EBOOK

Tackling the legacy code challenge

diffblue
AI for Code

Tackling the legacy code challenge

More of the world's businesses and public services depend on well-written code than ever before. However, with rapid innovation and the constantly evolving roles of employees and contractors, the knowledge of the code in an organization fades quickly, and code that is sometimes only a few years old becomes complex to maintain. Global companies are investing billions of dollars into digital programs that become full of legacy code in a handful of years, and with the move to cloud and microservices, the need to understand and transform monoliths and legacy code has become an imperative.

The characteristics of such software are the same across various organizations, especially in those highly reliant on custom software (like banks and technology firms): a piece of essential code that works but has a poor test suite. When changes are required—from small bug-fixes, to new features and requirements triggered by compliance—immense resources can be needed to ensure that even the slightest change does not break unrelated parts of the codebase. And as time progresses, the amount of technical debt incurred by ad hoc fixes increases, which means that the cost and resource escalates. In 2015, Gartner estimated worldwide technical debt at more than $1 trillion, and development teams are scared to wade into this growing mess.

Current Alternatives

There are several established options for tackling the problems of legacy software, including:

- Rewriting code from scratch
- Using code refactoring tools
- Employing contractors to test and document

• Redeploying developers working on otherwise high-value applications to meet business needs

Rewrite Code

Out of all of the approaches, this is probably the most common and instinctive response from the various stakeholders. After all, if something in the total system is considered old and difficult to maintain, wouldn't just replacing it be the fastest, lowest risk approach? Well, not necessarily. First of all, almost any rewrite will require substantial additional resource and time; delays and high cost may not always be practical for the business. Secondly, the introduction of new software will no doubt increase the risk of new bugs. Third, redeploying existing staff is likely to delay other projects and deliverables, perhaps significantly. And last but not least, there are often faster, less expensive, lower-risk methods that can be employed to fix legacy code.

Use code refactoring tools

Products like Eclipse, NetBeans and Visual Studio offer basic refactoring functionality in their product offerings. Typically, these tools are useful in that they:

- help developers rename and safely delete instances and classes
- extract interfaces, superclasses, parameters, etc from methods
- rename default namespace, remove unused references, etc.

While these capabilities are all nice to have, they work on a very local level. They neither improve code readability, nor do they boost test coverage; therefore, the code remains challenging to both change and maintain.

Employ contractors to test and document

Appointing external contractors to plan and execute tests and to document legacy code is a good solution in some cases. IT partners tend to be experts in the field and experienced at delivering quality results, quickly. However, while external partners may help deliver a solution to the legacy problem, it is rarely going to be a one-off cost, since legacy code takes a while to tame. In the meantime, new legacy code will be appearing. Of course, going offshore can help reduce total costs, but the budget can still run into millions of dollars per annum.

Redeploying developers working on otherwise high-value applications to meet business needs

While business leaders often understand that unexpected interactions and bugs happen, they have a plan to move the business forward, and this is mostly driven by creating and enhancing applications. So, while developers can always be repurposed to meet pressing escalations and issues, this should be a last resort. Often, incidents and diverting resources are a clear driver to expand teams, change architectures or take on application transformation programs.

Diffblue's approach

There is another, newer, approach to maintain legacy code in concert with any or all of the above. We have created Diffblue Cover, an AI-based application that automatically generates the test suite for existing code, without human intervention and the associated delays. More tests means that unintended changes can get identified early, and do not create problems further down the development pipeline or, worse, in production. Diffblue Cover will enable you to understand the impact of changes to the source code and make more informed decisions about development, helping you know your code better and avoid regression bugs.

Under the hood, Diffblue Cover uses Diffblue's AI engine, specifically designed to work with code. The engine scans the codebase, reasons about possible executions of a program, creates minimal, meaningful unit tests, and presents them to the technical user for consideration. At Diffblue, we believe that AI is bound to become developers' best friend, relieving them of mundane and stressful tasks, such as fixing unknown codebases, and instead letting them spend their time on creative work and problem-solving.

Learn more at www.diffblue.com