Does a Low-Code Approach Improve Your Total Cost of Ownership?

Authors:
Ken Baldwin—PLM Systems/ Configuration Specialist
Erika Marwood—Sales Engineer, Micro Focus | Serena

Low-code platforms are touted to provide key savings of rapid development and improved response to business needs. These are clearly value elements of the return on investment; however, viewing your development investment in terms of total cost of ownership (TCO), Low-Code has even greater potentials.

It is not uncommon for maintenance cost to equal or in some cases exceed the total initial development costs over the life of a tool. Low-Code greatly improves the maintenance of your development through organization of functional elements and their related data elements. This serves to normalize styles of coding, while improving documentation of functions through the actual coding elements of the interface.

This paper will use implementation examples of Micro Focus Solutions Business Manager (SBM) with articulated benefits of Low-Code in business tool development and maintenance. It will provide extended insight into the continued benefits of standardization that Low-Code inherently brings into the equation for total cost of ownership (TCO).
Table of Contents

Disclaimer .................................................................................................................. 1
Low-Code and Custom Application Development ................................................. 1
Example: .................................................................................................................. 5
Applying the Math .................................................................................................... 6
Applying SBM .......................................................................................................... 8
Conclusion ............................................................................................................... 17
Disclaimer

This paper recognizes the complexity of TCO involved in information technology tools, which includes a web of complex interconnection of hardware, software and data components dependent on magnitude sophistication that contribute to the total cost, each having a variety of product pricing structure owned or contracted services. This paper is focused on the specific cost of application software maintenance due to change and more precisely the Change Tools element of Low-Code.

Low-Code and Custom Application Development

In business today, application development and deliver (AD&D) plays a critical role in maintaining a leading edge by leveraging business resources. This is especially true in businesses involving complex problems that require multiple teams focusing on shared data. In the aerospace industry this is precisely the case regarding the design that set the World Standard in Business Jet Aviation.1

An aircraft is a large collection of complex systems that must work in perfect concert to constantly carry out a flight mission, safely and efficiently. Engineering dynamics in the aircraft design process require an extremely responsive application that adapt to manage new information which evolve as an aircraft design matures.

In this fast moving type of environment it is not unusual for application requirement to consume design, development and testing time that exceeds the solutions benefit’s window. Quite often these application development projects never progress past a proposal and are replaced with a costly manual effort to achieve some portion of the benefit they would have contributed.

Low–Code has proven itself in resolving this time lag between requirements and deployed solution. It is no surprise that more and more of the application tool include varying level of Low-Code.

In 2014 Forester Research, Inc. coined the term ‘Low-Code’ platforms as:

“Platforms that enable rapid delivery of business applications with a minimum of hand-coding and a minimal upfront investment in setup, training and development.”

---

1 The World Standard is a registered Trademark of Gulfstream (Registration Number: 2407571)
Declarative applications have evolved over several decades in many forms, as the family of Database query languages such as SQL, to a level where development focuses more on defining what is required and less on how to code it. This combined with significant advancement in graphical user interface (GUI) has matured this evolution to a point where a user with little or no programming skills may simply drag-and-drop their requirements to, in fact, code their application change through tools that embodies Low-Code.

Forrester is not the only place where the term “low-code” is gaining traction. Paul Rubens, an online journalist, also references “low-code” stating the following: “Low-code, rapid development platforms provide a way to incorporate user feedback into apps during development. This improves the turnaround time for consumer-facing applications while ensuring that projects don’t turn into white elephants.”

Why are low-code platforms important to businesses? In 2016, Chris Anderson published a book talking about how Amazon applied the long tail ranking, discovered by George Kingsley Zipf, to their business methodology. Instead of going after the large bookstore sellers, they were focusing their efforts on the niche online vendors. As their current stock price indicates, it was not a bad model to follow. This same “long tail” applies to application development. Within any organization, there are different types of application development that occurs. There are the applications that bring in revenue for the company and that is where the majority of the development efforts and resources are focused, which makes sense. There are the supporting departments though that require applications do perform their activities as well. In some cases there are off the shelf products, but in many cases there are not. So those departments start to rely on applications such as Microsoft SharePoint lists, Microsoft Excel spreadsheets, Microsoft Access databases and email to manage their processes. Where this may come to a breaking point when an audit trail is required or you need to maintain historical data and produce reports with accurate federated master source based data (not data that is stale the moment you save the image into a presentation). If there are business drivers such as governmental fines or the company’s reputation would be on the line if processes are not followed and audited then those business processes may gain priority to have those development resources build them a custom application. This custom application development comes at a price though, not only for the developers but for the business as well.

Paul Rubens, an online journalist, also references “low-code” stating the following: “Low-code, rapid development platforms provide a way to incorporate user feedback into apps during development. This improves the turnaround time for consumer-facing applications while ensuring that projects don’t turn into white elephants.”

3 The Long Tail, Chris Anderson, Hyperion, July 11, 2016
It is an accepted notion in traditional (hand-coded) AD&D that each time you apply an enhancement some percentage of that effort is required for maintenance due to impacts precipitated by the enhancement change on existing code. Robert L. Glass in his book, “Facts and Fallacies of Software Engineering” presents Fact 41, which introduces the 60/60 Rule “Maintenance typically consumes 40 to 80 percent (average, 60 percent) of software costs. Therefore, it is probably the most important life cycle phase of software.” To simplify this rule into a graphic representation we offer the following graphic.

![Software Maintenance Graph](https://via.placeholder.com/150)

Simply put, on an average the total cost of AD&D (under traditional hand-coded methods) 40–80% (averaging 60%) of the total cost is contributed from maintaining existing code, so of that 60%, 60% (36% of the total) can be attributed to maintenance due to the changes made to apply enhancements.

From the business’ perspective, now that they have a custom application, they get in line when they want those new enhancements behind the development of the revenue generating applications again.

This mega-paradigm in and of itself is compelling business to take advantage of Low-Code’s significant changes in the economic power of AD&D, however when calculating the total cost of ownership (TCO) on AD&D, Low-Code presents an even greater economic advantage. From this it can be applied that if Low-Code eliminates hand-coding, it would eliminate this contribution to software maintenance in the total TCO of the application.

**Does Low-Code truly eliminate the maintenance due to enhancement cost?**

**What are the factors contributing to this reduction of cost from the equation?**

To answer these questions we must explore how Low-Code is developed and what results from the fundamental factors of that development process.
Low-Code has evolved since the beginning of user interface (UI). One of the most common examples to we all used is web page configuration. If you consider a dynamic form change tools commonly found on web applications, such as a payment page. The code development involved adapting the page based on the user’s input selections and potentially any prior account information to customizing payment type and information required for the transaction. Quite literally, the web application code is transforming the page based on the user’s selections. Of course, all of these options are predetermined to be controlled by the source code. However these options may be altered by the developer within a modular structure, which enables each element’s behavior to be reconfigured to a conditional value.

To develop such a tool for a new website would require significant coding to develop this element’s integration with several account functions and multiple data sources. We tend to take this for granted; however the application developer must address and test all these integrations to provide consistencies and reliability throughout the application. At the platform level this same function may appear in a multitude of related applications, providing consistency and intuitive ease of use.

It is all about planning of functional standardization. Before a platform developer can begin developing Low-Code they must clearly define what is required. So the better they research the business market the better their Low-Code with support a broad market benefit. Once they have the requirements well defined, they plan the back-end schema for efficient management of the data. If they have succeeded with their development the result is a development of modules of code objects as standard building blocks required for building the required business solutions.

The goal of Low-Code is to develop a robust set of this standard objects that are capable of working independent of hand-coding and are designed in as refined system of components that are completely compatible within that application platform. These objects are then represented in the GUI for intuitive assembly by a user familiar with the business application needs, but without the requirement of coding skills.

The jewel in the most successful achievement of this lofty task of Change Tool development is an advanced layer of verification that not only report potential errors in the user’s assembly of design components and their supporting configuration setting, but guides the User with easy to understand explanation integrated with links to the GUI representation of the touch-point where changes need to be applied.
The level of savings in the maintenance element of TCO is directly attributed to the success of the platform developer in achieving a minimal-level user verification, beyond the GUI options guiding the User’s initial choices to appropriate assembly and configuration of the Change Tool object, as well as an intuitive direction of post verification adjustments.

Without the platform developer’s success in these areas, developers must write custom code to achieve the desired functions, introducing a risk of losing this potential low code saving in the maintenance factors of TCO.

As an example of how these saving can benefit your organization we offer the following example where these saving apply.

**Example:**

Solutions Business Manager (SBM) has these characteristics of: Change Tool objects; advanced verification tools and in the following example we will outline these elements of potential maintenance element of TCO saving. This single example is typical of our finding over multiple projects of varying degrees of complexity, with respect to other applications.

The example applied here is a typical process database application, involving several common business process tools:

- **Database Tables** (Primary Record Table, Field Configuration, Systems Control Tables,)
- **Process Elements** (Submittal, Transition, States)
- **Notification Elements** (Trigger Conditions, Email Actions, Escalation Events, Termination Conditions)
- **Graphical Reporting** (Field Selection, Filter Application, Calculated Field, Graphing Options)
- **Web based user interface** (HTLM5 web format IO)

The developer ROM (rough order of magnitude) for this project was 1006 hours of coding and test. To put this in perspective this is six months of a dedicated developer. One challenge in the math of calculating relating hours to code development is a standard conversion factor.
Glass⁵ offers a discussion on the options, LOC (lines of code), FP (function points), FP (feature points), which are to this day in debate. We applied the expert average method (ask several people skilled in the art of coding to apply their best method of estimation and average their result), which we would suggest for the best estimation of expected result.

**Applying the Math**

If we figure this application initial development estimate of coding and test at 1006 hours is equal to the combination of the initial enhancement (23%) and the repair of flaws (17%) in the TCO or 40% TCO. The other 60% portion of TCO of software is equal to 1509 hours making TCO = 2515 hours

\[
1006 \text{ hours} = 40\% \text{ (TCO)} \cdot 60\% \text{ (TCO)} = 1509 \text{ hours} \cdot 100\% \text{ (TCO)} = 2515 \text{ hours}
\]

To calculate change (enhancements over time) you will need to look at the consistency of your business.

These changes tends to be additive record data capture and reporting as management focus shifts due to the business dynamics, but they may include organizational adjustments, resulting from resource leveling as the business evolves and resource demands shift.

Based on experience in aircraft design business the churn is highest (5-10%/Qtr) at the start (first few quarters) of a program tapering off to a lower (1-5%/Qtr) rate as program completion approaches, but over the course of a 5 year software live-cycle this can equate to 25-50% of the original application development.

---

⁵ Facts and Fallacies of Software Engineering, Robert L. Glass, Addison Wesley, October 25, 2002, Fact 8 (Estimation)
Applying this change factor to our example project we can expect the following software TCO:

\[
(0.25(1006) = 251 \text{ hours}) + (1006 \text{ original enhancement 40% TCO}) = 1257 \text{ (Total enhancement of 5 yrs)}
\]

\[
(1257 \text{ hours} = 40\% \text{ (TCO}) : : 60\% \text{ (TCO)} = 1886 \text{ hours}) : : 100\% \text{ (TCO)} = 3143 \text{ hours}
\]

Of this TCO 3143 hours, according to Robert Glass, 36% of this TCO can be attributed to maintenance due to enhancements (MdtE)

\[
3143 \text{ hours} * 36\% \text{ (MdtE)} = 1131 \text{ hours (over 5 years)}
\]

It is striking that this maintenance due to enhancements is greater than the original software development investment. Can it be claimed Low-Code truly eliminates the need for added maintenance cost due to enhancements? Realistically there will be MdtE resource applied even with an excellent advanced verification tool set. We offer the following 5% factor, again based on experience with SBM of verification reconfiguration adjustments to the calculation to level this comparison

\[
0.95(1131) = 1074 \text{ hours saved over 5 years}
\]

The answer to the question posed earlier: **Does Low-Code truly eliminate the maintenance due to enhancement cost?** Yes! If all of the components needed for your application come with the platform and you completely build your app with the low-code components, then you will have an extreme reduction in the maintenance cost.
Applying SBM

When utilizing the Micro Focus® low code platform, Solutions Business Manager, to the above example we can see how the maintenance due to enhancement cost is reduced.

1. Code Standardization
   a. Rather than opening a text editor and writing JavaScript to provide dynamic form capabilities and requiring commenting so the next developer knows what was done, SBM provides a GUI to configure the form logic, as seen in Figures 1 and 2.

---

Fig. 1

SBM Action Editor, User Conditional logic for GUI Form Change
2. UI/Mobile design and definition

a. SBM provides a drag-and-drop interface that allows you to only include elements on the form that you have defined as part of the application. You are not required to type `<div>` elements or have complete HTML or mobile application knowledge in order to create a business user friendly user interface.
3. For custom applications you must define your database schema and the design of your system tables, selection values, etc. SBM already provides this infrastructure when you start your first process, so you simply need to start to define the data you want to capture in your business process.

Fig. 4

Out of the box SBM application database schema
4. Many home grown solutions and band aid solutions (built upon applications or platforms such as, Microsoft Excel, SharePoint, etc.) provide a way to capture lists, but not with an automated audit trail. SBM’s State Change History and detailed audit trail are “always on” and available for viewing by permissioned users.

---

**Fig. 5**

SBM record State Change History and details from record history log
5. Pro-active, reactive and escalation notifications are usually very minimal in custom developed applications and take time to integrate with existing email systems. SBM provides this capability as a platform feature that provides extensive flexibility in rules and email format and recipients.

![SR | Service Request Resolved](image)

**SR | Service Request Resolved**

- **Request Id:** REQ_223644
- **Title:** Software Issues: Expired Temp license for HP QC/ALM used in SA Demo Image
- **Description:** The temporary license the SAs have been using for the HP QC environment expired on July 31. As luck would have it, I have a demo on Monday that I am prepping for today. I need to obtain a new temp license ASAP
- **Resolution:** I have requested and received an updated license file from HP which is valid for six months from the date received. I have updated the software inventory to reflect the new license information and email the new license file directly to Erika.

The technician assigned to your ticket has marked it as resolved. Please confirm that the ticket is resolved to your satisfaction by clicking the link below and then clicking the “Close” button within your ticket.

**IMPORTANT:** This notification will repeat every 48 hours until the ticket has been closed.

**Please click the link below to view the details of this incident:**

**Fig. 6**

SBM Email (Notification w/imbedded record data)
6. Real Time Reporting can be included as a Low-Code implementation opportunity for going beyond listing type format, to provide advanced graphics reporting, without requiring manual or custom coding alternatives to integrate Microsoft Excel or some other external reporting application. From an SBM perspective, the developers time is spent building the application and then the report definition is put in the hands of the end users with the reporting packing included in the platform. Real time reports that can be viewed or scheduled to be sent to users are easy to configure and provide a wide array of formatting options.

Fig. 7

SBM Kanban Board Project Display
Fig. 8

SBM Calendar View Project Display
Fig. 9

SBM Dashboard View
Low-Code can also serve in seamless delivery to Multi-platforms without additional development effort from the business. The SBM platform form design is leveraged in both the web user interface as well as for the mobile interface that is compatible on both iOS and Android operating systems, through the same Low-Code interface.

**Fig. 10**

Mobile Device View of SBM Content
The last area, but perhaps the most important is the hidden jewel! The **advanced layer of verification**. Since the Low-Code solution consists of predetermined functional elements which are designed to work together, the error messaging related to their assembly (to address the requirements of business logic) may also be predetermined. In addition, intuitive error messaging provides the user an opportunity to resolve these issues with minimal impact to integration testing cycles.

**Fig. 11**

SBM Validation Messaging

### Conclusion

Coupling Business Knowledge to Business Tools

+ Rapid Software Development Turnaround

+ Near Elimination of MdI/E

= **Low-Code is destined to become a bigger part of the future!**