Xsemble Positioning

Against Other Low Code / No Code Platforms

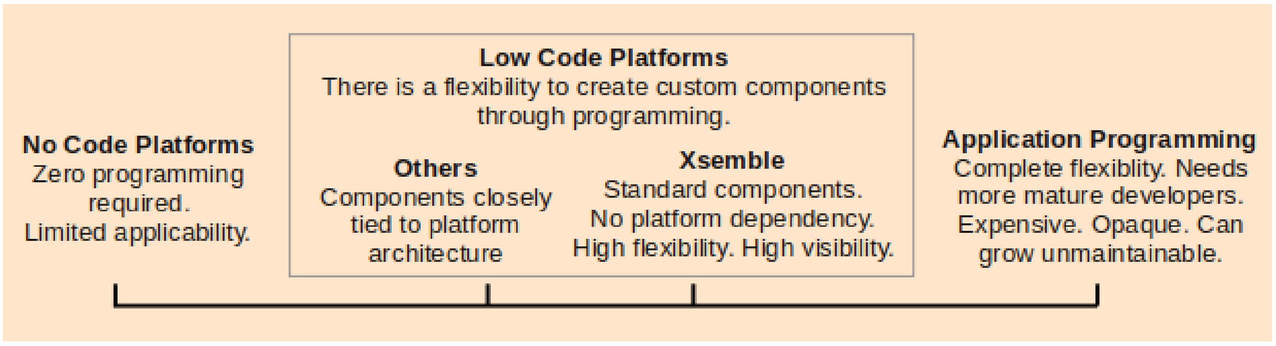
Ashish Belagali
10Xofy
June 2019

No Code Platforms / Low Code Platforms is a hot area, so much so that Gartner and Forrestor both have taken a keen interest in it in recent years. The no code / low code paradigm is a natural extension of Component Based Development (CBD). At the core of an application, there are components which are designed to work independently of one another, and hence can be arranged in various ways as per the application needs. Instead of relying on the developers' maturity and discipline, these platforms provide the software infrastructure that orchestrates the working of components. Going by the definition of the term, Xsemble is a low code platform. However, Xsemble differs from others in some significant ways.

The other low code platforms such as Outsystems, Mendix, Appian and Koni also differ from one another in their capabilities, approach and appeal. For instance, Mendix has a collaboration system inbuilt into their platform while others do not have it (so users can make use of tools like JIRA). As another instance, Appian seems to be doing a niche market positioning by providing components and examples targeted towards the niche market. However, Xsemble is more different at a fundamental level from all of them. This article is to articulate those differences, to help the reader understand Xsemble's unique value proposition.

We will deliberately not compare factors such as component availability and the market positioning, as these keep changing with time. Instead, we shall look at the fundamental differences.

Positioning Based on Flexibility


No Code Platforms
Zero programming required.
Limited applicability.
Low Code Platforms
There is a flexibility to create custom components through programming.
Others
Components closely tied to platform architecture
Xsemble
Standard components.
No platform dependency.
High flexibility. High visibility.
Application Programming
Complete flexiblity. Needs more mature developers.
Expensive. Opaque. Can grow unmaintainable.

In order to understand this further, let's elaborate on these terms further:

1. The full-fledged application programming is the conventional way of creating complete applications through programming, and ranks highest on the flexibility programmers have. However, this flexibility is often abused. In practice, the programs deviate from design, degrade in quality, grow complex and become unmaintainable quickly. For robust and long lasting programs, certain discipline is needed at every step; and low code platforms is a great way to impose it.

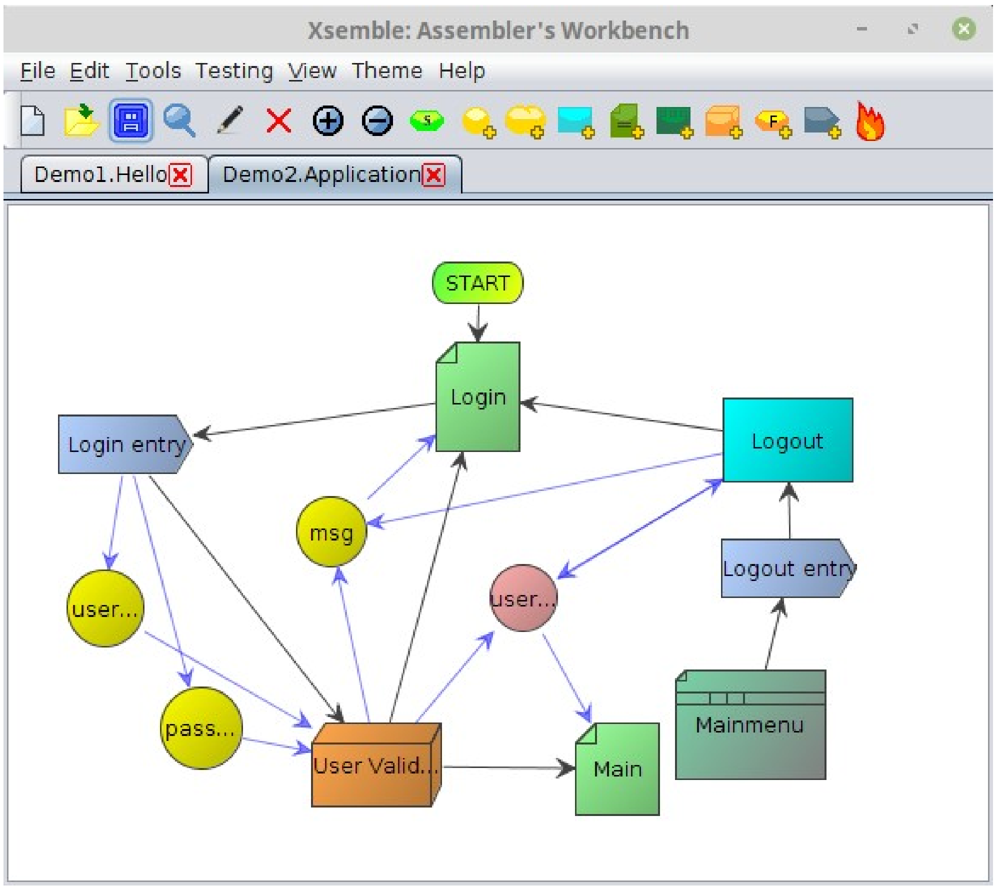
2. The no code platforms are a drag-and-drop environments where a non-programming end user can organize pre-existing components. The low code platforms provide further flexibility by letting the users create custom components. Low code platforms are therefore more popular and the no code platforms are seen as restrictive in capability.

3. The components are closely tied to the platform architecture. In other environments, the implementations are dependent on platform classes, and there is a learning curve for that. In case of Xsemble, there is no platform dependency within the component implementations. Therefore, Xsemble keeps the way open to reuse the same components within another backend framework. Programmers would realize that this is a big plus.
4. Within an Xsemble component implementation, a programmer is very much within the familiar programming world and can use all the rich constructs that have evolved and matured in programming. Xsemble goes great length to help developers define component dependencies (typically on various files), and making their reuse painless. Thus, Xsemble takes the best of both worlds: on one side, the convenience of organizing components and the full blown programming wisdom, on the other.

Visual Flow Diagram

The entire application is a flow, as per Xsemble. (The internal name for Xsemble is Method Graph Framework, which is derived from the fact that Xsemble conceptualizes an application as a large flow consisting of interconnected nodes, whose behavior is akin to Methods as found in programming languages.) One may liken this flow with flowcharts or activity diagrams, with one major difference that while the flowcharts or activity diagrams capture only the control flow, the Xsemble flow diagram captures both the control flow and the data flow.

For better human comprehension, the flow may further be broken into a project hierarchy containing projects and subprojects. The Xsemble technology is innovative in these respects.



On the other hand, the other low code platforms have UI editors and then there are several microflows (that correspond to the processing logic) which are invoked at an event, such as a button click. These microflows are disjoint, and unable to help one capture the complete flow of the application. Further, they capture only the control flow.

Xsemble flows are, then, much more expressive (and colorful and prettier too!). Such visuals are extremely useful to keep all the stakeholders on the same page and do brainstorming.

In other low code platforms, the web pages and processing units are completely disparate. Outsystems has even two separate editors for the two. On the UI side, these frameworks work on an underlying technology such as React or Xamarin.

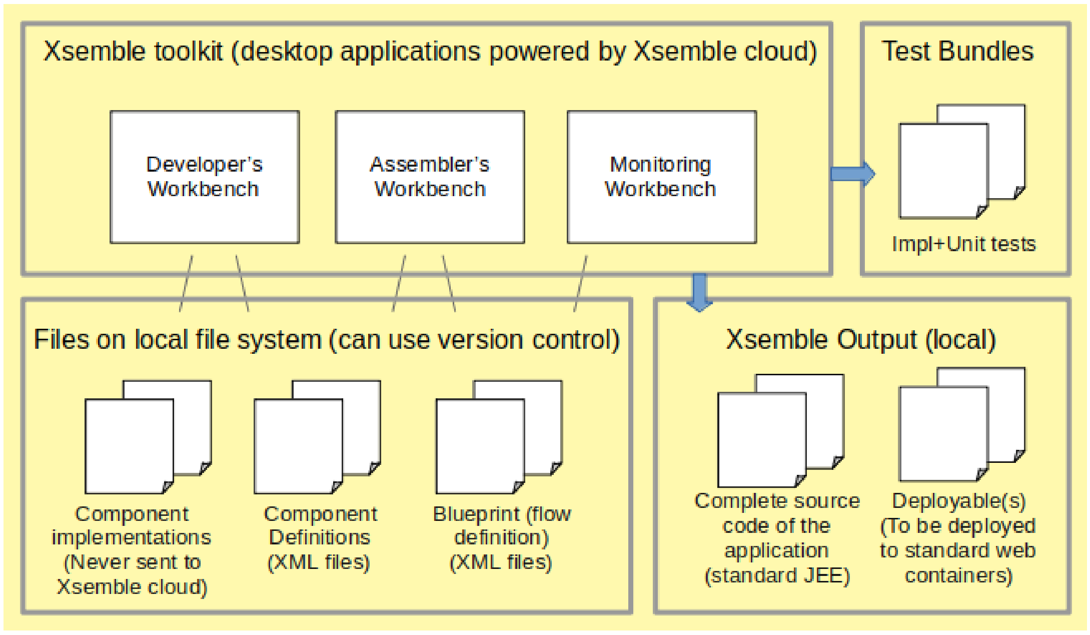
In contrast to that, Xsemble does not have its own integrated editor. This means that users can to use a tool of their choice, such as Eclipse (for processing blocks) or Maqetta (for UI). Further, Xsemble does not prescribe any UI side framework, and the programmers are free to choose what they like. In other words, there is complete flexibility. Same can be said about the database connectivity mechanism and several other aspects of programming. Because of the flexibility, the best practices in programming world can be better utilized with Xsemble. For instance, programmers have the choice to use “connection-per-request” pattern and/or a connection pool while connecting to the database so that they can optimize on the connections. Also, Xsemble is careful about not introducing low-performance constructs (such as reflection), so that Xsemble based programs can keep running optimally.

Artefact Organization

The components, the code that runs behind them and their interconnections are the artefacts that are needed to be maintained somewhere, by any low code / no code platform. Organizations will see a major advantage in the way Xsemble manages the artefacts, as against other low code platforms.

The other platforms are web based, and all the artefacts are managed on the server side, which means that the users would not get to access them. (Mendix has a desktop based IDE which is a full-featured alternative to its web equivalent, but you have to check out the project into desktop based IDE when the data is pulled from the web, and then it has to be checked into the web side for it to be available for further processing.)

Xsemble toolkit, on the other hand, has desktop applications which manage these artefacts locally. The figure below is a schematic which shows Xsemble’s way of organizing artefacts, and can aid in understanding the differences.


Xsemble toolkit (desktop applications powered by Xsemble cloud)
Developer's Workbench
Assembler's Workbench
Monitoring Workbench
Test Bundles
Impl+Unit tests
Files on local file system (can use version control)
Component implementations (Never sent to Xsemble cloud)
Component Definitions (XML files)
Blueprint (flow definition) (XML files)
Xsemble Output (local)
Complete source code of the application (standard JEE)
Deployable(s) (To be deployed to standard web containers)

Consider the following points:

1. These artefacts are files on the local file system, and they can be version controlled using common and sophisticated version control systems such as git. Note that the component definitions as well as the flow definition are persisted into plain XML files, and a version control system works great to manage the differences between any two subsequent versions.
2. The desktop applications take help of Xsemble cloud for generating code and instructions. In that process, the definitions of related components and the relevant workflow nodes is sent to Xsemble cloud. The Xsemble cloud caches them only as long as they are needed and clears them when the job is done. In any case, the implementation code is NOT sent to Xsemble cloud. This means that an organization that has concerns around sharing proprietary algorithms on cloud can safely use Xsemble. The same cannot be said about other low code platforms.
3. The output of the *burn* process in Xsemble (which is called as Generate application in other platforms) is the complete source code of the application. This source code is standard Java JEE code – along with the build scripts – and is sufficient to create the deployable using standard Java process without needing Xsemble. (Xsemble can optionally deliver the deployable by firing the same build.) This means that an organization has a choice to take this source code and run with it further without using Xsemble beyond that point. While this is not a good choice, having this choice means that there is no vendor lock in.
4. The components created for Xsemble do not use any Xsemble specific classes (barring some advanced and rare situations). This means that the programmers' learning curve to create Xsemble components is much smaller than the other low code platforms, where you need to learn the working of internal classes of these platforms.
5. Xsemble has a unique functionality to export out test bundles. These test bundles comprise of implementations of chosen components and their unit tests (auto generated templates if not present). These test bundles could be zipped and shared with a programmer / test engineer so that the component implementations and/ or unit tests can be created / enhanced. The test bundles can be opened up in standard Java IDEs as projects. They contain all the relevant dependencies (supporting files etc) and do not need Xsemble. Once the changes to the test bundles are complete, they could be returned to the sender and then the changes incorporated into the component repository. This is a good way to outsource the development of components and/or their unit test cases.