White Paper

Top 6 Reasons to Map Your Application Code

CodeLogic
Reveal. Optimize. Innovate.

Contents

Introduction

Many organizations have messy application code – but it wasn't always that way. Back in the day when Waterfall development was the norm, application developers took a measured approach which emphasized a linear progression through successive project stages. Change was easy to manage as the frequency of code updates were dictated by a well-established and predictable timeline.

In contrast, today's development teams work at an unprecedented pace, churning out new code at a far faster tempo and rate which creates a new host of challenges that were unimaginable not long ago. This trend of accelerated code development makes it nearly impossible for everyone on an application team to truly keep up with what code is actually in the app, what has changed, and how it all fits together.

Change is Creating a Gap

The increased rate and frequency of changes means things are moving so fast no one has the time to document code changes. With every update, the codebase is different, and your development team gets even further away from truly understanding the application. In the rush to deliver the next project or get up to speed on a new app, developers are relying on tribal knowledge, outdated application wikis, or incomplete application architecture diagrams sitting on office whiteboards to keep track of what's in the application and how it works.

This lack of visibility creates a ripple effect of bad outcomes for the business, many of which aren't noticed immediately – compounding into bigger problems for more departments. Eventually, these issues grow into gaps that begin interfering with business goals.

Why? Because a lack of clarity around the code within your application introduces risk, delays projects, and reduces quality.

What gaps do companies encounter due to a lack of visibility in their code?

Typically, businesses see one or more of these three bad gaps in their business:

Knowledge Gap: When key team members leave, they take critical app knowledge with them and this puts your business at risk. Without a clear understanding of what's in an application's code or how it all fits together, organiz-ations may have to restructure teams, delay projects and increase time-to-market.

Velocity Gap: As your app's complexity increases, on-boarding new team members becomes more difficult and less efficient without accurate, up-to-date developer documentation.

Quality Gap: Application quality declines if developers and testers don't understand what is in the app and what gets impacted by a code change. This leads to bad changes which can result in a myriad of issues. From performance issues that impact the customer experience, to application outages which can impact revenue, or security issues which lead to lost customers, a PR nightmare, and potential legal issues.

Of course, none of this happened by accident. In pursuit of faster application release cycles - development methodologies, tools and team structures have changed. This spirit is part of what makes the discipline of software development so dynamic. Faster release cycles could only come to fruition through a significant disruption.

This disruption has major implications for how software is created today. And it's part of why code visibility is murkier than ever, while at the same time becoming more important than ever.

How We Got Here

The software world embraced Agile, DevOps, and full-stack development, all in an effort to achieve greater speed. Alongside these massive changes, software teams themselves transformed and shifted. A continued emphasis on growth and business outcomes inspired a philosophy of continuous change, innovation, and improvement.

- **Agile and DevOps:** To overcome silos and the limitations of traditional development methodologies organizations embraced newer approaches. As a result, Dev and Ops teams collaborate better and are able to deploy code changes faster, delivering features to customers more quickly and effectively. Unfortunately, the process of updating developer documentation is sacrificed as a result of this new-found speed.

- **Full-Stack development:** Today's full-stack teams are taking on more and more architecture related tasks. The concept of you build it, you run it, means developers are more accountable than ever to ship code that works. But this full-stack approach combined with a lack of up-to-date documentation means no one knows how everything works. Knowledge then ends up being fragmented across development teams. Wherever a developer goes, the knowledge they, and only they, possess goes with them. This has the potential to create big problems.

- **Fast changing teams:** Companies are growing and realizing they are now software companies, as a result, companies struggle to hire enough developers. With teams in flux, constant training, offshoring, and reassigning developers to other projects to cover gaps is the new normal. Combine these staffing challenges with a lack of developer documentation and increase in tribal knowledge, it's not hard to realize why it has become more difficult and time consuming to get new app team members up to speed.

- **A push to rewrite or refactor legacy apps:** Everyday companies rely on their decades old mission critical applications, written in seemingly ancient programming languages, to keep their customers happy and businesses running. This means it is vital to ensure the code that underpins these mission critical apps remains resilient and maintainable. Transforming legacy apps using more modern event-driven, cloud based, or micro-services architectures can breathe new life into old code, but it comes at a cost. Teams are now forced to understand what's in the legacy code base and how it all fits together before they can make recommend-dations on how to refactor or rewrite the code. Without this level of insights and intelligence, it is difficult to know which parts of the older code base are candidates for modernization and which are not. This lack of understanding can take developers down a rabbit hole that leads to excessive cost, risks, and efforts that may not yield results.

- **Migration off expensive database platforms:** Some organizations are tired of paying a hefty price for their tried-and-true database management platform. This leads to a movement to migrate to a less expensive alternative. While there are many tools available to help teams migrate data from an old platform to a new one - there's more to it to achieve success. To successfully perform a database platform migration teams must be able to see all the connection points between the app and database. Without an accurate, up to date view of all the app to database connections and dependencies, database migration efforts are likely to be costly, time consuming and prone to failure.

- **Enterprise adoption of open source:** The adoption of open-source software has become commonplace among enterprises. Unfortunately, there are potential security and legal risks that go along with it. Therefore, it becomes important to understand how the open-source code connects to the rest of the codebase and where the dependencies are, if the open-source code must be changed or removed.

All of these trends have something in common - it's all change that's difficult to anticipate, quantify, and communicate to others. As you develop a better appreciation for the impact of change, you begin to realize how important it is to document and share it. Without truly knowing what's in the code and the connections and dependencies that exist within it, development teams are essentially flying blind every time they want to make change. As a result, the projects they work on are at risk of missing deadlines, running over budget, or failing altogether.

Map Your Code and Close the Gaps

By mapping application code and seeing all the hidden connections and dependencies, teams gain a new level of vital application intelligence and insights. Application teams can use this intelligence to overcome the range of challenges highlighted at the beginning of this paper. Dependency mapping is helpful for developers, architects, SRE’s, testers and technology leaders who all need an in-depth understanding of the code underpinning their applications.

Here are the top 6 reasons teams should map their application code:

1 See Hidden App Modernization Bottlenecks

Refactoring or rewriting an application can be a scary proposition – you’re essentially breaking the application apart without necessarily understanding how it fits together in the first place. Not only do you need to know how the application works before you begin making modernization decisions, but you also need to know how it’s impacting the other applications it communicates with.

Before rewriting or refactoring your application, you need deep visibility into the connections and dependencies that exist within and across your applications. Mapping app code allows developers and software architects to see what areas of code are ideal candidates to be rewritten or converted into microservices and what areas are not. As dependency mapping reveals the hidden layers behind your app, you can identify the best places to begin rewrite or refactor efforts first. More challenging parts of the application can be avoided or scheduled to be completed with your best developers assigned to the task.

2 Uncover Technical Debt

Hidden technical debt can be destructive to your application’s performance if your team isn’t ready to plan, manage or reduce it. In the moment the debt is added, your team may have delivered a product faster, created an effective prototype, or gained some other business advantage – but just like financial debt, technical debt eventually demands repayment and begins dragging productivity or value down. If your team is regularly paying a high price for technical debt, it’s time to identify where it exists and develop a plan to remove it.

Mapping code exposes each connection and dependency from class, to method, to database. Making it easier to see problem areas such as which database tables and columns are being overused and which are not being used at all. Addressing these areas will help eliminate technical debt and improve application performance.

3 Onboard New Developers Faster

If you have better developer documentation, then you can save time and improve your team’s productivity. Code mapping allows developers to see all hidden connections and dependencies in your application. Accordingly, they can better understand the code item they are working on and clearly identify all the classes, methods and database connections that are impacted by code changes.

When new developers on a project see the underlying connections and dependencies in an application, they are better prepared to bring value to your organization from day one.

4 Understand Exposure of Open Source

Today, many software development teams are turning to open source solutions to help them move faster. As a result, the way software applications are developed and implemented is changing. Compared with custom software development, open source can be more cost effective and less likely to lock-in organizations to a particular vendor.

Open source has many advantages, but its growing popularity is also revealing some potentially significant tradeoffs – a lack of official support, security vulnerabilities, and sometimes unpredictable licensing challenges that can create risks for your business.

Mapping your code shows where your open source software exists to highlight potential security, support or legal risk. Once you know the risks, managing the potential downsides of open source code adoption becomes more straightforward.

5 More Accurate Test Case Coverage

Just like the development team, your test pros need good information to guide their work. Code mapping helps ensure that their test case coverage addresses all the classes, methods and database items impacted by code changes
the developers are building. By mapping the application code, your testers can be confident they are running the right test cases and deliver better results for the business.

6 DBMS Platform Migration

Database vendor lock-ins can lead to excessive expense for organizations – so today many companies are looking closely at more cost-friendly options. As application teams look for alternatives to their current database platform, the ability to accurately map database connections across an applications codebase can be helpful in creating a more effective database platform migration plan.

One of the biggest challenges created by vendor lock-in is a lack of visibility that builds up over time as your application grows in complexity. Switching a database and and undertaking a database platform migration can be time consuming and risky. A truth that some legacy vendors undoubtedly count on.

Understanding and seeing all the database connections across an application allows you to better estimate the platform migration effort and develop a realistic plan of attack. With the full story at-a-glance, you will reduce the likelihood that you'll be caught by surprise.

An Opportunity for Organizations

Thinking about trends and challenges presented in this paper, how many of these reasons apply to your team or organization? Code dependency mapping is valuable for a variety of different teams, roles, and projects. An automated approach to mapping, documenting and visualizing application code connections and dependencies enables greater productivity and accuracy.

To learn more, visit us at www.codelogic.com

CodeLogic
Reveal. Optimize. Innovate.