Technology Trends

Low Code Application Development

Enterprise Architecture, Chief Technology Officer Branch

Version 0.1
Date 2019-5-8

Shared Services Canada | Services partagés Canada

Canada

Table of Contents

Business Brief

A *Low-Code Development Platform* (or simply *low-code*[1]) is a software development *environment* and *platform* that enables modern programmers to develop *application software* (apps) through graphical user interfaces instead of writing code in a traditional programming language. While originally intended for applications involving databases, business processes, content/document management systems, and web interfaces, low-code is now mature enough to develop almost all types of applications except for those deeply embedded, or very high-performance (such as games or scientific computing). Low-code has its technical roots in *rapid application development* (RAD) and *fourth generation programming languages* from roughly 1992 to the early 2000's.

Low-code is built on the concept of *higher abstraction levels*, where the developer can directly express business processes and requirements without getting mired in coding details. For some apps developed in low-code, a small amount of simple code is typically still written by hand – especially for applications not following one of the vast number of built-in templates. (Those requiring absolutely no handwritten code are referred to as *no-code apps*.) This reduction in manual coding has some important effects:

- App development (including feature capture, testing/validation and deployment) proceeds much faster than usual – enabling *agile development*, and reducing cost and improving time-to-market. As a side effect, this also reduces the number of errors and requirements mismatches.
- Less coding (and less intricate coding) allows a broader range of people to engage in app development – no longer limiting this to highly skilled programmers (who are typically rare, talented, and expensive).
- Low-code originally catered to apps-from-scratch, but is now also able to integrate legacy or third party systems (e.g. ERPs and databases such as SAP, Oracle, DB2, SQL Server, etc.) to build complete apps even more quickly.

Low-code is not without its challenges:

- Finding developers: despite the low technical barrier to usage, most low-code systems are proprietary and require at least a modest amount of system-specific training.
- Highly skilled traditional developers often view low-code with skepticism and defensiveness – partly due to the slow devaluation of the traditional skill set.
- Licensing costs are often opaque and somewhat higher than those of traditional development environments and tools (many of which are open-source).

[1] Some industry analysts and vendors of low-code systems use very different terminology, for example: Gartner uses *high-productivity application Platform-as-a-Service* (hpaPaaS).

According to the March 2019 Forrester Wave report, the current leading low-code systems (out of a total of 13 evaluated in this latest report) are Microsoft PowerApps, OutSystems, and Mendix (now acquired by Siemens), with Kony and Salesforce following closely. Similar vendor rankings are also given by Gartner, Ovum, and IDC.

Technical Brief

Briefly, low-code is a development environment and runtime platform that allows for building apps of almost any kind (with the exception of deeply embedded or very high-performance apps) while only writing a very small amount of simple code. In many cases, absolutely no code is written – making them "no code apps."

To the *low code developer* (often someone more specialized in the application domain than in coding), the low-code tool appears like most *integrated development environments* (IDEs). Instead of a window in which to type code, the functionality of the application is built visually. That likely starts with drawing the windows of the application, and attaching actions (again, visually) to the various GUI elements such as buttons, drop-down menus, list boxes, etc. In general, many apps support a *business process,* which is then drawn diagrammatically in low-code. Each step in the business process may activate other windows or take some further action.

Such actions can involve communicating with other applications (such as email), pulling in documents, consulting a database, or taking remote action. Creating an interaction with another application is usually a simple visual connection diagrammatically, and most low-code systems support APIs (which remain only *visual* in low-code) to a vast number of other vendors' applications. Similarly, pulling in documents/files is done visually, along with any processing of such files – including sharing them remotely.

One of the most significant wins in low-code is database integration: the developer drags and drops a database, which is usually then *autodiscovered* by the low-code environment, making the structure of the database visually obvious. The developer may then visually create database queries (in a form of *visual SQL*), and use those results for further processing or display in a window. Naturally, various operations across databases are also supported. The leading vendors of low-code support all of the major databases across many versions. In a limited number of cases, a very complex database query must be written by hand in SQL – though that process is coached by *intelligent assistants* within the tool, and is done visually.

Most real-life app development involves interfacing to legacy systems – something also well enabled in low-code. Such legacy systems (e.g. SAP, some other ERP, or database) appear as *connectors* in a palette of supported systems, which can then be used visually, while the low-code system manages the actual API usage.

All of the low-code systems have extensive palettes of additional components – in many cases written by the low-code vendor in other languages for performance reasons. Some of the pre-built components include:

- Location services using GPS on mobile devices.
- Cloud support for all of the major vendors, as well as private clouds.
- Cameras, including gesture and facial recognition, etc.
- Audio support, including sound generation and voice recognition/synthesis.
- Multilanguage support.
- *Internet-of-Things* (IoT) interfaces and complex event processing.
- Log file and audit trail support in applications requiring governance.
- Security primitives, such as encryption, signatures, and authentication.
- Machine learning and artificial intelligence engines.
- Visualizations for large-scale data.
- Back-end interfaces to big-data systems such as Hadoop.

Low-code tools additionally have user communities contributing new components or wrappers for legacy systems, and proficient programmers (in other languages such as Java, C++, C#, etc.) can easily produce their own components as needed. The leading low-code vendors have extensive support for mobile and desktop apps, including keyboards, styluses, and touch-screens. Much of that support is encapsulated in a single project, meaning that a mobile version of an app can be co-developed with the desktop version.

Low-code environments directly support version control through a variety of interfaces such as to Git, Mercurial, Subversion, etc. This allows not only for the safety of being able to roll back changes, but also for multi-developer projects.

Following the assembly of the app, the low-code environment allows for immediately testing/debugging it without a long compile-build cycle. This directly supports agile development and co-refinement of requirements where domain experts/users can make adjustments to the app as early as possible. Such testing can be done in a staged testing platform with test databases, etc.

Finally, deployment is usually a single-step process in its *lifecycle manager*, where the low-code environment is aware of the deployment platform's parameters and the app can be pushed directly into production and use. Should anything go wrong, rolling back to a previous version is usually equally easy.

Under the hood, low-code is of course also making use of traditional programming languages and tools – though this is not usually visible to the developer. The visual programming is translated directly into C#, Java or similar (some platforms such as OutSystems support both), which is then compiled using the normal development tool-chain. The low-code environment tracks the minimal amount of code generation and recompiles, allowing for a very interactive development experience oriented towards experimentation and prototyping. The output C# or Java can also be used standalone

in case of a move away from low-code, though the actual code is often not elegant enough for true understandability.

Industry Use

All of the major technical industry analysts (Forrester, Gartner, Ovum, and IDC) have reported on low-code for a number of years – following Forrester naming it "low-code" in 2014. Those reports have at times identified up to fifty providers of low-code environments over the past decade. Some of those providers have faded, while three have emerged as the leading providers of very broad low-code solutions[2]: Mendix (now a part of Siemens), Microsoft (with PowerApps), and OutSystems. The Forrester Wave from March 2019 is shown in Figure 1, while the 2018 Gartner Magic Quadrant is shown in Figure 2.

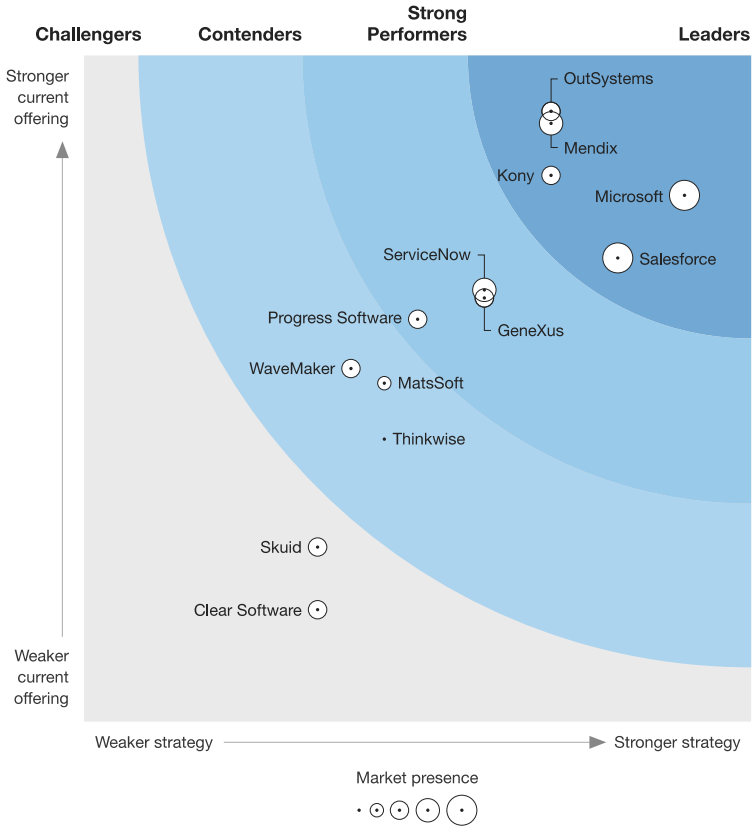

Figure 1 The Forrester Wave: Low-Code Development Platforms for AD&D Professionals, Q1 2019. Used without permission.

[2] A *broad* solution is one able to interface to many other types of components (both legacy or written in other programming languages), runs on several platforms, is able to produce apps of all types.

Among other requirements, the systems featured in the Forrester Wave were chosen because they each:

1. Offer a comprehensive declarative development approach: the level of abstraction matches that of the client, which is critical for expressing the requirements.
2. Provide a low-cost-of-entry commercial model: they allow for free trials, and provide online training material.
3. Support building many business use cases, from web and mobile apps to database, event processing, IoT, and business process apps.
4. Primarily target large enterprises: revenue over a billion USD and geographically dispersed teams.

Websites of the low-code tool leaders typically provide reference customers, and these include numerous prominent banks, insurance companies, airlines, government departments, and the US Army – though in most cases no details are given about the precise application domain. Most of the analysts' reports, as well as self-reporting by OutSystems and Mendix, indicate that 88% of companies are adopting low-code, while 74% of *those* companies are integrating the business side into low-code development, thereby directly involving the clients who dictate the requirements[3].

[3] For example, see https://www.mendix.com/why-developers-should-embrace-low-code/

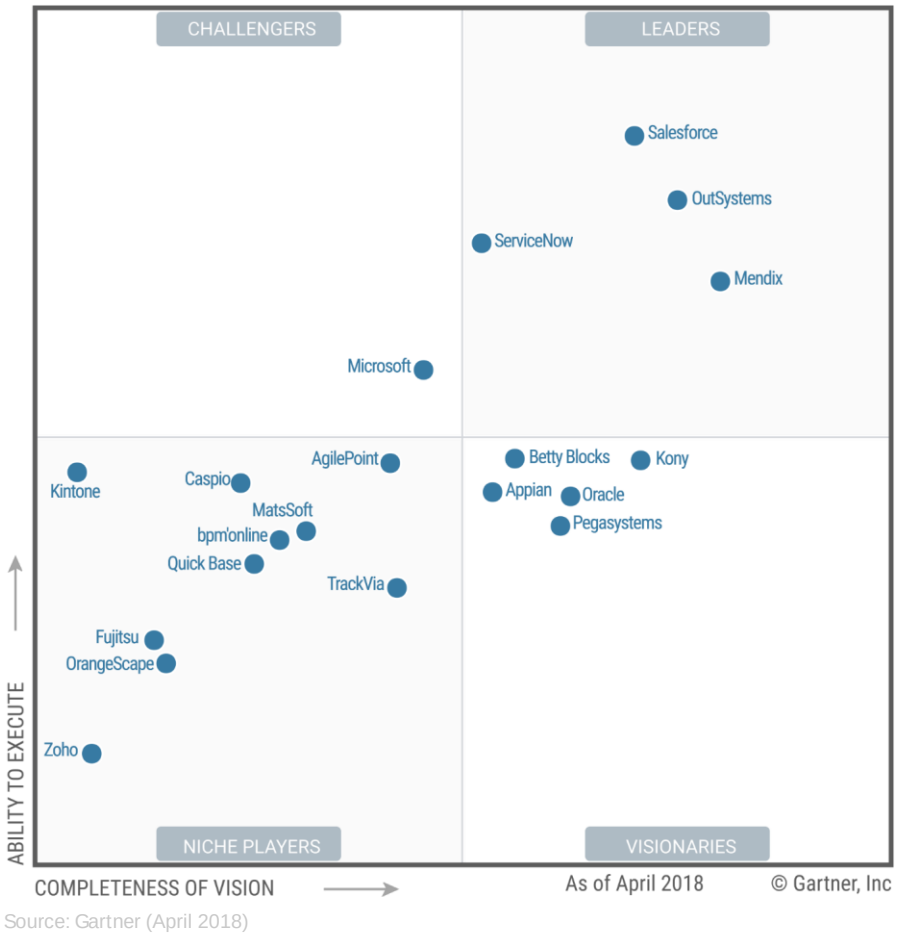

Figure 2 Gartner Magic Quadrant for Enterprise High-Productivity Application Platform-as-a-Service. Used without permission.

Canadian Government Use

Modern low-code can literally be used in *all application areas* except for deeply embedded systems and very high-performance computing, though even in those two areas, low-code can provide much of an app while interfacing to software components written in other programming languages.

Given this breadth, literally *all* IT application areas of the Canadian Government are candidates for low-code. For example:

- Apps consisting of a business process, including interfaces to document/content management, emails for reminders, etc.
- Data gathering and database apps that interface to large scale and distributed data storage – including managing identity and privacy.
- Communication and task management apps that allow for chats, emails, video conferencing, and calendar management.
- Data-science and visualization apps that potentially contain some machine learning or artificial intelligence to process and present large amounts of data.

Analyst group IDG and OutSystems have specifically explored OutSystems for digital government, basing their evaluation on the goals shown in Figure 3, though the conclusions are valid for all low-code systems. In particular, low-code can simultaneously increase quality while decreasing service delivery costs.

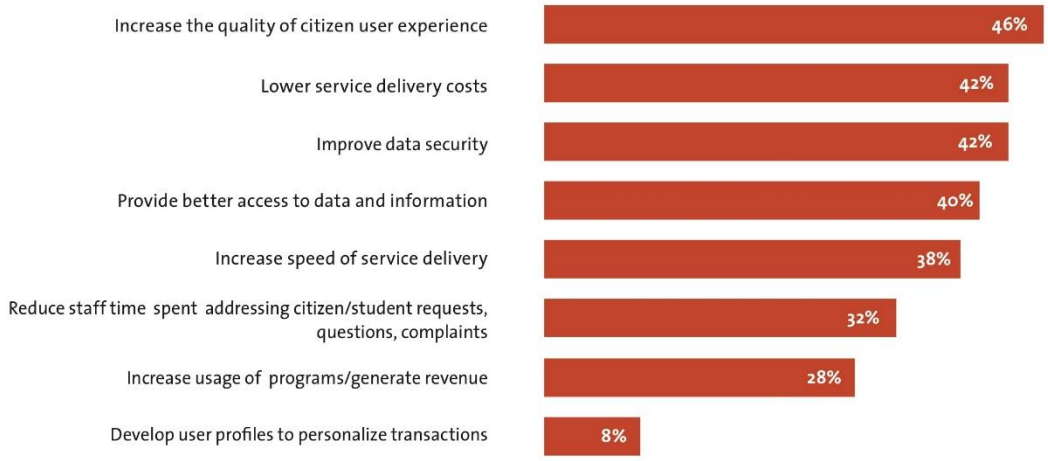

Figure 3 Top Goals with Respect to the Digital Experience Provided for Citizens. Used without permission from: Improving Digital Experience for End Users in the Public Sector, IDG Research Services, December 2018.

It is worth noting that achieving these goals (likely via low-code) fits into the higher levels of Gartner's Digital Government Maturity Model (such as Level 4 and Level 5).

Implications for Shared Services Canada (SSC)

Value Proposition

Given SSC's heavy involvement in IT services and management, the value proposition of low-code relies on the production, deployment and maintenance of apps. Low-code moves app development to higher abstract levels (such as directly modeling business processes, etc.) and requires far less code to be written, giving the following specific value propositions:

1. Shorter time-to-market, giving higher client satisfaction, more accurate schedules, and also less temptation to build workarounds or purchase ill-fitting temporary solutions.
2. Fewer person-hours of development, giving lower cost.
3. Broadening the pool of potential app developers, alleviating staffing pressure, and potentially lower costs while shortening development timelines.
4. Direct integration of legacy systems, allowing for incremental roll-out and use of low-code for cost and risk management.
5. More accurately capturing and building for customer requirements, leading to lower rework and maintenance costs.

Challenges

While low-code vendors pitch it as a solution to *all app projects*, it is not without its challenges and issues, all of which are relevant to SSC:

- The nature of low-code is to allow for non-coders (ideally, domain experts for the client) to create apps, however, low-code environments are largely proprietary and require at least a modest amount of training. This applies even to highly skilled developers.
- Most organizations experience some pushback from traditional developers, partly due to skepticism, though often due to defensiveness as more (and cheaper) people become "developers."
- Most traditional development environments involve a one-off purchase of the IDE, however, low-code licenses are considerably more complex, often involving per-deployed-app costs. Those costs are sometimes opaque and need to be fully worked through for the ROI analysis.
- There are several low-code systems to choose from, though the current leaders (which include Mendix, Microsoft and OutSystems) have consistently been ahead of the pack[4]. Which platforms to choose is a challenge that depends

[4] That is not quite true for Microsoft, which is relatively new to low-code, though their development environments and general platforms are very mature and strategic.

heavily on legacy software to be supported, existing deployment platforms, etc. Supporting more than one low-code platform will bring its own challenges.

- The proprietary nature of low-code gives a certain level of "lock-in," preventing SSC from changing vendor or leaving low-code altogether. Vendors usually portray low-code as having no lock-in because C# or Java code is generated, which may be maintained further without low-code. In practice, this is not true for large apps.
- Low-code's ease ("democratization") of app development can cause problems for SSC: clients will be tempted to develop on their own if SSC is not fast enough, thereby creating a *shadow IT (hidden/skunk works)* mentality. If allowed to grow, this will create maintenance issues as well as an existential problem for SSC.
- Security has been the chronic weak link in low-code, and vendors are only now attending to it in a structural way. Security primitives have long been available in low-code development palettes, but their use has not been obligatory and they are not interwoven with the environment.

Considerations

In choosing whether, how, or when to move to low-code, SSC has numerous considerations. SSC would obviously gain from the low-code efficiency improvement, and the main considerations revolve around when, how, and on what scale to move to low-code. As clients become aware of SSC's low-code plans, they will exert increasing pressure to shorten timelines/schedules as well as lower project costs. SSC will need to consider even more agility in client interactions, as well as involving them more directly in projects using low-code – all while SSC retains direct control over app development. Regardless of how fast SSC adopts low-code, there is a risk that its popularity is picked up by forward-looking (or desperate) clients that initiate Shadow IT low-code projects. Completely preventing such projects will be difficult, and SSC must consider how to bring such projects "in from the cold," to be integrated and supported from inside SSC.

Rolling out low-code in SSC also requires some other considerations. Despite its simplicity, low-code will require at least a modest amount of developer training. This would have to be coupled with change management to integrate and not alienate traditional developers into the move; not doing so risks creating a two-tier developer corps, or having traditional developers actively working against the low-coders. SSC will need to consider a sequence of projects appropriate for low-code – whether based on risk-management/criticality or potential for savings. The security needs of SSC are somewhat unique (thanks to the broad cross section of clients), and ensuring low-code adequately supports security is a challenge needing exploration.

SSC should consider a low-code trial for several reasons: gaining exposure, evaluating low-code tools, and quantifying the ROI (at least for one project). Ideally, all of the leading low-code systems should be evaluated – though the field may be thinned (e.g. to Microsoft PowerApps or OutSystems) based on breadth of the system as well as the match to existing technologies in-use by SSC. During the trial, several aspects should be

measured: training cost (which is then translated to *amortized training costs* as they would be across several projects), amortized licensing costs, elapsed time, client satisfaction, and amortized maintenance costs.

References

- "The Forrester Wave: Low-Code Development Platforms for AD&D Professionals, Q1 2019." Forrester. https://reprints.forrester.com/#/assets/2/160/RES144387/reports
- "Magic Quadrant for Enterprise High-Productivity Application Platform as a Service." Gartner. https://www.gartner.com/doc/reprints?id=1-4XVPI4N&ct=180430&st=sb
- "Ovum Decision Matrix: Selecting an Enterprise Mobile Application Development Platform." Ovum, 13 April 2018.
- "What Is Low-Code." OutSystems. https://www.outsystems.com/blog/what-is-low-code.html
- "App dev has evolved: Why developers need to embrace Low-Code." Mendix. https://www.mendix.com/why-developers-should-embrace-low-code/
- "Low-Code Guide." Appian. https://www.appian.com/assets/sites/14/2016/12/low-code-guide.pdf
- "Low-code development platform." https://en.wikipedia.org/wiki/Low-code_development_platform
- "App dev has evolved." Mendix. https://www.mendix.com/why-developers-should-embrace-low-code/
- "Accelerating Digital Government With Low-Code Development." IDG sponsored by OutSystems, December 2018.
- "Introducing the Gartner Digital Government Maturity Model 2.0." Gartner. https://www.gartner.com/doc/reprints?id=1-667LEM1&ct=190130&st=sb
- "The Best Low-Code Development Platforms for 2019." https://www.pcmag.com/roundup/353252/the-best-low-code-development-platforms