Carson Smith

Jiajia Gu

RHET 1302.011

2 December 2024

Position Paper

As the world of Information Systems (IS) and Information Technology (IT—the two collectively referred to hereafter as "IT/IS") expands and takes on new types of roles, responsibilities, and projects, the market for outsourcing these activities grows and evolves too. Gone are the days where "the IT department" was just the nerdy guys in some closet tucked away in the office who got your email to work again. A modern IT department is expected to not only support an organization's technology, but also carry out Business Analytics (BA), Business Intelligence (BI), and Digital Transformation. With so many new roles, IT departments, and even directors and executives above these departments, often look outside the organization, where they encounter low-code and no-code products (commonly referred to as just "low-code.") These low-code products offer tempting, "easy," solutions to pressing problems. Low-code products allow non-technical or minimally technical people to create entire applications with minimal code writing, most of it being generated. Instead, the user creates the app through a graphical, oftentimes drag-and-drop process, which is deceptively easy. Some proponents of low-code say this is for the better: it liberates software development to the non-technical masses, so that business leaders can focus on other important aspects of their product and their strategy (Owens). However, other say these products are often too good to be true—they typically invite more problems than they solve (Little). Between these two stances, I would argue that low-code is

plagued with issues when used as freely as custom code. Some of the worst offenders related to low-code products are vendor lock-in, a lack of scalability, and challenges integrating the low-code product with the rest of the codebase. These problems often lead to more man hours, costs, and issues than that which would've been needed for a custom coded solution.

Vendor lock-in is an issue that occurs when an organization becomes overly reliant on product that is typically provided on a subscription basis. This issue can apply to many different fields, but in tech it is especially prevalent because of over-reliance on Software as a Service (SaaS) and Platform as a Service (PaaS) product models. These products are usually cloud-based software products, such as simplified development environments or simply pre-built applications, that are only made accessible through a subscription—some common examples of this business model are Microsoft Office or Adobe product suites. This creates an issue where the organization cannot break from the product (or the "vendor") without significant work being done and without accruing significant cost in time and money to transition the business away from the vendor (Forbes). Normally, this is not an issue if the product is reliable and well-made, future-proof, and workable—like the previously mentioned software examples. But low-code software typically does not consider scaling and workability in its design principles (Klezcz). As a result, when a business inevitably has to transition away from software that was very likely doomed to fail, it results in unnecessary reworking and remaking of product (Levinson 17). Even if the SaaS or PaaS solution works in the long-term, the longer a business works with and develops on said solution, the deeper their lock-in gets—that is to say, as a business builds itself around a subscription service, removing that service will only get harder the longer the business waits to do so. For many, this effectively puts businesses at the whims of the vendors, who make them choose between dedicating man hours and funding towards transitioning away, or putting

up with issues like downtime, price fluctuations, and third-party support personnel. Vendor lock-in is a real issue, both inside and outside IT, and only exacerbates other issues that low-code brings.

Another critical pitfall is the lack of scalability in low-code applications. Scalability is a required feature of production applications—it's the application's systems' ability to change in deployment scale depending on the load or demand for that application. When it comes to low-code applications, especially those built on PaaS solutions, the scale of deployment is usually not in control of the business, and instead is automated to some extent by the platform. This is a fundamental limitation that acts as a barrier for any low-code application to be used in a development environment (Klezcz). For this reason, low-code can see some benefit in prototyping and early development stages, such as for testing a concept, an MVP (minimum viable product), or a feature. However, most PaaS platforms cannot handle deployment stress beyond a certain point, and failing to consider this pitfall may leave the business struggling with availability issues if the computing behind the application gets overwhelmed. When this issue is encountered, there's not much that can be done to get out of it, besides drop the development platform, but that comes with the pitfalls of vendor lock-in, meaning that for a production release, the application should undoubtedly be developed as custom code with a scalable back-end.

Lastly, low-code often comes with tough integration challenges, as without an open source code, efforts to integrate a low-code front-end or back-end with existing systems and codebases can be very difficult. Most notably, low-code systems often generate poor-quality, not human-readable, not easily to modify database—or backend— code, according to Nick Scialli, a senior software engineer at Microsoft. This poses an issue to developers on the existing

codebase—how do they integrate a database that is made of "spaghetti-code," written in a way that they can't read or understand how it works, and in a way that violates much of the company's standard practices when programming. Integrating a low-code product with a custom coded database is very difficult—nearly impossible. Moreso, the Department of Health and Human Services faced challenges, and eventually delays, when trying to link the custom coded existing code by DHHS with the low-code generated code from one of their contractors, resulting in the Healthcare Marketplace website being down intermittently for months while engineers tried to patch things up (Levinson 17-18).

As previously stated, Klezcz considered low-code a viable tool for prototyping, which I concede. In an environment that is temporary, without production strains, and solely for trying out ideas and creating concepts, low-code can be a powerful tool that eliminates overhead for writing code that will probably just be rewritten anyways. However, low-code still falters in the ways stated above, by folding under demand, creating inefficient code, and turning into vendor-lock in when overused. All the issues with low-code stem from using it as a production tool and not as a development tool, which is where it belongs.

All things considered, low-code's popularity stems from its claims to timesaving, cost reduction, and ease of use, and while these things are true, they are also potential pitfalls. It was seen with the Healthcare Marketplace that overuse of a similar solution lead to incompatibility and production issues, eventually leading to the codebase being rewritten. Its timesaving abilities are not completely worthless—in a situation where the stresses of a userbase demanding a product and the need for a product to integrate with other programs is not critical, low-code shows its worth. These situations include concept testing, app prototyping, and overall early development stages of a program. Whereas these early prototypes would normally be rewritten

later in the development stage, they can instead be put together easily with a low-code solution so that a custom-code solution can be written later on, based on the low-code.

Works Cited

Forbes Technology Council. “Low-Code And No-Code: 19 Potential Downsides To Be Aware Of.” 28 July 2023. Accessed 6 November 2024. https://www.forbes.com/councils/forbestechcouncil/2023/07/28/low-code-and-no-code-19-potential-downsides-to-be-aware-of/

This is an article by Forbes. In the article, the authors outline various shortcomings and limitations of Low-Code and No-Code, including poor infrastructure design which causes performance and reliability issues, vendor lock-in, technical debt, and lack of customization. The article concedes that low-code is a powerful tool for basic development, but urges the reader to be skeptical and wary of these tools.

This is an effective source because the article touches on basically all of my claims regarding low-code. It is reliable because Forbes is a well known general publication, and their Technology Council is no exception. However, this source is a less detailed overview document, and a secondary source.

Khan, Masudal, interview by Carson Smith, 9 October 2024.

This is an interview with my Introduction to Programming (ITSS 3311) professor. In the interview, I asked him about his work experience in Developer Operations (DevOps) and Information Technology (IT). I asked if he has ever worked with low-code software solutions, and what his experience was like working with them. I asked him about the overall long-term effects of these coding solutions on his department and company. Lastly, I ask for his personal preference of low-code solutions, and for an explanation behind his preference.

This is an effective source to my argument because his thoughts generally back up my position that low-code information technology solutions do more harm than good. Furthermore, he has a wealth of professional experience in IT, having worked in the industry since 2002. He has seen and implemented low-code developments.

Kleszcz, Michal. "The top 5 limitations of no-code and low-code platforms." 13 December 2023. Accessed 10 October 2024. https://www.apptension.com/blog-posts/no-code-and-low-code-limitations.

This is a blog post written by the CTO (Chief Technology Officer) of AppTension, an end-to-end software development company. In the blog post, Michal touches on 5 most glaring limitations and drawbacks of no-code and low-code tech solutions, which he considers to be scalability concerns, vendor lock-in, security issues, development pitfalls, and the lack of team collaboration to develop these solutions. Michal concedes situations where low/no-code is viable, such as for early idea development and prototypes, but maintains these solutions are still not ready for production.

This is an effective source for my paper because it reinforces two of my points: vendor lock-in and scalability issues. Michal is a credible source, being the CTO of a software development company, he likely has experience with both fully programmed and low-code development. However, Michal could be biased, as AppTension is a company focused primarily on helping other companies transition away from low/no-code.

Levinson, Daniel R. "HealthCare.gov – CMS Management of the Federal Marketplace." February 2016. Accessed 10 October 2024.

https://oig.hhs.gov/documents/evaluation/2981/OEI-06-14-00350-Complete%20Report.pdf.

This is a case study written by personnel in the Department of Health and Human Services that overviews the development of HealthCare.gov, a health insurance marketplace that, on release in 2010, catastrophically failed and was not fixed until 2 months after its planned release. The case study details one of the causes of this failure to be a contractor's overusage of Model-Driven Architecture (MDA)—a type of automatically generated, software-written code that is an early example of a low-code development solution. It was found that CGI Federal—the contractor in question—had overused their MDA, leading to issues with integration and release performance.

This is an effective source for my paper because it illustrates how the overusage of a type of low-code solution had catastrophic effects at release. It is credible because it is a federal government case study written by the Department of Health and Human Services. However, the case study is relatively old, having been released in 2016 with a topic from 2010.

Little, Jay. "Low Code Software Development Is A Lie." 14 April 2023. Accessed 10 October 2024. https://jaylittle.com/post/view/2023/4/low-code-software-development-is-a-lie/

This is a blog post written by Jay Little, a software engineer for Paylocity. In this blog post, Jay details how low-code solutions do not work as intended because writing code requires more than just a knowledge of programming languages. Jay asserts that clients need a qualified and trained software engineer to create a product that is practical and functional, which solves the client's problems effectively and efficiently. The problem

with low-code solutions, Jay claims, is that non-technical clients don't know how to create or validate a technical solution, no matter if they actually wrote the code or not. Jay also touches on long term issues like vendor lock-in, which typically keeps the client paying for the software long after they've created it.

This is an effective source because it offers a first-hand account of the professional and technical problems that arise when dealing with low-code solutions. Jay's experience is in line with my claims regarding these solutions, that they do not generate quality product, and leave the client with long-term issues like vendor lock-in. However, this is a casually written blog post, even if first hand, which takes some of the credibility away.

Misera, Matthew. "RedZone Robotics Case Study." https://www.lowcode.agency/casestudies/redzone.

This is a case study that documents the usage of a low-code solution by RedZone Robotics. In this case study, RedZone Robotics used this product in order to prototype a user-facing robot control app. However, the solution was not implemented for production, and instead re-designed with custom code. RedZone Robotics had to pay the contractor as long as they used their prototype, which demonstrates vendor lock-in.

This is an effective source because it illustrates that even when a company uses a low-code solution effectively, its scope is limited, and they are still subject to issues like vendor lock-in. It is reliable because it is a case study conducted by a low-code company.

Nolle, Tom. "What IT pros need to know about low-code limitations." 11 October 2022. Accessed 10 October 2024. https://www.techtarget.com/searchitoperations/tip/What-IT-pros-need-to-know-about-low-code-limitations

This is an article written by Tom Nolle, a chief editor of Andover Intel, which is a well-known and reliable tech and market news online publication. In this article, Tom summarizes the use case of low-code. He then picks at low code's limitations, such as low customizability, poor integration, and poor application performance. Although he concedes that the products are satisfactory for small or prototype projects, he does not believe the products are appropriate for production or user-facing projects.

This is an effective source because it reinforces my claims regarding low-code solutions. It is credible because the author is a well-known tech writer. However, the author is just that—a writer—with no documented work experience.

Owens, Sean, interview by Carson Smith, 7 October 2024.

This is an interview with my IT for Business (ITSS 3300) professor. In the interview, I asked him about his work experience in IT Development. I asked if he has ever worked with low-code software solutions, and what his experience was like working with them. I asked him about the overall long-term effects of these coding solutions on his department and company. Lastly, I ask for his personal preference of low-code solutions, and for an explanation behind his preference.

This is an effective source of my argument because his thoughts generally back up my position that low-code information technology solutions do more harm than good. Furthermore, he has a wealth of professional experience in IT, having worked in the industry since 1997. He has seen and implemented low-code developments.

Scialli, Nick. "Why I'm skeptical of low-code." 30 December 2024. Accessed 10 October 2024. https://nick.scialli.me/blog/why-im-skeptical-of-low-code/.

This is a blog post written by Nick Scialli, a senior software engineer at Microsoft. In this post, Nick outlines several real-life pitfalls of low-code development among non-technical clients. For the most part, the post focuses on technical issues that occurred, such as problems with integration, poorly designed and generated database code, and problems with maintaining and improving the product once created. He clarifies that the article is not just to completely discredit low-code solutions, but to inform others to treat these solutions with skepticism, so that they do not suffer the same pitfalls.

This is an effective source for my argument because it backs up the technical concerns I had regarding low-code, especially difficulties in integrating and maintaining the software. The source is credible because it's first-hand experience written by a software developer at Microsoft. However, this is also a casual blog post, which reduces some credibility.