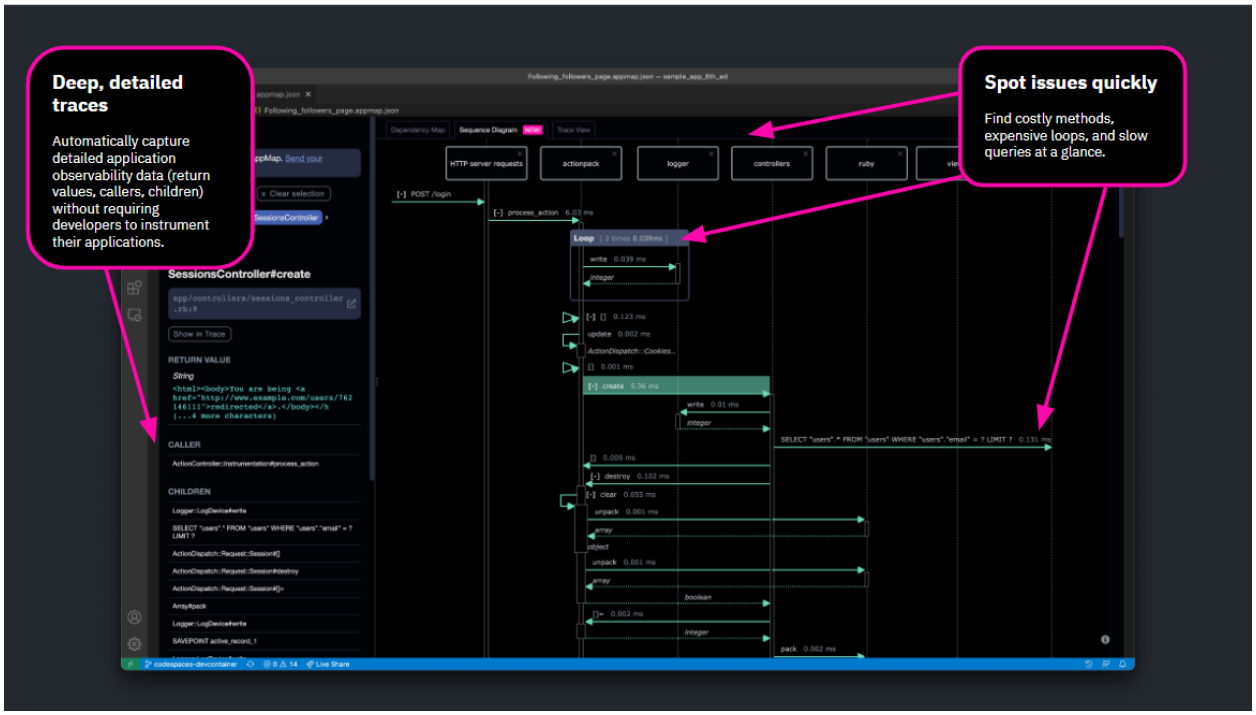
The Strangler Pattern can be difficult to initiate for developers and architects coming into a new codebase. It requires a deep understanding of the dependencies between components. Architects typically need to analyze the codebase and identify areas that are highly coupled and can be separated into distinct services, rather than coming from a point of institutional knowledge of the codebase. Onboarding and analyzing the codebase can be tedious and time-consuming and take months to complete. In large codebases, it involves reviewing dependencies between thousands of components and identifying areas where changes in one component could have a significant impact on other downstream components. The challenge for

the developers working in the Strangler Pattern is to ensure that the new microservices integrate seamlessly with the existing system and do not introduce new problems resulting in stability or performance issues.

Runtime code analysis and observability are critical analysis tools for software application modernization projects that are being shown to drastically reduce onboarding costs, improve delivery quality, and drive business success. As software applications grow in complexity, the importance of runtime code analysis during the software analysis and subsequent modernization process will continue to increase.

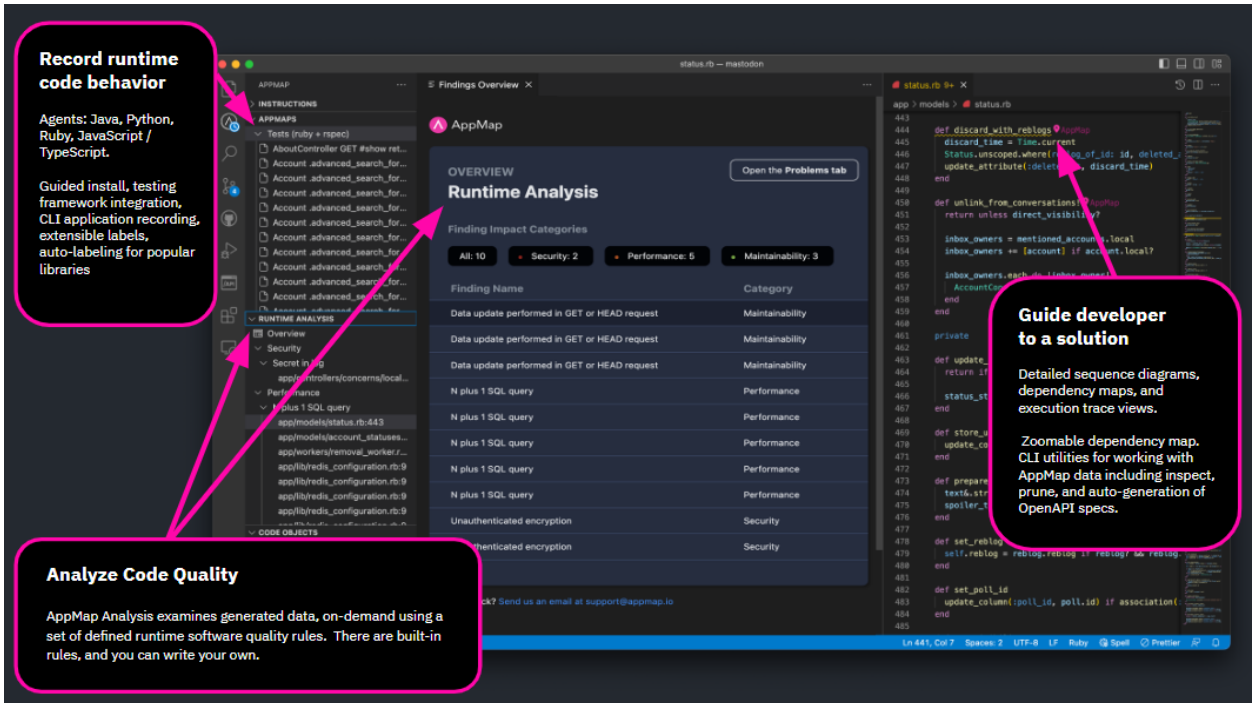
Runtime code analysis is key

Runtime code analysis tools, like AppMap, provide insights into the codebase before and during modernization projects. Runtime analysis automatically generates visual diagrams of the behavior of applications and generates missing internal API documentation, providing developers with detailed insights into the dependencies between different components.



Sequence diagram view of runtime code behavior

Runtime analysis also detects latent defects, unexpected dependencies, and security issues that may not be apparent during static analysis and initial onboarding. These require resolution for new code to be introduced successfully. By providing detailed insights into the application's behavior, developers can see if the code has adequately isolated the targeted business logic.



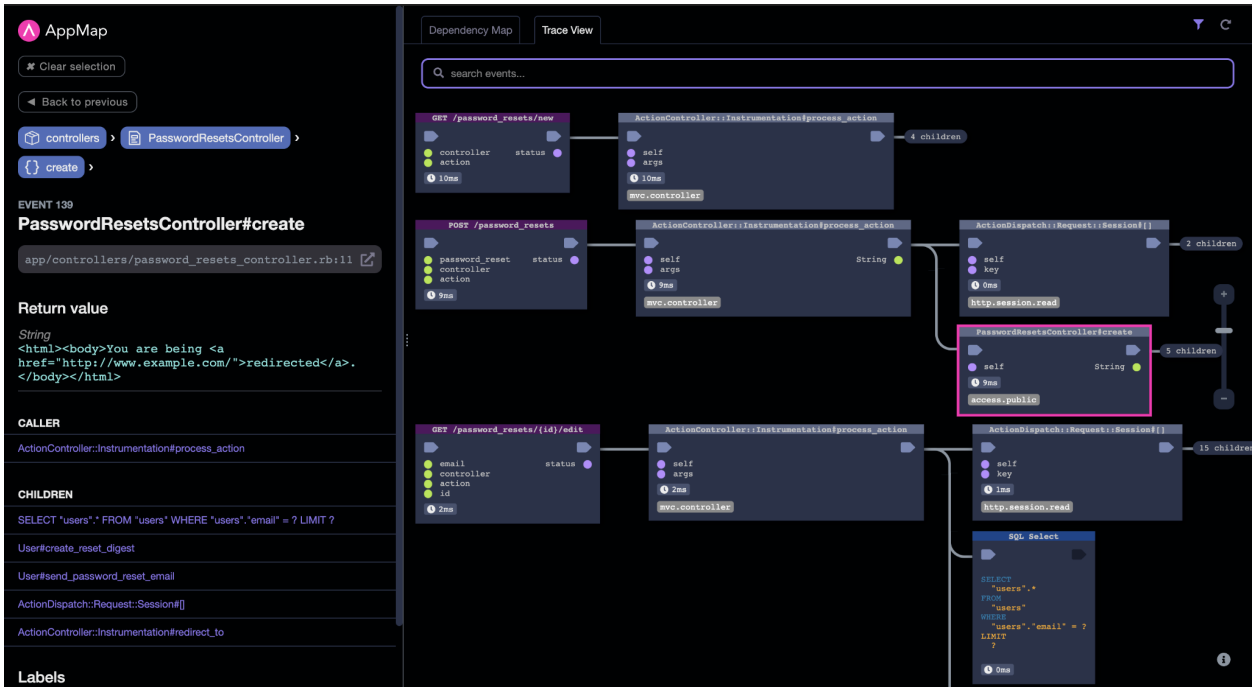
Runtime Analysis report in AppMap in the code editor

Automated analysis ensures the newly introduced service is performant and secure. This process can help organizations to achieve better delivery and ensure that their modernized applications are scalable, effective, and efficient. By detecting defects early in the development process, runtime code analysis during development reduces the cost of software development and improves the quality of the final product.

Challenges in using traditional observability to instrument “Dark Code”

Runtime analysis differs from traditional observability because the requirement is to characterize software without relying on production environments. Runtime analysis is far less expensive and time-consuming because it doesn't require any instrumentation and takes minutes to deploy. Large codebases are done in an hour rather than many days or weeks. Local runtime analysis includes far more granular code behavior details

than traditional hosted observability products for a significantly lower cost to the business. AppMap natively records these code behavior interactions automatically.



Function calls and loops in AppMap

Furthermore, using products like DataDog, Dynatrace, and AppDynamics for this purpose can be costly. Deploying observability solutions like OpenTelemetry (OTel) require the developers to understand their code well enough to know exactly where to add the necessary instrumentation to debug it. Many OTel projects fail simply because the developers cannot understand complex legacy codebases well enough to know where they should be adding instrumentation.

AppMap runtime data can include detailed timing information about function performance and can help identify the slow paths in your code changes before you commit them to your project. Runtime analysis works upstream of production and can integrate with other tools and processes such as CI/CD pipelines. AppMaps can be exported to commonly used documentation platforms such as Confluence. AppMap can export automatically generated API documentation to existing repository platforms such as Postman, SwaggerHub, and others.

Application modernization projects are a boon to consulting engineering companies; accelerating reverse engineering is the competitive edge companies need to stand

out. Changing team composition and the current economic climate has engineering teams thinking about accomplishing more with less and moving large complex modernization projects to third-party vendors with expertise in modernization work. The economic opportunity for consulting software development organizations in application modernization is significant.

Vendors looking to onboard client code for modernization projects can bridge the lack of documentation about the client's software and deliver results faster.

A key tool for successful onboarding to new customer code and scoping of application modernization projects is runtime code analysis. Runtime analysis with AppMap provides invaluable insights into the behavior of applications and can help you deliver fast improvements to your clients.

To schedule a demo or discuss your organization's needs with us directly, please contact us at sales@appmap.io or find us in the AppMap Community Slack channel.

References:

Fowler, M. (2016). Strangler Pattern. Martin Fowler. https://martinfowler.com/bliki/StranglerFigApplication.html

Vlietland, E. V., & Louwman, R. W. (2018). An Exploration of the Strangler Pattern for Application Modernization. Journal of Software Engineering Research and Development, 6(1), 1-23. doi:10.1186/s40411-018-0056-4

Marinescu, R. (2013). Continuous Integration, Delivery, and Deployment: Reliable and Faster Software Releases. Springer Science & Business Media.