Low-Code's effects as Abstraction and Automation

A Research Paper submitted to the Department of Engineering and Society

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Richard William Taylor

Spring 2023

Advisor

Kent Wayland, Department of Engineering and Society

Introduction

The ability to efficiently solve problems is and was the main driver of the creation and advancement of computer technology. As society's demand for advancement in all sorts of fields grew, so did the need for more efficient methods of solving the problems that arose. In the more modern world, the problems that drive technological innovation are different, and the innovations that spawn as a result are different as well. Business', organization's, and individual's success is more and more reliant on efficiency: The faster they are able to provide their service or product, or fix issues when they arise, the more value they hold to clients and consumers. Software has become the prime tool for such efficiencies, and therefore there is always a drive to be able to more efficiently create software itself.

The main method developers have used to create software faster is through abstraction. As time goes on, less and less details are visible to the programmer, hiding anything except the most relevant details about the program. Older programming languages and required the programmer to control almost every aspect of how the software interacted with the computer: from managing allocation of memory to controlling the math and logic behind every operation the software tells the computer to make. The abstraction of software detail, and the efficiencies in programming it has afforded, have continued, leading to the creation and use of Low-Code platforms. Low-Code platforms can be seen as the latest form of abstracting code, going as far as removing the need for any programming experience or knowledge at all (O.E. Team, 2021). This brings massive changes to the field of software development. Traditionally, creating software solutions to business problems is done either by in-house IT departments or is contracted to software-specific companies, which leads to application development being both more time consuming and expensive (Coleman, 2022a). However, low and no-code platforms are bringing non-technical professionals into these roles, with Gartner reporting that as much as 41% of employees outside of IT fields are now creating or customizing their own technological solutions (2021). As the importance of understanding technical detail and possessing programming education or experience lessens, the role of the Software Developer and the IT field will shift as well. It is important to analyze the effects of this change, as it impacts the efficiency of a major market in the global economy, and an estimated 1.6 million developers in the U.S. alone (U.S. Bureau of Labor Statistics).

This research aims to understand how this shift will affect both traditional and new, nontechnical developers, as well as the shift in what values are important in the field of software development, to answer the question of if the growing adoption of Low-Code is truly a revolutionary change, advancing automation of a high-skill field and displacing traditional developers, or whether it is analogous to the changes brought by past forms of software abstraction.

Background and Literature

The aim of abstraction in software is to hide unnecessary details and complexity from the developer. High-level programming languages like Python abstract details that lower-level languages like C require users to include. Languages like C, in turn, abstract users from the machine executable code that it is assembled into, which itself is a list of instructions, an abstraction of the details of executing basic logic comparisons on single bits of data by the CPU. Each new generation of abstraction provides faster, easier development, as programmers no longer need to focus on individual and intermediate steps, but rather what the program is actually meant to do (Coleman, 2022b). Low-code platforms serve as the next level of this abstraction. Just as high-level programming languages compile or assemble into more complex and less human-friendly code, so does low-code. However, this is taken a step farther with most low-code and no-code platforms. High-level programming languages still look, to someone with little or no programming experience, the same as low-level ones: they are pages filled with text commands and symbols that the programmer must read through and understand how they interact. Low-code and No-code platforms look different- most are very visual, allowing users to create components, widgets, functions, or other general "chunks" of functionality, and then tell the platform how they are connected and how they should interact.

This greatly expands on what is possible for non-technical personnel: it allows for users to create complete software applications without writing much or any of the code that they traditionally comprise of. This new class of "Citizen Developers", those outside of traditional IT or development fields, is often able to develop their own applications and tools, instead of relying on separate IT teams to translate the description of their needs into a solution (Hurlburt, 2021, p.4). Platforms range in capabilities and usages: From simple website and front-end builders such as Wix, WordPress, and SquareSpace, which are used primarily by individuals and small businesses, to powerful platforms from software companies like Salesforce, Mendix, Microsoft, and more, that can create complete, sophisticated software systems and applications for enterprises (Bendor-Samuel, 2022).

Conversely, Low-code platforms may be seen as an agent of change in a different way, representing the first step into automating and potentially replacing high-skill jobs. Recently, advancements in technologies such as Artificial Intelligence, Machine Learning, and Big Data, have led to the notion that high-skill tasks can be automated with software, in a similar fashion as to how low-skill or routine work has been automated with robots and computer technology in the past (Acemoglu & Restrepo, 2018, pp. 204-205). Brekelmans and Petropoulos utilized data from

24 European countries to predict “probability of automation” among different job groups, finding that unlike past automation, which served to increase polarization between “high” and “low” skill jobs, recent technologies are likely to have a deeper impact across a wider range of jobs and tasks (Brekelmans & Petropoulos, 2020), including Software Development. Low Code platforms share the approach of removing human dependence from these tasks, which leads to a level of automation in the industry of software development itself, setting them apart from the abstractions in programming languages seen in the past.

Methods

To answer the question of how low and no-code platforms compare and differ from past forms of programming abstraction, I analyzed literature revolving around the capabilities and limitations that the platforms provided to both Professional and Citizen Developers, coming from both users of these services as well as the companies that produce them, in order to obtain views from different actors affected by the system. By gathering the views of the different groups that are most affected by the adoption of Low-Code, it was possible to get a better understanding of how the values proposed by the platform providers are actually being used for change in the field of software development. I also gathered information on past abstractions in programming and how they are classified from past conferences and experienced developers, who were developing both before and after some past abstractions became widespread, such as adoption of higher-level languages over low-level ones. This information allowed for insight into how values of programmers shifted following past abstractions, and why they afforded benefits to programming efficiency, which was used to better understand the benefits provided by Low-Code platforms, as well as differences in the “power” past abstractions gave to traditional developers and Low-Code gives to non-technical professionals. Finally, in order to gain insight on how current professionals view low-code as an agent of change, as well as its perceived and predicted success, Information from forecasts by industry leaders and interviews with various technical experts was drawn from. By gathering information from both low-code providers, users, and traditional developers, the views of different actors who take part in the sociotechnical system were analyzed, in an attempt to better understand the affects Low-Code platforms might have on the Software Engineering profession, and how they compare to traditional automation and programming abstraction.

Results

The groups most affected by the advent and expansion of Low-Code technology are "traditional developers", those who leverage technical programming skills and expertise to create traditional software artifacts, "citizen developers", the new class of less technical or non-technical professionals who utilize Low or No-code platforms to create their own applications and solutions, instead of relying on traditional developers, and of course the Companies that create and provide the Low and No code Platforms themselves. Most platform providers tout the benefits felt by both citizen and traditional developers, focusing on the idea of "democratization of programming", or extending the ability to create independently to citizen developers (O.E. Team, 2021), while maintaining that through incorporating Low-Code techniques in addition to their traditional development, traditional developers can greatly increase the efficiency and ease with which they create software (Coleman, 2022a). Providers argue that their platforms bring revolutionary changes to who is able to create, differing from general program abstraction of the past, but also provides this traditional form of abstraction to developers. Traditional developers mirror some of these views, but further acknowledge the limitations that low-code imposes on citizen developers. These professional developers primarily note this restriction when asked about how the platforms affect their future. Jon Chan, Director of Engineering at StackOverflow, pointed out that "Low-code/no-code tools tend to look for a general use case, and that can restrict how flexible that software can be" (Dene, 2022). Stack Overflow CEO Prashanth Chandrasekar describes the second limitation of Low-code for citizen developers, stating that "Early developers blindly using low code or no code tools without learning the fundamental principles of writing code will inevitably hit a ceiling, particularly when they have to unpack what they created." (Dene, 2022). Mainly, Traditional developers view low and no code platforms as somewhat novelty, making development somewhat more accessible, but not enough to allow for high complexity tech to be made without professional developers, and as the amount of interconnected software and technology driving everyday life increases, IT professionals must remain to better understand and manage these systems (Hurlburt, 2021).

The Low-Code revolution is also often touted as a form of "High-skill" automation. Traditionally, high-skill work encompasses complex tasks involving human judgement or problem solving- which would include development of any kind of software artifact, no matter how simple or complex. Acemoglu & Restrepo, through the creation of a model for this type of automation, show that recent advancements in automation serve to significantly affect labor markets of "high-skill" jobs (2018), potentially including programming and software development. Brekelmans & Petropoulos, using data from 24 EU countries, evaluate that these new technologies, of which many consider Low-Code platforms to be a precursor to, are likely to have a non-negligible impact on high-skill jobs, implying that the labor market affects modeled by Acemoglu & Restrepo are possible. However, they also conclude that such "a high probability of automation may also be associated with the creation of new [high-skilled] tasks and jobs though the productivity gains from adopting [automation] technologies" (Brekelmans & Petropoulos, 2020).

Low-Code is also often related to former programming abstractions, instead of outright automation. Abstraction in programming aims to hide as much detail as possible to allow for the programmer to focus on design and overall function of their program. With low and no-code platforms, the details being abstracted are largely blocks of code itself, allowing users to interface with machines in an easier format, leading to many describing the platforms as nothing more than a new generation of programming languages, more abstracted than the last (Coleman, 2022b). This viewpoint, however, ignores some of the bigger offerings of low-code: namely, that it provides a much friendlier interface to a less-technical developer, and allows for use of the "abstracted blocks of code" without needing the prior knowledge of how or why they work at all.

Abstraction in programming is tightly coupled with the idea of Program Comprehension, the activity of understanding how a program or software system works. Through observation and survey of developers, Koschke et. al. found that over 83% of professional developers encounter problems due to lack of information about functionality of a program or its components, primarily with "what the program is meant to do" and "why the code is implemented this way". Furthermore, they found that experience of developers is important in the task of comprehension, and that overall, developers will take strategies to avoid comprehension if they are able to (Koschke et. al., 2014). Low-code platforms can shift the ability or necessity of developers to comprehend the code they create, through abstraction of details, thus changing the values of the developer.

Companies whose main business does not revolve around software or tech creation find themselves needing software solutions to solve various business problems. Low-code platforms are rising in popularity because they allow more and more people to create tech solutions, and shift to learning traditional development practices as this need for software in previously non-tech markets rises (Dene, 2022). The low-code development technology market has increased substantially over the past few years, owed in part to remote-development during the COVID-19 pandemic, reaching $13.8 Billion in 2021 (Gartner, 2021). This continuous rise in the technology is expected to continue, suggesting a long-lasting shift in software development as a field and sustained changes in how people develop tech.

Discussion

In discussing the potential impacts of the Low-Code revolution on Software Development Professionals and the field as a whole, it is first important to understand that Low-Code is, depending on the definer, a new technology, and is being successfully marketed and growing rapidly. With this, expansion in capability and scope of platforms are sure to come in the following years. This information alone is enough to conclude that changes to the profession of software development are coming, but the magnitude of these changes, and what impact they will have on the values that are desired in a Software Developer are largely unknown, and what further discussion is attempting to discover.

The results found shows something of a disparity in how Low-code platforms are perceived by the actors most affected in the system. Traditional developers, tech experts and executives, and the platform providers themselves all agree that low-code democratizes programming, bringing the ability to create software to the masses and new found citizen developers, but the opinions of these actors on just how useful this new ability is are conflicting. Of course, Low-code providers will advertise their products as all encompassing, incredibly powerful means to redistribute the power of software among all people, technically inclined or not, because this will sell their systems. When this raises questions as to just how useful developers are with these new options available, most developers rely on the widespread limitations of the technology, in its current form at least, to justify their continued employment. It is important to note these limitations of low-code solutions, which mainly arise when complexity of the system that needs to be implemented increases. Citizen developers may be able to quickly produce various levels of software artifacts, but the majority of a software systems lifecycle is spent in the maintenance phase, and when an issue arises that requires more technical knowledge than the Citizen develop had used to initial develop, such as a problem with platform itself, or the way two components interact, they might be unable to solve it. Combined with the evidence that program comprehension is one of the primary tasks developers undertake during maintenance, the hiding of technical detail and knowledge by Low-Code may be more of a hinderance to Citizen developers than a benefit. When a bug or error requires the developer to understand what it is that some abstracted block of code, generated by a low-code platform is actually doing, the developer using the platform may not be prepared to handle it. This is a skill that traditional developers master throughout their education and careers, filling the gaps in knowledge that are allowed by low-code. It has been shown that all sorts of businesses will need complex, custom software solutions to their problems, and until Low-code platforms are greatly expanded in capabilities, it seems that traditional developers will be relied upon to create these solutions.

As most agree that the Software Developer is not going to become obsolete with the increased use of Low-Code by non-developers, it is interesting to analyze changes to the role from the lenses of regular programming abstraction and high-skill automation. While new tools like AI and Machine Learning are opening up the traditionally “high-skilled” fields to automation, as shown by Brekelmans & Petropoulos' study of such jobs, it is predicted that such tools will open up more high-skill jobs themselves, which in turn could greatly change the labor market for those jobs, as modeled by Acemoglu & Restrepo. This is analogous in a way to former computer and robotic automation of low-skill labor in the past: as factory assembly workers were replaced by machines, such machines needed to be created, controlled, and maintained: leading to new jobs being created themselves. This applies to the expansion of low-code as an automation tool as well, as new roles open when people are needed to create and maintain low-code platforms. However, these roles aren't completely new: they just have a focus on new skills and technologies, requiring software developers to have different abilities and knowledge than what might have been commonplace in the past.

An argument many traditional developers and other tech experts make is that the field of software development and the skills required of a software developer in the era of Low-Code will not make a major change at all. They argue that Low-Code is like many new tools and programming languages that come before it: another form of abstraction. One such voice, CTO of StackOverflow Jody Bailey, sums up this sentiment well, stating that "[They've] been hearing about low-code solutions for 30 years, only they had different names back then" (Dene, 2022). Abstraction in programming, and by extension that provided by Low-Code, is not a bad thing that makes developers less knowledgeable: it's a technique to allow them to develop more complex things efficiently. As mentioned, program comprehension makes up a large portion of a developer's role. Abstracting some functionality to a method, or block of drag & drop code, or anything to make it more human-accessible, means a developer doesn't need to spend their time understanding how that functionality works, or why it is implemented the way it is. This abstraction provided by low-code allows for developers to focus more on the large problems at hand, instead of the smaller methods that piece together to solve them, leading to faster, focused development. This phenomenon is shifting the skills developers value away from controlling the computer to understanding how different components and functionalities work together, and using them correctly to solve a larger problem.

Such shifts in necessary skills valued by developers over time is well documented, but most notably, old skills rarely become completely obsolete, and there are still many use cases for lower-level abstraction programming languages in various settings today. With the idea of Low-Code as a form of abstraction, this implies that the traditional development we see today won't change completely, but rather new skills will be introduced and slowly, older forms of development will become less popular. To understand this shift, we can analyze how the values of developers changed in the past, as coding in assembly and machine languages fell out of style, making way for high-level languages more common today. This allowed for bigger, more complex systems in less time, as developers needed not focus on small details or optimizations, but could rely on underlying systems to handle those cases. Those changes did not happen overnight, but slowly built over many years, with new iterations and more detail abstracted over time. In the same way, low-code affords these benefits, but to a greater extent, and like the shift to high level languages, will take time to allow for low-code platforms to advance and mature into more powerful tools.

In their current state, low-code platforms are more akin to new, further levels of program abstraction than means for high-skill automation that many other fields are starting to experience. The main difference between rising popularity of Low-code and past abstractions, like the increase in use of high-level languages like Python and Java, is in who most benefits from the complexities being hidden. With low-code platforms, it is not only the lower-level computer instructions and actions being hidden from the user, but the intricacies of how the code itself is written and works. This has allowed for people with les-technical backgrounds and limited technical expertise to be able to create their own solutions for the first time, instead of relying on relaying their needs to others, leaving much of the intricacy and functionality of the solution up to interpretation.

Conclusion

Low-Code is not the next big IT fad. As time goes on, market forecasts and industry leaders have predicted Low and No-code solutions and platforms to grow in popularity, fueled by the efficiency they afford to businesses and individuals in creating software solutions. Current limitations of the technologies are likely to shrink, becoming more robust and offering more customizability. This technology is here to stay, and might develop in ways no one could have imagined before. Low-Code is just one of those ways in which developers try to increase their efficiency, as to create bigger, better solutions to more and more complex problems. As the technology develops, as well as artificial intelligence technology that has increasingly become more advanced, it is an important to continue to analyze how such advancements will change the way programming and software development is done. More research into understanding the limitations of Low-Code, and in what scenarios or environments those limitations are most restricting, will help answer that question.

The demand for software as a solution will not slow down, and techniques and tools like Low-Code are developers' ways of keeping up with demand, as the values the developer provides, and skills and knowledge they need shift. The software developer completes the same tasks as those from 40 years ago, but in much, much different ways, and the software developer

40 years from now is sure to as well. Low-Code is just one part of these changes that have been seen before, and will be seen again.

Works Cited

Acemoglu, D., & Restrepo, P. (2018). Low-Skill and High-Skill Automation. *Journal of Human Capital*, 12(2), 204 - 232. https://doi.org/10.1086/697242

Bendor-Samuel, P. (2022, June 8). *What Enterprises Need To Know About Low-Code Software Development Platforms*. Forbes. https://www.forbes.com/sites/peterbendorsamuel/2022/06/08/what-enterprises-need-to-know-about-low-code-software-development-platforms/?sh=24baed607ff9

Brekelmans, S., & Petropoulos, G. (2020, June 29). *Artificial Intelligence's great impact on low and middle-skilled jobs*. Bruegel. Retrieved February 1, 2023, from https://www.bruegel.org/blog-post/artificial-intelligences-great-impact-low-and-middle-skilled-jobs

Coleman, S. (2022a, April 22). *Low-Code vs. Traditional Development*. Low-Code Connection. https://lowcode.com/articles/low-code-vs--tradition al-development.html

Coleman, S. (2022b, August 13). *Abstraction in programming: Taming the ones and zeros.* VentureBeat. https://venturebeat.com/programming-development/abstraction-in-programming-taming-the-ones-and-zeros/

Dene, K. (2022, August 10). *Will low and no code tools ever truly disrupt tech development?* Stack Overflow Blog. https://stackoverflow.blog/2022/08/10/will-low-and-no-code-tools-ever-truly-disrupt-tech-development/

Gartner, Inc. (2021, February 16). Gartner Forecasts Worldwide Low-Code Development Technologies Market to Grow 23% in 2021.

https://www.gartner.com/en/newsroom/press-releases/2021-02-15-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-23-percent-in-2021

Hurlburt, G. (2021). Low-Code, No-Code, What's Under the Hood?. *IT Professional, 23*(6), 4 - 7. https://doi.org/10.1109/MITP.2021.3123415

Koschke, R., Maalej, W., Roehm, T., & Tiarks, R. (2014). On the comprehension of program comprehension. ACM Transactions on Software Engineering and Methodology, 23(4), 1–37. https://doi.org/10.1145/2622669

O'Reilly Editorial Team, (2021). *Low-Code and the Democratization of Programming*. O'Reilly Media, Inc.

U.S. Bureau of Labor Statistics. (2022). *Software developers, Quality Assurance Analysts, and testers: Occupational outlook handbook*. https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm