Codeless vs. Code-Based Application Development and Customizations

Understanding the differences between them and how they can help customers tailor applications
Table of Contents

03 Executive Summary

04 What is code-based vs. codeless development?
05 Principles of codeless development frameworks
05 Visual UI designer
05 Dynamism in the runtime UI
05 Implicit support for CRUD operations
05 Ability to write business logic
06 Ability to define permissions
06 Output for various devices
06 Integrations
06 How codeless, low-code, and code-based development differ?

08 Customizability vs. configurability
08 Customization best practices and tips
09 Configurability best practices and tips

10 Codeless and code-based development frameworks offered by BMC Helix
10 BMC Helix ITSM workflow development platform

11 Conclusion
Historically, application development is done by writing code with technologies that are trending. Different programming languages and frameworks have evolved over the last several decades and helped with developing applications for specific business needs, providing the flexibility to write an application from scratch or use available frameworks.

Recently, the use of codeless or low-code frameworks to develop applications has been gaining traction due to faster time to market and ease for application developers. It is also becoming increasingly popular since there is a lesser need for specialized skills like knowing a programming language.

This white paper will provide some insights into different types of application development techniques such as code-based, codeless, and low-code. There is a greater focus on providing details around relatively lesser-known codeless development frameworks, but this paper will also help those who are considering different approaches.

The paper also sheds some light on how application developers can allow their customers to customize or configure the applications they develop.

Finally, the paper provides guidance on various scenarios and conditions that should be considered and clearly calls out that there is no one-size-fits-all solution when deciding which technique to use.
What is code-based vs. codeless development?

Code-based or coded development is the oldest, most traditional, and still most popular method of application development for building applications that satisfy business requirements for a particular customer or situation. As programming languages have evolved, new ways of development using natural languages and frameworks have emerged, but the basic concept of code-based development remains the same.

Codeless or no-code application development is not a very new concept. BMC has been one of the leaders in providing solutions for codeless application development for many decades. Usually codeless frameworks help create user interface (UI)-based applications quickly, providing tools with WYSIWYG (What You See Is What You Get) capabilities, which allow application developers to create prototypes and develop simple UI-based applications very quickly.

Developing complex business logic and integrations are key requirements for any enterprise application. While code-based application development allows a lot of flexibility for achieving these requirements, codeless frameworks do as well, without writing a single line of code.

An approach that’s a happy medium between codeless and code-based development is a low-code approach, where most of the basic application development is implemented with codeless framework and any additional complex UI or business logic, as well as integrations, are implemented with a low-code approach.
Principles of codeless development frameworks

Frameworks for codeless application development should be very generic and flexible to allow for development of an application from any domain with any functionality. Typically, such frameworks make use of a metadata layer to store screen or component info. Key principles for such frameworks include:

Visual UI designer

It’s very important for codeless application development frameworks to have a UI designer that supports drag-and-drop of UI elements to construct a fully working screen. There should be a rich library of UI components, including the basics such as text fields, numeric fields, dropdowns, buttons, radio buttons, checkboxes, grids, tab panels, images, menus, and file upload. It’s also preferred to have a WYSIWYG output capability so that an application developer can easily build the screens without worrying much about the runtime view.

Dynamism in the runtime UI

Codeless frameworks should have a facility to allow developers to control certain display aspects at runtime. This includes controlling UI properties such as:

- Showing or hiding a UI element or a section with multiple UI elements
- Enabling or disabling a field
- Changing display value(s) and contents
- Changing labels
- Changing positions of the fields (a nice to have)

Application developers should be able to define these dynamically based on user inputs or actions such as clicking a button, selecting a value from a dropdown or radio buttons, or clicking on a UI element, etc. There should also be support for implicit events such as screen loading or unloading or data being submitted to the server.

Implicit support for CRUD operations

As described earlier, codeless development frameworks are used for quickly creating simple UI-based applications that mostly involve CRUD (Create, Read, Update, and Delete) operations. Without much effort from the application developer, the framework should allow application entities to be defined, and a runtime view should allow searching, creation, reading, updating, or deletion for those entities. Preferably, there should also be a way for the end users to search for the data of their interest with query-by-example (QBE)-style searches.

Ability to write business logic

While the focus of a codeless development framework is primarily to provide very good UI support, almost none of the application can be complete without writing business logic. Part of the business logic can be implemented on the client side using UI-side constructs that can prevent invalid user inputs. However, it’s never a good idea to write business logic on the UI layer, especially for web applications, as a UI layer can be easily bypassed or manipulated using tools like browser development tools, fiddler, and any network tools.

Important validations or business processes such as status transitions should never be written in the UI layer. It’s a best practice to have such logic on the server side in the business logic layer so that no one can bypass it. A codeless development framework should allow a workflow application to
be written through server-side constructs without the subject matter expert (SME) needing to know how it is implemented. Usually this is achieved via a visual designer or a set of workflows that allows for creation of a flowchart or stepwise logic using a workflow design.

**Ability to define permissions**

Just like business logic, a UI layer should be avoided when dealing with permissions for a screen, fields, or data within a screen. A codeless development framework should allow permission models to be defined by means of server-side constructs in a codeless way. The application developer should be able to define a permission model by creating groups or roles in the system and mapping them to permissions such as availability of screen, functionality, fields, or even data for a specific set of users. The implementation of enforcing such a permission model must be done on the server side by the framework to avoid any security issues.

**Output for various devices**

While this is not a requirement for a codeless development framework, it’s a very important consideration to allow developers to use the write-once, run-anywhere principle.

A developer should be able to define UI and business logic once to get the output they desire, agnostic of the device on which the application is being used, i.e., a browser running on a computer system with any screen resolution or a portable handheld device like a cell phone or tablet.

Recent advances in progressive web application (PWA) technology and a codeless development framework make it easy for developers to develop responsive and progressive web applications that respond to the device type, screen resolution, and available form factor. Write-once, read-anywhere applications can be created without the developer needing to understand the underlying technology that powers such responsive and adaptive output.

**Integrations**

Most applications, especially enterprise applications, must be integrated with other applications to fetch or send data to complete an end-to-end business use case. Historically, due to multiple technologies and a lack of standards for integrations, it has been difficult to integrate multiple applications developed by multiple vendors with the customer application. This process of integration has been simplified with newer, standard technologies such as SOAP, REST, JMS, and Kafka, and data format standardization such as XML or JSON. It is, however, still a challenge to generalize the data structures and other expectations such as fault tolerance while integrating with different applications.

It is valuable for a codeless development framework to provide basic integration capability, at least via a JSON-over-REST mechanism, to give application developers a way to easily integrate with other applications without needing to write any code. Usually, these frameworks provide a way to define the JSON structure on both sides and the mapping between them. These frameworks also provide a way to create connections through REST endpoints that include URL, http method, headers, and other parameters.

**How codeless, low-code, and code-based development differ?**

Codeless and code-based development have fundamental differences in both the way the application is developed, and the skillset required. The low-code approach tries to pull best of both worlds.

The following table compares each approach.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Codeless</th>
<th>Code-based</th>
<th>Low-code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need for special skills</td>
<td>Almost none</td>
<td>Very high</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Any special workflow constructs are provided by framework.</td>
<td>Knowledge of UI and backend technologies and languages is a must.</td>
<td>Since minimal code is required, special skills requirements are limited.</td>
</tr>
<tr>
<td>Speed of development</td>
<td>High</td>
<td>Low to medium</td>
<td>Medium to high</td>
</tr>
<tr>
<td></td>
<td>Time to market is quick if integrations and business logic are limited/simple.</td>
<td>A lot of design and development is needed to implement end to end.</td>
<td>Expectation is to have 90 percent of the system ready in a codeless fashion; the remaining ten percent takes time.</td>
</tr>
<tr>
<td>Efforts for non-functional compliance</td>
<td>Medium to high</td>
<td>Low</td>
<td>Medium to high</td>
</tr>
<tr>
<td></td>
<td>Codeless frameworks provide built-in support for security, performance, and scalability, which limits application developer potential.</td>
<td>Security, performance, and scalability can be designed and implemented right from the beginning of the design, and full control in code is available for any needed changes.</td>
<td>With a codeless framework as the backbone, it's almost the same as codeless application development.</td>
</tr>
<tr>
<td>Limitations in development</td>
<td>Medium to high</td>
<td>Almost none</td>
<td>Medium to high</td>
</tr>
<tr>
<td></td>
<td>Many limitations may exist to allow generic development and protect the framework from unsupported constructs.</td>
<td>Almost no limitations exist since these are hand-coded special apps that can be easily changed to achieve any technical implementation supported by the technology.</td>
<td>The same as codeless application development.</td>
</tr>
<tr>
<td>Ease of customizing applications</td>
<td>High</td>
<td>Low to medium</td>
<td>Medium to High</td>
</tr>
<tr>
<td></td>
<td>Since these frameworks are metadata driven, they easily support customization constructs for application developer and customers.</td>
<td>Applications cannot be customized easily, and developers must make provisions for all possible configurations that allow tailoring the behavior.</td>
<td>The same as codeless application development, except for the parts which are coded.</td>
</tr>
<tr>
<td>Ease of integrations</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Limited flexibility due to dependency on generic features available from the framework.</td>
<td>Complete flexibility to define interfaces and integrations with a code-based approach, without any limitations.</td>
<td>Integrations are typically done with a code-based approach and provide complete flexibility.</td>
</tr>
</tbody>
</table>
Customizability vs. configurability

If an application is developed and owned by a single customer, it’s easy to make changes to the application to meet a new requirement or a change in its existing logic for that customer. However, if the product is generic and caters to multiple customers (i.e., software as a service (SaaS) or generic products like BMC), it is very difficult to achieve a change for one specific customer unless it’s a generic change that is applicable to all customers.

While both codeless and code-based development techniques can be used effectively in developing applications from scratch, there are some differences when it comes to tailoring applications to meet a customer-specific requirement or business logic.

Codeless application development frameworks require a special provision to allow customers to easily tailor their application in a codeless fashion. This is typically done by allowing codeless framework metadata changes using the same designer tools. This is called customizability of the application. Since these frameworks are already built to function generically, customizing them is relatively simple.

A code-based application, on the other hand, is not as flexible and it’s difficult to make changes to the application behavior without touching the code, so a developer must think of all possible tweaks that a customer may require for the application to behave differently. The developer must provide adequate configurations for customers to change the application’s behavior or UI to suit their needs. Customers may still not be able to change complex processes or business logic with this approach.

Customization best practices and tips

Customizing applications, which is typically done in a codeless fashion, is a double-edged sword. While it allows customers to change the complete look and feel or behavior of the application, it can also diminish functionality if done badly. When a codeless application development framework is used to allow customizability, some best practices must be followed, including:

- The developer decides whether the UI/ functionality can be customized, and customization should be restricted where possible.
- Retain any customizations in a separate metadata layer to avoid losing them when the base functionality changes.
- Never overwrite any out-of-the-box logic but provide a way to disable or enable it.
- Give customers an easy way to reconcile their changes after upgrading to the newer application.
- Do not customize with hacks or in ways that are unsupported and can potentially cause long-term issues.
- Clearly define the distinction between customizability and configurability.
- Provide codeless constructs for customization to allow customers to use the same WYSIWYG designers, even for tailoring their presentation logic.
Configurability best practices and tips

Typically, configurability means providing customers with a set of properties with pre-defined values to change an application’s UI or behavior, such as a configuration setting to show or hide certain fields or sections based on certain conditions. The conditions should be pre-defined so that customers can easily change them. Configurability is a must for code-based applications, but it can also complement customizability for codeless applications.

Some best practices to follow for configurability include:

- Always provide a central place to store the settings and a generic UI to manage configuration settings and their values.

- Provide a superset of the configurations and options that can be used for each setting so customers can always choose a valid value.

- Think of all possible scenarios that a customer may need at runtime while using the application and make a provision for each of them via a configuration setting.

- When implementing different logic based on inputs, provide a configuration setting that drives the implementation.

- When you are unsure about customer preferences, provide a configuration setting to quickly tweak the behavior of the application based on feedback.

Customizability of the application provides a way for customers to tailor application UI or behavior in a codeless way. While this is one of the most powerful features of codeless development frameworks, it can get ugly when a new application UI or functionality is introduced by application designers in the base application. Unless the customizations are re-evaluated in the context of new changes from a new version, they may start breaking the application unexpectedly.

To avoid such unexpected issues, there should always be a way for customers to easily view and reconcile their earlier changes versus the changes of a new version. There should also be visual reconciliation tools to see the impact of any changes and resolve issues as they arise.
Codeless and code-based development frameworks offered by BMC Helix

For decades, BMC has been a leader in providing frameworks to customers for codeless and code-based application development and customization, evolving its solutions into more user-friendly frameworks to create and customize applications easily with modern UI and technologies.

**BMC Helix ITSM workflow development platform**

With years of development and innovation, action request systems (ARS) have allowed customers to customize their IT service management (ITSM) solutions with creative innovations. Starting earlier this year, we announced that this would be folded into the **BMC Helix ITSM** platform, once customers have migrated to the solution. This allows customers to literally “run and reinvent”—running business-critical custom applications while modernizing with new ways to enhance, extend, and customize BMC Helix as they journey towards becoming an **Autonomous Digital Enterprise**, our future-state business framework.

With the BMC Helix Innovation Suite, BMC Helix now has more enhanced, powerful features that allow customers to extend, enhance, and customize processes and workflows and even create innovative applications and services using their choice of no-code, low-code, or code-based methods. In the no-code and low-code space, BMC Helix provides a powerful process designer that allows business analysts to define business processes using Business Process Model and Notation (BPMN)-style flowcharts without needing to write code to change application behavior.

The BMC Helix Platform also provides a robust professional development environment; including software developer kits (SDKs) that allow creating, enhancing, and integrating applications with Java and Angular technology to provide a modern experience to users. BMC developers can use this platform to deliver solutions in lines of business such as HR, multi-cloud management) area. See **this document** for more details.

BMC uses this platform to build, enhance, and deliver solutions in the ITSM space. It involves designing and developing UI using the WYSIWYG-based Developer Studio designer, which allows writing dynamic UI logic and server-side business logic in a codeless way using workflows. These Developer Studio features also enable customers to customize UI and business logic as needed in a codeless way. See **this document** for more details about this framework.

With recent Progressive Web Apps (PWA) technology advancements, this codeless framework has been further enhanced to provide a more modernized look and feel for applications, allowing developers to make use of the aforementioned write-once, use-anywhere principle to create views that are equally responsive and adaptive on traditional PC-based web browsers or mobile devices. See **this document** for more details.
Conclusion

It is possible to develop a UI-based application through code-based or codeless development techniques. BMC Helix empowers customers to choose which type is the best application development method for enhancing, extending, and customizing applications to meet their preferences and requirements.

Here is some guidance on making the choice between codeless vs. code-based development.

- When quickly developing simple applications that are UI-heavy, such as CRUD-like simple use cases, codeless development clearly provides faster time to market.

- When complex, proprietary business logic must be abstracted or hidden by the application, using code-based application development is a better choice due to its additional flexibility.

- For easy, quick customizations, metadata-driven codeless application frameworks eliminate the customer’s worry about managing specialized technical skills.

- If you’re providing several possible configurations for a code-based application, make sure that they really provide value to the customer and that the customer understands the impact of each on the application.

- A combination of codeless and code-based application development i.e., low-code development, is better for most enterprise applications that will be changed frequently. It provides enough flexibility for business analysts to tailor the business flow without needing to know technical skills and gives technical experts the ability to hide the complexity in components and modules.

For more information
To learn more about the codeless versus code-based application development, please visit https://www.bmc.com/it-solutions/bmc-helix-business-workflows.html