Low-code software development

// How to maintain a long-term advantage

Dr. Rick Klompé

Software Improvement Group

Low-code software development

How to maintain a long-term advantage

Author

Dr. Rick Klompé

Senior Consultant at Software Improvement Group

Contents

The rise of low-code platforms

Digital transformation is changing the way enterprises develop applications. Under increasing pressure to quicken their time to market, organizations across every industry have caught on to the advantages of low-code platforms as a way to support rapid development.

Gone are the days of month-long development cycles and long, silo’ed processes. Enter a new era of low-code software, where staff in various departments can create their own applications, relieving the pressure from their colleagues in IT.

But with the increasing number of business-critical applications developed on low-code platforms, it is essential to apply development best practices to these technologies. This enables organizations to avoid pitfalls, get deliverables “first time right” and ensure long-term flexibility and agility.

Applying software development practices to low code

Low code is a subset of technologies based on visual coding. Although it differs a great deal from traditional technologies, low-code development is still software engineering. That means the same best practices for line-based software development must be applied. Failure to do so opens the door to quality issues that slow down business innovation - and negates the low-code benefits the organization sought in the first place.

Solid software quality pays off, whether low-code or line-based. New functionality can be added and defects can be resolved much faster and cheaper when an application has a high level of maintainability. For low-code however, these practices require a slightly different approach for success.

A new kind of developer

As the graphical user interface allows non-technical business users to build an application themselves, it is essential to train these audiences not only in the specific guidelines for the low-code platform, but also to provide them with awareness around the importance of the technical quality of their product. Furthermore, they need to have an understanding of the basic principles of good software engineering.

Guidelines for developers

From the very start of development, technical quality must be managed and maintained to make sure that defects can be fixed quickly and new functionality added efficiently. Particularly for low-code platforms, the development process can be very Agile with shorter and more frequent iterations. Since the new developers are closer to the business, functionality can be delivered faster. There is a tendency for them to focus on implementing individual features to the exclusion of the overall architecture or landscape. Long term, low technical quality and architectural issues will hurt the agility of the application.

Therefore, development teams should take the following five guidelines into account to improve the technical quality of their low-code applications:

- Keep objects short and simple
- Avoid hidden code
- Minimize object duplication
- Reduce structural complexity
- Take architecture design into account

Guidelines for organizations

As more and more organizations make low code a significant part of their software landscapes, management should consider the following three guidelines to prevent issues leading to reduced business agility and longer time to market:

- Foster an internal culture around technical quality
- Actively manage your landscape
- Check technical quality regularly

With an understanding that the same software quality principles apply to low-code development, development teams and their management put themselves in the best position possible to capture the benefits promised by this new technology – and maintain them in the long run.

Low-code platforms

A promise for rapid application development

Low-code software development has made it easier and more efficient to build business applications. With intuitive visual user interfaces and pre-fabricated building blocks, these platforms empower the “citizen developer,” people with little or no coding experience.

This has allowed companies to move development of new business functionality from IT to other departments, shortening the development chain and eliminating miscommunication and misinterpretation between the business and IT.

However, developing business applications with low-code platforms still requires software engineering skills to ensure the end product is high quality and future proof.

Rapidly increasing takeup

Usage of low-code platforms is growing fast and furiously, with the market expected to grow over the next few years to dozens of billions of dollars. Some estimates suggest that these technologies will significantly replace traditional, line-based software development languages by up to 30-50%.

There's hardly a company to be found that hasn't used low-code platforms for application development, or, at least, a proof-of-concept. And it's easy to see why. Low-code platforms promise a myriad of benefits, such as quicker time to market for new applications, seamless integration with other systems, and ease of application maintenance.

Software engineering skills still required

Of course, building applications with low-code technologies differs from traditional software engineering in that it doesn't entail writing lines of code. Low-code platforms allow coding at a higher-level, using the visual language. But it's still a form of coding, which means that best coding practices still apply.

After all, applications mature and grow in both size and complexity over time. Those initial small and simple units become larger and more complex as new functionalities are introduced. With the clock ticking, development teams can easily find themselves compromising on software quality. Technical debt starts to snowball and business innovation slows - undoing the benefits of rapid application development.

There are a few reasons for the problem here. First, basic software engineering practices aren't usually applied. Developing applications with low-code platforms isn't typically thought of as software engineering, as business functionality is created without typing lines of code. Functionality is implemented through pre-fabricated building blocks that are configured and connected in the required sequence using form-based wizards and drag-and-drop user interfaces, thus hiding a lot of complexity.

Second, it stands to reason that people with little or no coding experience also lack knowledge of development best practices; nor do they have awareness around the importance of technical quality and limiting technical debt.

Finally, metrics and tools to measure and manage the technical quality and maintainability of this (low) code are not yet widely available. As a result, quality management for low-code applications is often limited to correct usage of the technology only. This means that the focus is on the proper use of the pre-defined functions to the exclusion of avoiding bad constructs. Software quality aspects like maintainability are equally important to the design and creation phases, but get little attention.

How to improve the technical quality of low-code applications

5 guidelines for development teams

Having measured the technical quality of a low code application, it's now time to manage and improve its quality level. From the start of development, the technical quality must be kept under control and maintained at a sufficient level to ensure that both resolution of defects and addition of new functionality can be executed quickly.

Development teams should take the following five guidelines into account to improve the technical quality of their low code applications:

1. Keep objects short and simple

If the complexity of the implementation is minimized, it will be easier to analyze, modify and test the application. Low complexity is accomplished by keeping implementation objects like workflows, transactions and mappings simple and short, and by limiting the amount of logic that each object contains. The single-responsibility principle, a widely-accepted best practice in general software engineering, is also applicable to low code.

2. Avoid hidden code

When developing low-code applications, functionality is implemented by using pre-fabricated building blocks. These standard blocks have to be configured and employed in a workflow in such a way that they work together properly to perform the required functionality. These building blocks can be configured by (de)selecting several options and defining dependencies with the other blocks in the workflow.

With all low-code platforms, the engineer can include a great deal of logic in a building block by inserting a specific piece of code into a blank field. But this comes with significant risk; this additional functionality isn't clearly visible when maintaining the application, workflow, specific transaction or mapping. We refer to this as hidden code.

Hidden code can pose a threat to maintainability, because there are no checks on its quality. This often results in long, highly-complex functions. Since the code is hidden inside of a transaction or mapping, there's no option for reuse other than to copy the code, resulting in a great deal of duplication.

3. Minimize object duplication

Generally speaking, duplication is something that should always be avoided in software. It usually leads to a higher maintenance burden and significant maintenance risk, as duplicates can easily be forgotten when making changes. This same principle holds for low-code applications; duplication in workflows, transactions or hidden code increases the size of the application to be maintained and introduces a risk for inconsistent operations if changes aren't implemented for all duplicates.

Given the nature of low-code platforms and their aim to enable rapid software development, low-code engineers tend to work on implementing (or changing) a specific, individual functional feature. As a result, they don't operate from a software system architecture and may even lack a system overview. This can lead to (partially) duplicated workflows, transactions and hidden code, which slow down development of new functionalities over time. We often see the results in high duplication percentages (up to 30%) sometimes within a single application, but also between applications. Moreover, most low-code platforms don't provide insight into any form of duplication.

4. Reduce structural complexity

The structural complexity of an application is defined by the number of relationships between its various parts. For low-code platforms specifically, this means dependencies between workflows, transactions or architectural components implemented in other technologies. When implementing new functionality by creating or updating a workflow, dependencies can be defined to connect with other workflows. If these dependencies aren't managed and don't follow a well-thought-out design, the result can be a highly-interwoven application with "spaghetti code." And that means a new legacy monolith has just been created in modern low-code technology.

To prevent this, a higher-level architectural design is required that defines the dependencies between (groups of) workflows or components. In addition, the implementation must be checked against the design.

5. Take architecture design into account

Although low code is promoted as a rapid development environment in which new functionality can be quickly implemented, it's best to work from a software architecture perspective while developing and think ahead about the target end state before starting to implement new functionality.

Software architecture is highly relevant to low code too, as low-code applications should not consist of an unorganized set of workflows and transactions. Design principles, like separation of concerns, single responsibility and reuse, are important to prevent ending up with a highly complex, interwoven and duplicated system with the same maintainability issues seen in monolithic legacy systems.

During development, the architecture should be further specified and the implementation checked for compliance with the design. For all low-code platforms, this will be a manual peer-review activity, as they don't yet provide insights into this architectural overview.

Making low code a significant part of your landscape

3 guidelines for organizations

An increasing number of organizations are building more and more of their applications with low-code platforms. Some have selected low code as the main technology for the renewal of their application land-scapes. Others use it more selectively. Either way, low-code technologies are becoming a significant part of application portfolios.

Organizations should take the following three guidelines into account to prevent issues leading to reduced business agility and longer time to market:

1. Foster an internal culture around technical quality

An organization can create maximum value with its development tools and the people who use them only when it commits to making technical quality an integral part of its ethos and operating practices. By now, the point is clear that low-code development is still software engineering. Management should take practical measures to help its people working with low-code platforms to understand this and adopt best practices accordingly.

The development process applied for low-code platforms should include the same software development best practices for other technologies. For example, before starting, a proper (high-level) design is needed and must be further developed and detailed during development. Design and implementation decisions should be documented, and quality control must be implemented and enforced as well. Furthermore, automation of the development process and testing will significantly contribute to the quality and agility of the applications.

2. Actively manage your landscape

When implementing a lot of business functionality in low code, it is advisable to do so using small applications. This landscape of applications has to be managed as well. It's important to keep in mind that every technology has its strengths and weaknesses; organizations should therefore be selective as to which of its capabilities they make use of. In addition, the relationships and communication between the applications must be designed and managed, and functional duplication in the landscape also prevented.

Generally speaking, this principle holds for implementing a (microservices) landscape, but particularly for landscapes developed with a low-code technology, since functionality can be implemented rapidly and functional duplication can be created rapidly as well. Staying in control of the quality of your application landscape requires code quality tools that don't only measure the technical quality of the individual applications, but dependencies and duplication at the landscape level as well.

3. Check technical quality regularly

As for any other software technology, the quality of the implementation has to be checked regularly during development, not just at the end. Quality has to be built in and implemented by design. In addition, quality should include more than just operating according to functional specifications; other product quality aspects, like maintainability, reliability and security, are just as crucial.

At the start of a development project, it may seem like too much overhead is slowing down the speedy development and high productivity promised by low-code technology. But the upfront work will pay off later. Studies show that development projects stay on track and agility remains higher when the code is high quality. Although these figures don't yet exist for low-code platforms, first signs are apparent that this holds true for low-code as well. In working with our clients, we at SIG see that companies which have been employing low-code development for a longer period of time see a drop in their development productivity if the technical quality of their applications deteriorates. When productivity of a low-code platform lags behind, it's usually either because the team is still in a learning phase or the application's poor technical quality is slowing it down.

Going forward with low code

Keep development velocity high by actively steering on quality

Developing business applications with low-code platforms can speed up an organization's innovation speed and agility. However, the benefits of a low-code platform can be lost over time if technical quality is neglected and the technology not properly used.

Development teams and their organizations must recognize that implementing business applications with low-code platforms is still software engineering, despite the fact that the software artefacts are made up of workflows and transactions instead of lines of code. Software development best practices can and should be applied for low-code platforms as well.

The only way to keep low-code development velocity high is by actively steering on software quality. The eight guidelines discussed in this e-book will enable your team and organization to deliver highly-maintainable applications at speed; prevent the technical debt snowball; and ultimately capture the many benefits of low code.

About the Author

Dr. Rick Klompé is a Senior Consultant at Software Improvement Group (SIG), where he brings 20 years of experience to help client organizations effectively manage and govern their IT.

Prior to SIG, Dr. Klompé held positions spanning consultancy, IT management and control primarily in the telecom and financial services industries.

Dr. Klompé holds a PhD in Informatics from the Delft University of Technology in the Netherlands.

About SIG

Software Improvement Group (SIG) helps business and technology leaders drive their organizational objectives by fundamentally improving the health and security of their software applications. SIG combines its proprietary tools and benchmark data with its consultants' expertise to help organizations measure, evaluate and improve code quality - whether they're building, buying or operating software.

As an independent organization, SIG has the industry's largest benchmark with more than 25 billion lines of code across hundreds of technologies. The expert consultants at SIG use the benchmark to evaluate an organization's IT assets on maintainability, scalability, reliability, complexity, security, privacy and other critical factors. The SIG laboratory is the only one in the world accredited according to ISO/IEC 17025 for software quality analysis.

SIG is headquartered in Amsterdam, with offices in New York, Copenhagen, Antwerp and Frankfurt.

Learn more at www.softwareimprovementgroup.com.