DZone®
A DEVADA MEDIA PROPERTY

TREND REPORT

Low-Code Development

Empowering Business Users, Enabling Developers

BROUGHT TO YOU IN PARTNERSHIP WITH

SAUCELABS

Welcome Letter

By Blake Ethridge, Community Manager at DZone

“Help, I only have 24 hours a day! I’ve got 500 projects, and 400 of those require IT and dev teams to assist. Suddenly, all my teams are remote, and my competition, who has a third of our overall headcount, is eating our lunch!”

Enter low code and its rapid adoption.

Citizen developers comprising business teams can now use low code to do more with less. The citizen automation they drive could begin freeing up the significant time spent on inefficient processes and in areas where they’d otherwise need dedicated development support. Operationally, overnight low-code tools return time *and* resources to both business and IT teams, while also allowing them to use the same tools to innovate, scale, deploy, and prototype faster.

In this age of hyper-automation, low-code solutions enable entire companies — regardless of an individual’s technical background — to increase efficiency, improve collaboration, and strengthen their competitive advantage.

IT and development teams who might often have to assist business users in building custom tools have seen this work decrease due to the capabilities low code offers, like automating and scaling processes and workflows without the usual overhead and project work that could bog them down for months of endless production. For example, a competitor might have an entire development team building tools or custom features that take months to complete, prototype, and deploy, whereas if you can use low-code tools to achieve the same, the resulting velocity and time savings will propel you light years ahead of your competition.

As we see in the following survey findings, the new world order is business teams leveraging low code for use cases including web forms, simple databases, and request handling — and to further drive collaboration with IT in accelerating the development of solutions and features in record time. DZone’s research shows readily where low code and its adoption stand today, how organizations are using these platforms, challenges (getting solution and technical architects involved early on), and the areas of opportunity.

I recently spoke with Javier Teso, Co-Founder of Kumologica, about their input on the latest trends around low code.

What’s driving the adoption of low-code tools?

Javier: The increasing demand for digital transformations is exacerbating the current shortage of IT professionals worldwide. There is a clear need to change the way software is developed. Zero- and low-code tools provide a higher level of abstraction that allows more people to enter the tech industry and for existing professionals to become more efficient.

What should companies focus on for their low-code efforts?

Javier: The ultimate goal is for the business domain experts to participate actively in the development process. Development tools, like zero and low code, should be viewed as a “collaboration tool” in the same way email and JIRA are.

Everyone is talking about low code, and in this report, you’ll find out how it’s affecting developers, business users, and software development itself through a deep analysis of our extensive industry survey, plus insights from a variety of experts within the DZone community. Welcome to DZone’s 2021 Low-Code Development Trend Report, and welcome to learning about it from the frontlines of this movement.

Sincerely,

Blake Ethridge

ORIGINAL RESEARCH

Key Research Findings

An Analysis of Results from DZone's 2021 Low Code Survey

By John Esposito, PhD, Technical Architect at 6st Technologies

In July 2021, DZone surveyed software developers, architects, and other IT professionals in order to understand the state of low-code development.

Major research targets were:

1. Ideal and actual low-code development by problem domain
2. Interaction of low-code and full-code development
3. Quality and maintainability of software built via low-code vs. full-code methods

Methods:

We created a survey and distributed it to a global audience of software professionals. Question formats included multiple choice, free response, and ranking. Survey links were distributed via email to an opt-in subscriber list and popups on DZone.com. The survey was opened on July 1st and closed on July 14th. The survey recorded 680 responses.

In this report, we review some of our key research findings. Many secondary findings of interest are not included here. Additional findings will be published piecemeal on DZone.com.

Research Target One: Ideal and Actual Low-Code Development by Problem Domain

Motivations:

1. Problem domains vary by depth and breadth — by complexity and openness. We assume that code is more information-dense than low-code programmer inputs and suppose that, as a result, low-code development methods are applied more in less complex, more focused problem domains.

   Note: We mean "information-dense" in an informal sense, which is necessary because we intend "information" from both machine and programmer points of view. Both the necessary entailments of the programmer input, given the definition of the compiler or interpreter (which can be formalized) — and also the less formalizable, hints about programmer intent. This is conveyed by things like variable naming, class (or struct or other namespace-type bucket) structures, and file structures.

2. Just as waterfall software development can be overkill with respect to design, full-code development can be overkill with respect to solution definition.

   A major motivator for the *Agile Manifesto* was that full-complexity specification and implementation are often premature, considering how poorly defined many business problems are before solutioning. Prototypes and frequent iterations are common ways to tackle complexity piecemeal, but prototyping and quick iterations can sometimes seem wasteful or frustrating insofar, as they may be accomplished using "throwaway" code (that has no relation to the final product, either as reworked code or as a disposable but helpful path to deeper understanding of the problem). For early prototypes and iterations especially, we supposed that low-code tools could improve software development.

   Note: Prototyping and iteration are also good practices in writing code, of course. Agile-style development can often result in higher-quality code, precisely because early drafts are more aggressively thrown away. But for certain tasks (e.g., a first pass at user interaction paths), a large amount of the code needed to produce a usefully iterable result is often too generic to count as a useful "first draft" of code to be rewritten later. These are the situations that become frustrating because they result in code that is truly wasted — not useful for production runtime (which is fine) but also not useful for learning (which is not).

3. Visual programming tools have been available for decades, and rigorous visual representations of logical, mathematical, and physical concepts have been around for centuries more (Euler and Venn diagrams, Pierce's existential graphs, Frege's two-dimensional Begriffschrift notation, Feynman diagrams, Spencer-Brown's laws of form, etc.). Yet the (English language) phrases "software developer" and "computer programmer" still strongly connote "writes code."

   We wanted to understand why, whether it makes sense to try to change these connotations, and whether thinking more carefully about low-code development might also improve notation systems used in other programming and mathematical contexts.

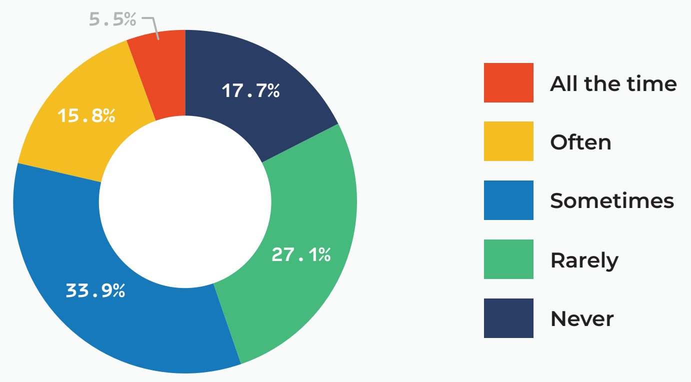
USAGE OF LOW-CODE PLATFORMS

We wanted to know how often low-code platforms are used and how software professionals' opinions about low-code development differ among those who have and have not done low-code development. So we asked:

How often have you built software using a low-code platform?

Results (n=672):

Figure 1



FREQUENCY OF USING LOW-CODE PLATFORMS TO BUILD SOFTWARE

Observations:

1. Far more respondents than we expected have used low-code platforms (82.3%) and more specifically, have used low-code platforms often (15.8%) or all the time (5.5% — i.e., over a fifth of respondents have used low-code platforms often or all the time).

   This may indicate higher than usual selection bias among respondents, or it may indicate that low-code usage is, in fact, much higher than we expected.

2. The distribution of responses to "primary programming language at work" is not substantially different among respondents to this survey vs. respondents to other surveys sent to the same population (e.g., ~50% Java, ~15% Python). Nor are the distribution curves of years' experience or company headquarters region (though responses to the latter question differed somewhat more than the others).

   From this comparatively close overlap in respondent demographics across our surveys sent to the same population, we tentatively conclude that there was *not* an extremely strong selection bias to this survey. This leads us to interpret responses to the "how often have you used low-code" question as relevantly accurate. 82.3% of surveyed software professionals have used low-code platforms at some point.

3. Respondents who primarily build web applications were somewhat more likely to have never used low-code platforms than respondents who primarily build enterprise business applications (17.5% vs. 14%). And those who build enterprise apps were somewhat more likely to have often used low-code platforms than web app builders (18.4% vs. 16.2%).

   This is consistent with the marketing of low-code tools and with our data on common low-code use cases, discussed below.

4. Low-code platform use by senior (>5 years' experience as a software professional) and junior (≤5 years' experience) respondents in Table 1 below may suggest a U-shaped curve in low-code platform usefulness over time, a greater degree of specialization among those more experienced, or something else.

Table 1

Use of Low-Code Platform	Senior	Junior
All the time	6.2%	3.9%
Often	16.5%	13.2%
Sometimes	31.0%	38.8%
Never	18.8%	15.5%

5. Some difference in specialized experience is suggested by significant differences in senior vs. junior opinions on the utility of low-code integration middleware: Senior respondents are much more likely to consider low code not useful for integration middleware vs. junior respondents (27.8% vs. 16.1%).

   Because (anecdotally) junior software professionals are less likely to be involved in integration design, we suspect that senior respondents' stronger rejection of low-code integration middleware comes from having done more integration overall. However, junior respondents are much more likely to have used low code for integration middleware (75.6% vs. 54.4%), although less likely to have done so often (10.6% vs. 16.3%).

   We take these results to mean that senior respondents' stronger judgment of the inutility of low code for integration middleware may come from a relative lack of experience using low code for it, despite presumably having more input on higher-level system design.

USE CASES FOR WHICH LOW-CODE PLATFORMS ARE USEFUL

Software development that involves complex business rules, elaborate build processes, high-level pipeline management, or assumptions about (and therefore couplings with) infrastructure, often also involves some kind of visual input. In our experience, some of these visual definition processes are more successful than others, but negative experiences with using visual methods can easily result in weighting the cost of the notion disproportionately high. For example, frustration with heavyweight UML modeling can often lead to discarding UML — or even all system diagrams — altogether.

This may be an oversimplified, foolishly homogenized overreaction to heterogeneous distribution of visualization's appropriateness. When the visuals result in an executable system, discarding the visual notation means discarding the system that creates executable commands completely. We wanted to understand which use cases low-code platforms are useful for.

So we asked:

In your opinion, what use cases are low-code platforms useful for?

IMAGINED LOW-CODE UTILITY BY USE CASE: OVERALL

Results across all respondents (n=664):

(See Figure 2 on the next page)

Figure 2

USE CASES FOR WHICH LOW-CODE PLATFORMS ARE USEFUL

	Not useful	Somewhat useful	Very useful	n=
Enterprise CRUD	12.5%	46.2%	41.3%	**656**
Interactive web forms	7.6%	38.8%	53.6%	**659**
Integration middleware	24.0%	44.9%	31.1%	**653**
Request handling (e.g., support tickets, inbound lead recording)	13.8%	37.1%	49.1%	**658**
Business process management	14.6%	35.7%	49.7%	**664**
Simple databases (e.g., address books, product lists)	11.0%	38.7%	50.3%	**662**
E-commerce and procurement	23.7%	43.9%	32.4%	**654**
Learning management	16.8%	47.8%	35.4%	**655**
Physical modeling	31.7%	41.5%	26.8%	**650**
Machine learning pipelines	28.2%	39.7%	32.2%	**650**
ETL	22.5%	40.3%	37.2%	**645**

Observations:

1. Interactive web forms were the use case most commonly marked as either somewhat useful (38.8%) or very useful (53.6%), with very useful easily dominating; only 7.6% marked not useful. The next-closest use cases were simple databases (11% not useful), followed closely by enterprise CRUD (12.5% not useful), request handling (13.8% not useful), and business process management (14.6% not useful).

   We have not done a comprehensive feature review of all low-code platforms weighted by usage (the latter probably not a number we would have access to anyway). But on informal scan of feature sets, and based on use cases advertised, this list of four top use cases (interactive forms, simple databases, request handling, business process management) seems to be well served by existing low-code platforms.

2. We take a stark difference between "somewhat useful" and "very useful" responses to indicate stronger opinion, and hence, sometimes better-informed judgment (barring irrational bias). The use case for which "very useful" responses most strongly outweighed their "somewhat useful" responses was interactive web forms (53.6% very useful vs. 38.8% somewhat useful — a 14.8% difference). Almost the same magnitude of difference in the opposite direction was obtained for "physical modeling" responses (41.5% somewhat useful vs. 26.8% very useful — a 14.7% difference).

   These results suggest that current low-code tools are well-tuned for interactive forms but poorly tuned for physical modeling. There may be some good domain-specific reasons for this difference (e.g., physical models are expressed mathematically, and mathematical notation by modern convention consists of one-dimensional strings; web form development can easily be WYSIWYG), but this may also indicate some room for improvement in low-code physical modeling.

   For example, the modern, basically Leibnizian notation for an integral ∫ is two-dimensional. And standard multiple-integral notation — which denormalizes the integration symbol set into a linear sequence: ∫∫∫ — does not leverage the same visual intuition as the single-integral symbol does. And it also does not clearly distinguish between integrations that commute and those that do not. At execution (calculation) time, the integrals are denormalized, of course, but successful mathematical notation represents the mathematical abstraction, not the calculation steps. Presumably, this is partly why Leibniz's notation outlasted Newton's.

   We conjecture that some portion of the large gap between "somewhat useful" and "very useful" responses to the physical modeling use case could decrease with better visualizations that blur the line between user input system and notation system — and may improve the latter along with the former.

3. The use cases for which the "somewhat/very useful" response difference obtained the least were ETL (40.3% somewhat useful vs. 37.2% very useful) and enterprise CRUD (46.2% somewhat useful vs. 41.3% very useful).

   We interpret these responses' wishy-washiness, combined with the generally high level of utility expressed, as a weaker signal of opinion. And hence, it is less useful in determining tool/platform maturity than the larger somewhat/very differences described above.

4. The three use cases most likely to be marked as not useful are, in order from most likely to least likely: physical modeling; machine learning pipelines; and (tied for third) integration middleware, e-commerce, and e-procurement. However, the largest bucket of free-response answers related to integration or middleware (on our qualitative analysis), which suggests that the response choice of "integration middleware" may have been too specific.

   Our suggestions above regarding the somewhat vs. very usefulness of low-code platforms for physical modeling apply to the high "not useful" response rate as well. Machine learning pipelines are specialized, and answers may reflect bias from respondents who are guessing. This interpretation is suggested by the significantly lower percentage of "not useful" responses for machine learning among those who often (20.9%) or all the time (28.2%) use low-code platforms.

   See discussion below for more on differences between those who have and have not used low-code platforms with respect to supposed ideal use cases for low code.

IMPACT OF LOW-CODE PLATFORM USE EXPERIENCE ON JUDGMENT OF SUITABLE LOW-CODE USE CASES

Respondents who build software using low-code platforms often or all the time evaluated low-code use cases somewhat differently:

Figure 3

BEST USE CASES FOR LOW CODE BY EXPERIENCED LOW-CODE USERS

	Not useful	Somewhat useful	Very useful
Enterprise CRUD	5.6%	31.0%	63.4%
Interactive web forms	5.7%	31.9%	62.4%
Integration middleware	8.5%	40.4%	51.1%
Request handling (e.g., support tickets, inbound lead recording)	7.1%	21.3%	71.6%
Business process management	6.3%	28.2%	65.5%
Simple databases (e.g., address books, product lists)	10.7%	29.3%	60.0%
E-commerce and procurement	13.8%	42.0%	44.2%
Learning management	15.7%	37.9%	46.4%
Physical modeling	20.3%	37.7%	42.0%
Machine learning pipelines	20.9%	31.7%	47.5%
ETL	13.0%	29.0%	58.0%

Observation:

The top five use cases that those who have used low-code platforms often or all the time are least likely to judge not useful are, in order: enterprise CRUD, interactive web forms, business process management, request handling, and integration middleware. The number of "often" or "always" responses is comparatively small (143), and the top two responses do not differ in count (eight — the difference between the two use cases obtained only with respect to percent of respondents for that use case). So our inferences here should be taken as especially uncertain. But the top five use cases vary significantly across the two populations in any case:

Table 2

TOP FIVE USE CASES FOR WHICH LOW-CODE PLATFORMS ARE NOT USEFUL

All respondents	Respondents highly experienced with low code
Interactive web forms	Enterprise CRUD
Simple databases	Interactive web forms
Enterprise CRUD	Business process management
Request handling	Request handling
Business process management	Integration middleware

Integration middleware, enterprise CRUD, and business process management received the greatest boost from those highly experienced with low code vs. all respondents. Among all respondents, integration middleware did not even make the top five marked the least not useful. Between all and experienced respondents, enterprise CRUD moved from third to first place, and business process management moved from fifth to third. Simple databases, the use case second least likely to be judged not useful by all respondents, fell out of the top five among highly experienced respondents.

These findings suggest that low code may be better for enterprise CRUD, worse for simple database, and better for business process management than software professionals generally believe. These three use cases may, therefore, constitute avenues for significant user experience and marketing improvements for low-code platform creators.

USAGE OF LOW-CODE PLATFORMS BY USE CASE

Among those who have used low-code platforms, we wanted to know which use cases low code has been used for. We asked:

What kind(s) of software have you created using a low-code platform?

Results (n=533):

Figure 4

SOFTWARE CREATED USING LOW CODE

	Never	Rarely	Sometimes	Often	All the time	n=
Enterprise CRUD	22.8%	15.9%	30.6%	21.3%	9.5%	527
Interactive web forms	16.9%	18.4%	30.4%	24.2%	10.1%	533
Integration middleware	30.8%	18.3%	25.5%	15.3%	10.2%	530
Request handling (e.g., support tickets, inbound lead recording)	25.7%	20.0%	25.7%	18.1%	10.6%	526
Business process management	28.7%	17.5%	24.3%	17.9%	11.5%	526
Simple databases (e.g., address books, product lists)	20.3%	18.1%	27.4%	22.6%	11.6%	526
E-commerce and procurement	43.4%	17.4%	19.9%	12.8%	6.5%	523
Learning management	45.4%	16.0%	20.6%	11.3%	6.7%	524
Physical modeling	49.6%	15.8%	18.6%	9.9%	6.1%	526
Machine learning pipelines	49.0%	15.2%	20.3%	9.1%	6.5%	527
ETL	39.2%	15.5%	20.8%	15.9%	8.5%	515

Observations:

1. With respect to general utility, actual use maps to ideal use (among all respondents) fairly well. The five low-code use cases least likely to be judged not useful and the five low-code use cases least likely to be marked as never performed are both, in order: interactive web forms, simple databases, enterprise CRUD, request handling, and business process management.

   This suggests that the overall functionality of low-code platforms matches software professionals' imagination of low-code capabilities reasonably well.

2. The degree of low-code platforms usage by use case and ideal use case do not map quite as well.

   Table 3

   BEST USE CASES FOR LOW CODE BY USE FREQUENCY

"Very useful"	"Often" or "all the time"
Interactive web forms	Interactive web forms
Business process management	Simple databases
Simple databases	Enterprise CRUD
Request handling	Business process management
Enterprise CRUD	Request handling

   The difference may suggest, for instance, that business process management has more low-code potential than is currently actualized. But this may also be accounted for by different degrees of task specialization — for example, interactive web forms are used in all types of applications, while many application types do not involve any enterprise CRUD.

3. Junior respondents are significantly more likely to have used low-code platforms than senior respondents for every use case answer choice available.

   This relationship was especially strong (>~20% difference) for interactive web forms (93.3% vs. 80.6%), e-commerce and e-procurement (70.3% vs. 51.4%), physical modeling (64.7% vs. 54.2%), and machine learning pipelines (65.3% vs. 54.7%), with enterprise CRUD not quite meeting the 20% difference threshold (82.4% vs. 74.9%). Considering that senior respondents have more years in which to use a low-code platform, the significantly greater use of low code among junior respondents strongly suggests that low-code platforms are becoming more mature for a variety of use cases.

   Note: If only recent years were considered, this difference might be accounted for by the hypothesis that senior respondents are more likely to be called in to solve problems too complex for low-code solutions. Since the question does not specify a time period ("have you created"), we take these junior vs. senior differences to reflect differences in experience, not just differences in present job roles.

Research Target Two: Interaction of Low-Code and Full-Code Development

Motivations:

1. Most software does not live in its own bubble.

   As more distributed architectures grow, with looser overall coupling, the need for a single tech stack across an application decreases as the importance of API contracts increases. If one microservice can be written in Java and another in Haskell, both part of the same application, then nothing stops an appropriate microservice from being implemented using low-code tools. Since this can and does happen, we wanted to know where and to what extent.

2. The problem of inaccessible source, already raised by API-first and microservice approaches to design, may be exacerbated by low-code solutions.

Perhaps all microservice source code can be searched — perhaps even across multiple repositories and written in different languages. But at least these languages are likely to be widely used and openly specified, allowing IDEs and code analysis tools to latch on to execution paths intelligently. Low-code solutions, however, are not typically as open and do not necessarily encourage the same paradigms as most popular programming languages. In fact, in our experience, low-code platforms tend to result in highly procedural, less structured programs, which can be hard to reason about.

In cases where applications are built using both low-code and full-code methods, we wanted to know how products of the two pipelines interact.

3. Low-code tools are sometimes (though not always) intended as a way to hand-hold inexperienced creators, allowing business users to focus only on the business logic being implemented.

   Handholding works best along happy paths, so handholding low-code solutions seem likely to do worse along unhappy paths. Our experience with low-code solutions is consistent with this deductive argument: We think that low-code systems tend to be difficult to debug. We wanted to know how widely this experience is reflected among software professionals at large.

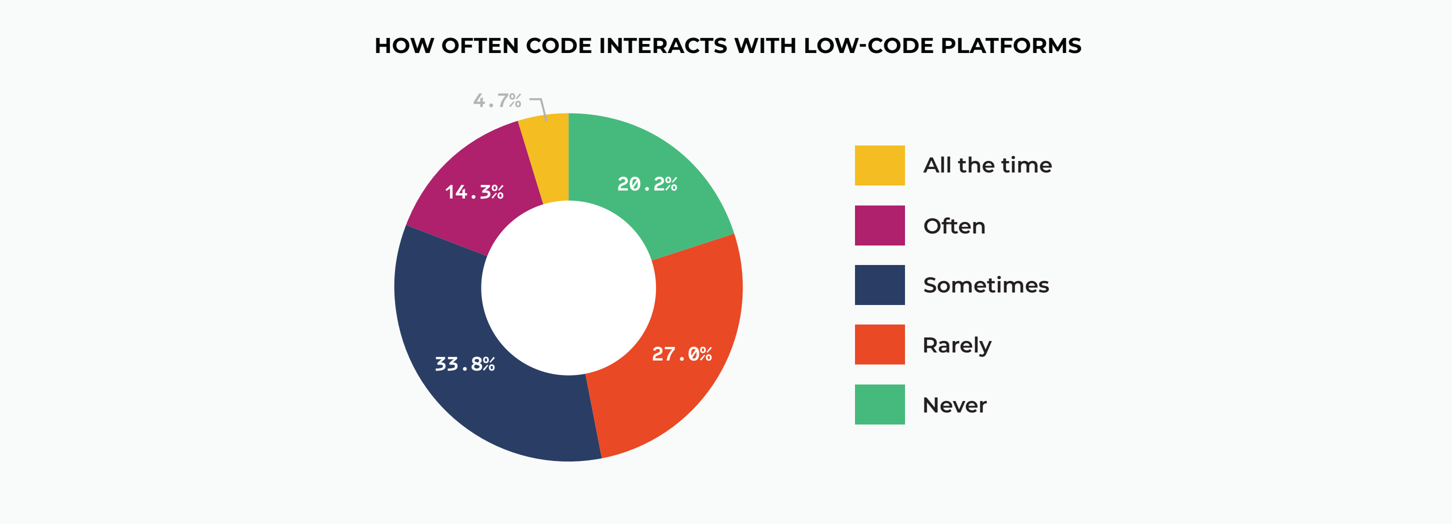
USAGE OF LOW-CODE PLATFORMS VS. INTERACTION WITH SOFTWARE BUILT USING LOW-CODE PLATFORMS

We wanted to know not only how many people build software using low-code platforms, but also how many people deal with the results of low-code platform use. As systems grow more decoupled, just happening to interact with the output of a low-code tool becomes more likely. So we asked:

How often have you written code that interacts with software built using a low-code platform?

Results (n=663):

Figure 5



Observation:

The distribution of "have interacted with software built using low-code platforms" and "have myself used low-code platforms" responses is remarkably similar (see discussion above). Further, 81.4% of those who have never built software using a low-code platform have also never interacted with software built using a low-code platform. This is four times the "never interact" rate vs. overall responses — that is, the intersection of "never built" and "never interacted with" results is large.

We interpret these two results to suggest a bubble of non-low-code-platform users exists that is fairly well segregated from software built using low-code platforms. This is an interesting finding that we did not expect.

DESIRE AMONG SOFTWARE PROFESSIONALS FOR DEVELOPMENT-EMPOWERED BUSINESS USERS

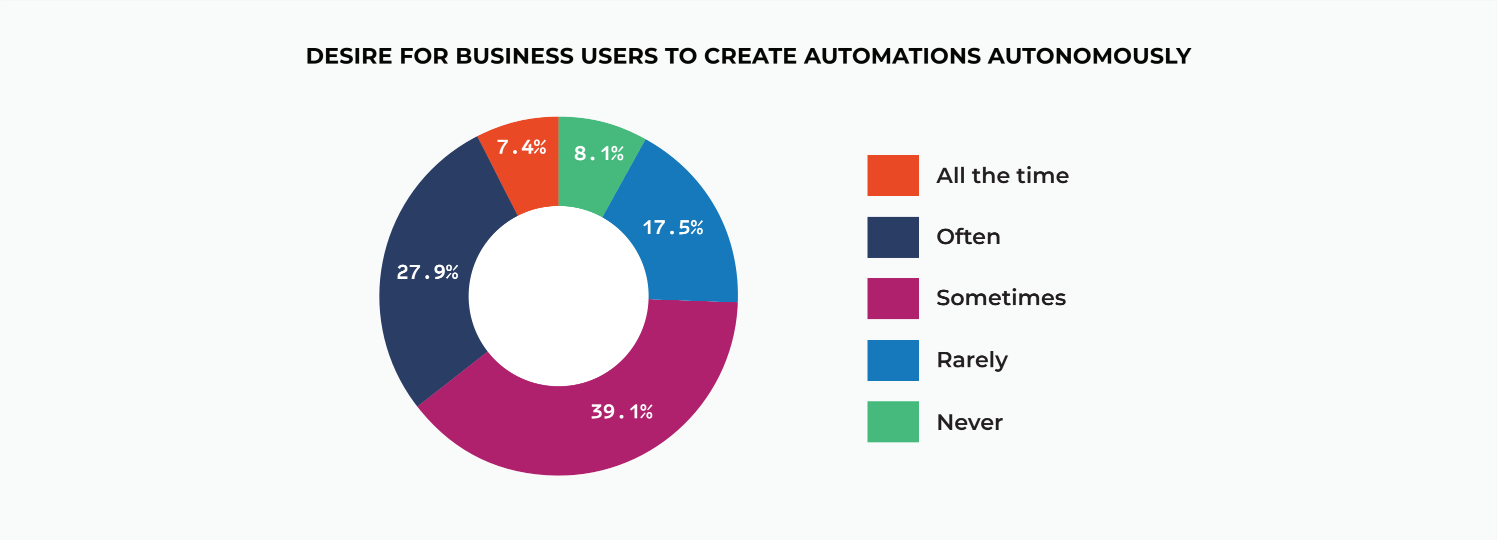
To professional developers, one of the selling points of low-code platforms is that you won't have to write the boring stuff anymore. On the other hand, experience in enterprise software consulting suggests that business users may not know what they want, making a low-code solution built by a business user no more useful than a long specification document written by a business analyst that implementation shows is completely incorrect.

We wanted to see how much software professionals actually do want business users to be able to write their own automations, so we asked:

How often do you wish business users could create simple automations without involving a professional developer?

Results (n=663):

Figure 6



Observations:

1. Software professionals do indeed want business users to be able to create their own simple automations. Only 8.1% (n=54) of respondents do not.

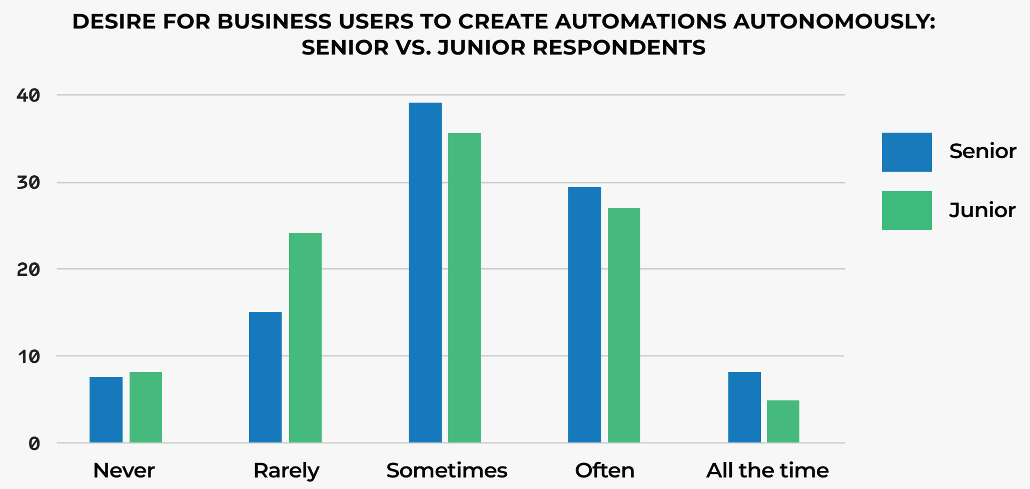
2. The reported frequency of this desire, however, in conjunction with respondents' relatively low fear that low-code tools threaten their jobs (see discussion below), suggests that software professionals do not deeply trust business users to write their own automations correctly. 39.1% and 27.9% of respondents reported that they sometimes or often, respectively, want business users to create their own simple automations, and only 7.4% want this all the time.

3. Senior respondents were significantly more likely to want to empower business users than junior respondents (see Figure 7 on the next page).

   Drawing on our own experience, we account for this difference by hypothesizing that senior-level software professionals are more likely to realize, with the humility of experience, that solutions are often simpler than they first appear. And they are also more likely to have accumulated painful experiences of business users vacillating unpredictably, resulting in many unnecessary rewrites.

   These are two very different causes, and we plan to address both in future research (on SDLC and requirements in general, not for low-code development in particular).

Figure 7



ITERATIVE DEVELOPMENT: UTILITY AND IMPACT OF LOW-CODE PLATFORMS

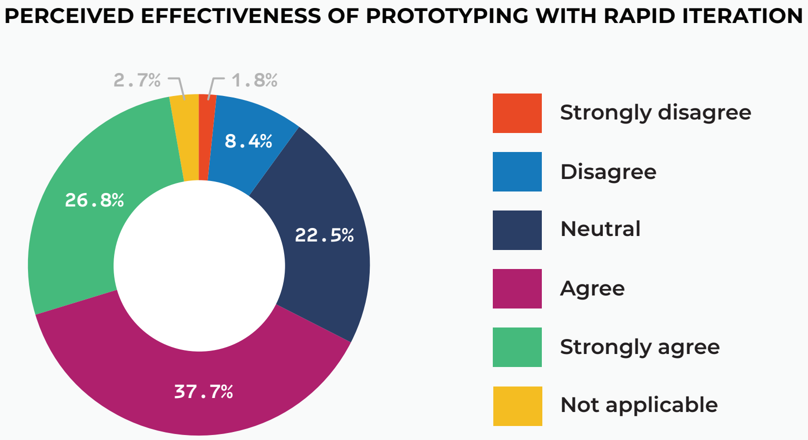
Prototyping and iteration on code can result in higher-quality code, as discussed briefly above. But both also help builders and users work together to figure out what really needs to be done — a reality to which Agile software development methodologies explicitly respond. Because low-code platforms enable rapid prototyping and fast iteration, we wanted to know how valuable these benefits are, in the judgment of software professionals.

So we asked:

Agree/disagree: Prototyping with rapid iteration is usually more effective at gathering requirements than soliciting requirements by interviewing business users.

Results (n=663):

Figure 8



Observations:

1. A majority of respondents agreed (37.7%, n=250) or strongly agreed (26.8%, n=178) that prototyping with rapid iteration is better for requirements gathering than interviewing. Both the assumptions of the Agile Manifesto and this value-add of low-code platforms are validated.

2. Almost nobody strongly disagreed (1.8%, n=12), and less than 10% disagreed (8.4%, n=56).
3. Low-code builders are somewhat more likely to agree with this statement. 41% (n=15) of respondents who have built software using low-code tools agreed (vs. 37.7% among all respondents) and 28.7% strongly agreed (vs. 26.8% among all respondents).
4. Senior respondents are much more likely to strongly agree (30%, n=139) than junior respondents (17.2%, n=22).

   On the assumption that requirements-gathering is the kind of informal activity that particularly benefits from experience, we take this result to mean that junior software professionals are overconfident in the power of interviews to elicit business requirements, and they perhaps would benefit from more prototyping and iteration, whether using low-code tools or otherwise.

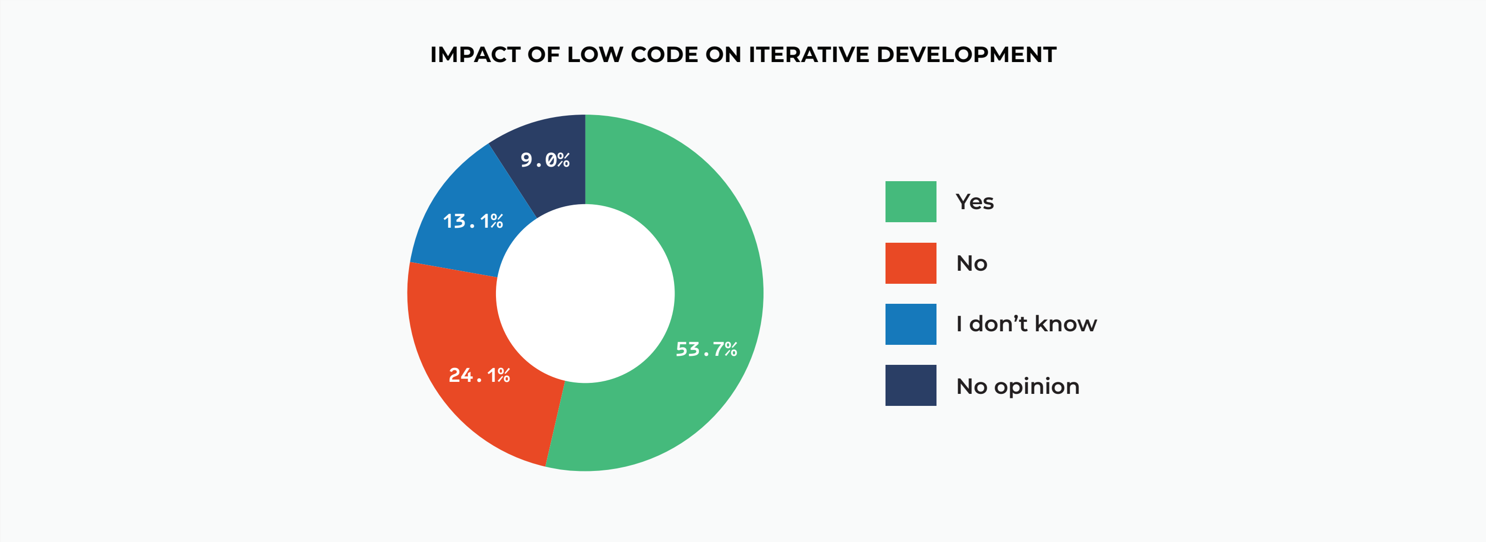
FACILITATION OF ITERATION BY LOW-CODE PLATFORMS

For reasons discussed above, it seems that low-code platforms should enable iterative development. We wanted to know if this holds true in practice. So we asked respondents who have used low-code platforms:

In your experience, did use of low-code tools make development more iterative?

Results (n=564):

Figure 9



Observation:

Low-code platforms may enable iterative development but not as clearly as the promise suggests. A majority (53.7%, n=303) thinks that low-code platforms do make development more iterative, but that majority is not overwhelming. Further, almost a quarter of low-code builders (24.1%, n=136) responded with a positive "no."

We would be more confident in the ability of low-code platforms to facilitate iterative development if the small majority of "yes" answers were paired with a slightly smaller minority dominated by "I don't know" answers, when in fact, only 13.1% responded "I don't know" — a little less than half the "no" responses.

IMPACT OF LOW-CODE PLATFORMS ON TIME TO MINIMUM VIABLE PRODUCT

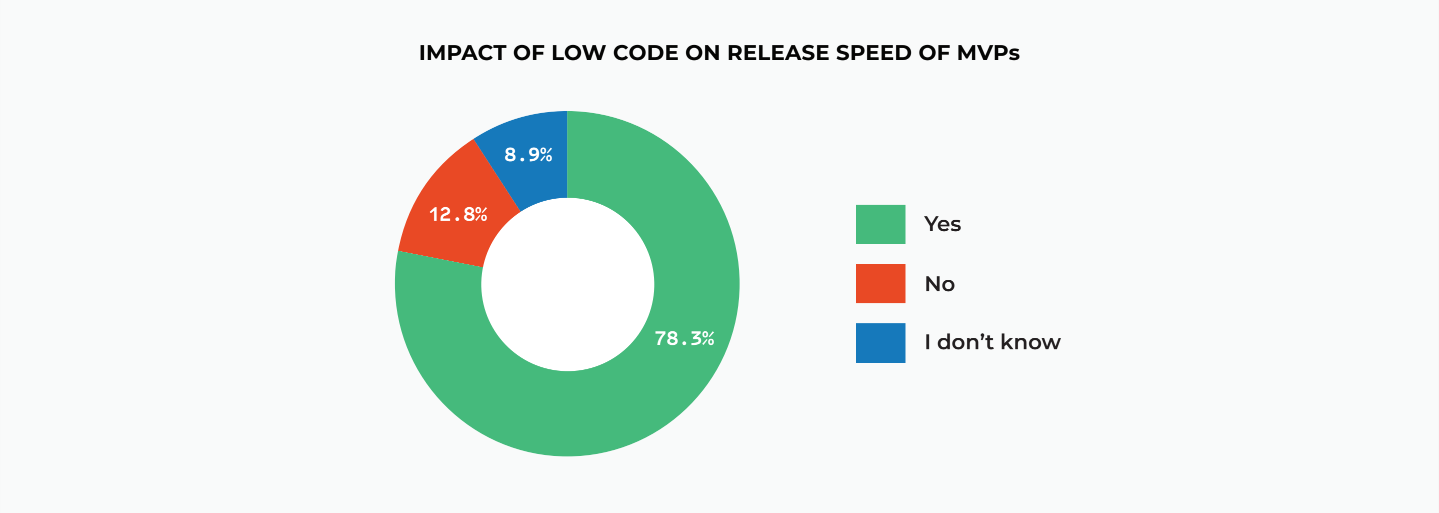
Iterations are not limited to pre-release development. The concept of the minimum viable product (MVP), coupled with the power of the "production crucible," expresses the Agile-friendly desire to release something useful and then make it better with knowledge gained from real-world usage. On this reasoning, faster time to MVP will result in better software, and faster time to MVP was a major reason given for using low-code platforms.

We wanted to see if low-code platforms deliver on this promise, so we asked respondents who have used low-code platforms:

In your experience, did use of low-code tools help you release a minimum viable product (MVP) faster?

Results (n=561):

Figure 10



Observation:

Here, low-code platforms deliver strongly on their promise. More than three quarters of respondents indicated that low-code platforms did help release an MVP faster.

DISTRIBUTION OF TIME SPENT ON LOW-CODE DEVELOPMENT AT EACH STAGE OF THE SDLC VS. FULL-CODE DEVELOPMENT

Some conflicting hypotheses regarding the impact of low-code development on the SDLC:

- Low-code platforms might allow you to spend less time on development, so perhaps they enable more time on higher-level design. On the other hand, abstractions are leaky, and low-code tools cannot handle as much low-level complexity as popular non-visual programming source code, which suggests that writing code requires more up-front design time.
- Low-code systems may tend toward procedural design, which may be harder to reason about, possibly resulting in slower development overall — at least for non-trivial solutions. But if many solutions are trivial in a relevant sense, any procedural tendencies of low-code systems may result in smaller slowdown of complex development than the speedup gained while developing simpler solutions.
- Low-code platforms are often more constrained than full-code platforms, which might result in easier debugging. But code precision can also facilitate better unit and integration test definitions, which could result in less time spent blindly searching for a bug fix.

We wanted to adjudicate among these hypotheses with real-world data. So we asked:

How much time is spent on each of the following tasks while building software using low-code platforms?

And to compare:

How much time is spent on each of the following tasks while building software by writing code?

Results (n=592):

(See Table 4 on the next page)

Table 4

TIME SPENT BUILDING SOFTWARE USING LOW CODE BY TASK

Task	Avg	Min	Max	StdDev	Sum	n=
Design	29.9	0.0	200.0	17.6	17,009.0	569
Development	32.4	0.0	200.0	17.3	18,912.0	584
Testing	25.6	0.0	700.0	31.8	14,865.0	580
Deployment	13.5	0.0	200.0	12.3	7,850.0	581

TIME SPENT BUILDING SOFTWARE WRITING CODE BY TASK

Task	Avg	Min	Max	StdDev	Sum	n=
Design	25.9	0.0	230.0	16.7	15,075.0	581
Development	39.2	0.0	200.0	16.9	23,187.0	592
Testing	22.9	0.0	200.0	13.2	13,460.0	588
Deployment	13.3	0.0	200.0	11.8	7,836.0	588

Observations:

1. On average, low-code development allocates more time to design and testing and less to development than development by writing code. Low-code development spends 4% more time on design, 6.8% less time on development, and 2.7% more time on testing.

   This consistent with our hypotheses: (a) low-code speeds up development, (b) low-code thereby facilitates more higher-level design thinking, and (c) low-code solutions are harder to test.

2. Standard deviation of time spent testing low-code applications (31.8) is a high outlier (vs. 17.6 for low-code design).

   This may be accounted for by combining two claims made in the hypotheses above: When low-code solutions are built well within the intent of their (relatively narrower) constraints, testing is especially easy, and when low-code solutions are not built well within the intent of their constraints, testing is especially hard.

   These two hypotheses are also consistent with the empirical claim, anecdotally asserted here, that low-code platforms are often considerably more powerful than the fat part of the Pareto distribution of intended uses.

Research Target Three: Quality and Maintainability of Software Built Using Low-Code Platforms

Motivations:

1. Professional coders are bad enough at managing complexity. Surely the non-professional coders that low-code platforms enable are even worse.

2. Procedural code is often harder to maintain than more structured code, and low-code platforms tend to define programs procedurally. So low-code software should be harder to maintain than full-code software.

3. Low-code platforms enable relatively easy expansion of functionality, but that tends toward spaghetti, as each new feature is added without reconsidering the overall design. So low-code software probably contains more spaghetti than full-code software.

4. The more happy-path-dependent a set of constraints is, the harder the constrained system is to debug. Low-code platforms tend to constrain users more to happy paths than full-code platforms, so we expect low-code platforms to be harder to debug.

IMPACT OF LOW-CODE TOOLS ON SOFTWARE QUALITY

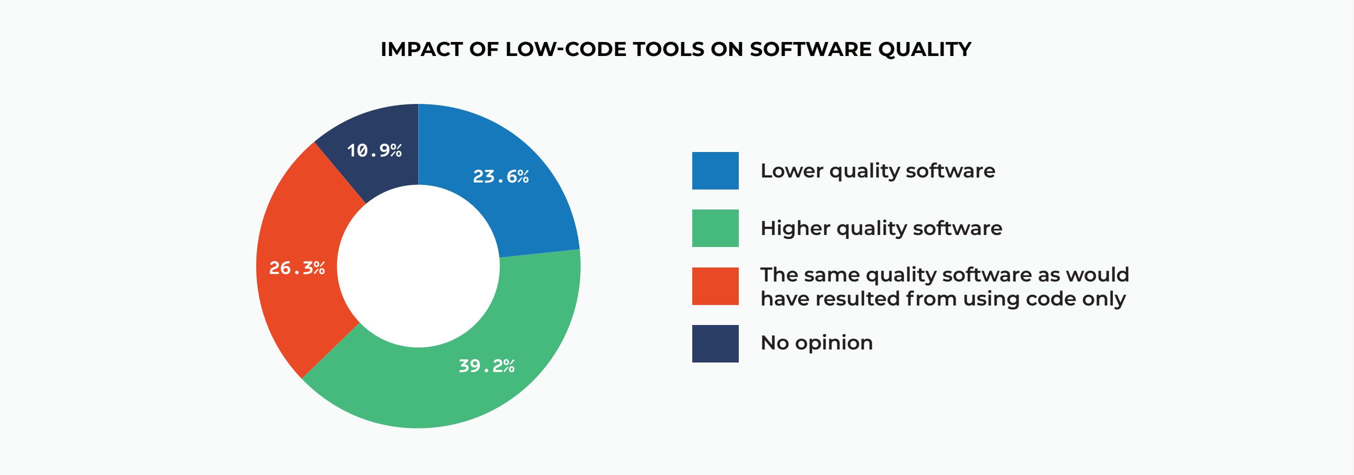
Circumspect professionals care about overall software quality and judge it in a way not necessarily coupled to metrics like bug reports, downtime, cyclomatic complexity, or other results of comparatively immediate feedback. Software professionals' judgment, though subjective, is a valuable mine of intelligent evaluation of software quality.

We wanted to know what those who have used low-code tools think of their impact on overall software quality. So we asked:

In your experience, use of low-code tools resulted in overall: {Higher quality software, Lower quality software, The same quality software as would have resulted from using code only, No opinion}

Results (n=559):

Figure 11



Observations:

1. 39.2% (n=219) of respondents judged that low-code tools resulted in higher quality software. This is a major vindication of the promise of low-code tools — or at least a strong refutation of any knee-jerk dismissal of low-quality tools as inevitably producing bad software.

2. Roughly the same percent of respondents (~40%) agreed that overall working with a low-code platform was a good idea (40.7%) and that the software built using a low-code platform would have been better if a technical architect or professional developer had been more involved (39.6%).

   Together, these results suggest that it might be worth investing in combined projects that both involve low-code tools and higher-level technical inputs from an architect or developer. Two more granular details support the combined team suggestion in complementary directions:

   - 21.2% of respondents strongly agreed that working with a low-code platform was a good idea, making the agree or strongly agree number a significant majority at 61.9%.
   - 27.7% strongly agreed that adding a technical architect or professional developer would have improved the software, also making the agree or strongly agree number a significant majority at 67.3%.

3. 27.7% strongly agreed that adding a technical architect or professional developer would have improved the software, also making the agree or strongly agree number a significant majority at 67.3%.

IMPACT OF LOW-CODE TOOLS ON TECHNICAL DEBT

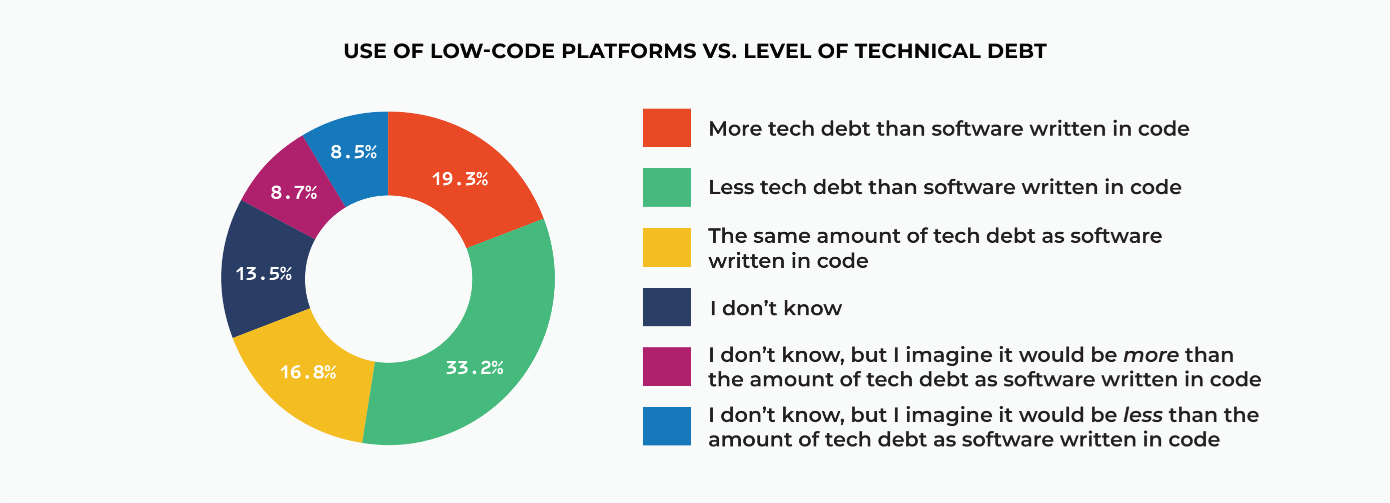
Perhaps the low-code platform lets you build version one; then version two requires something the platform can't do, so all of version one instantly becomes technical debt. Or perhaps the problem can technically be solved using a low-code platform, but the solution is hacky, jammed with epicycles, and brittle, while a full-code solution would be clean and easy to maintain. We wanted to understand how often these kinds of scenarios, which we have observed anecdotally, obtain in widespread practice.

So we asked:

Software built using low-code platforms results in: {More technical debt than software written in code, Less technical debt than software written in code, The same amount of technical debt as software written in code, I don't know, but I imagine it would be more than the mount of technical debt as software written in code, I don't know, but I imagine it would be less than the mount of technical debt as software written in code}

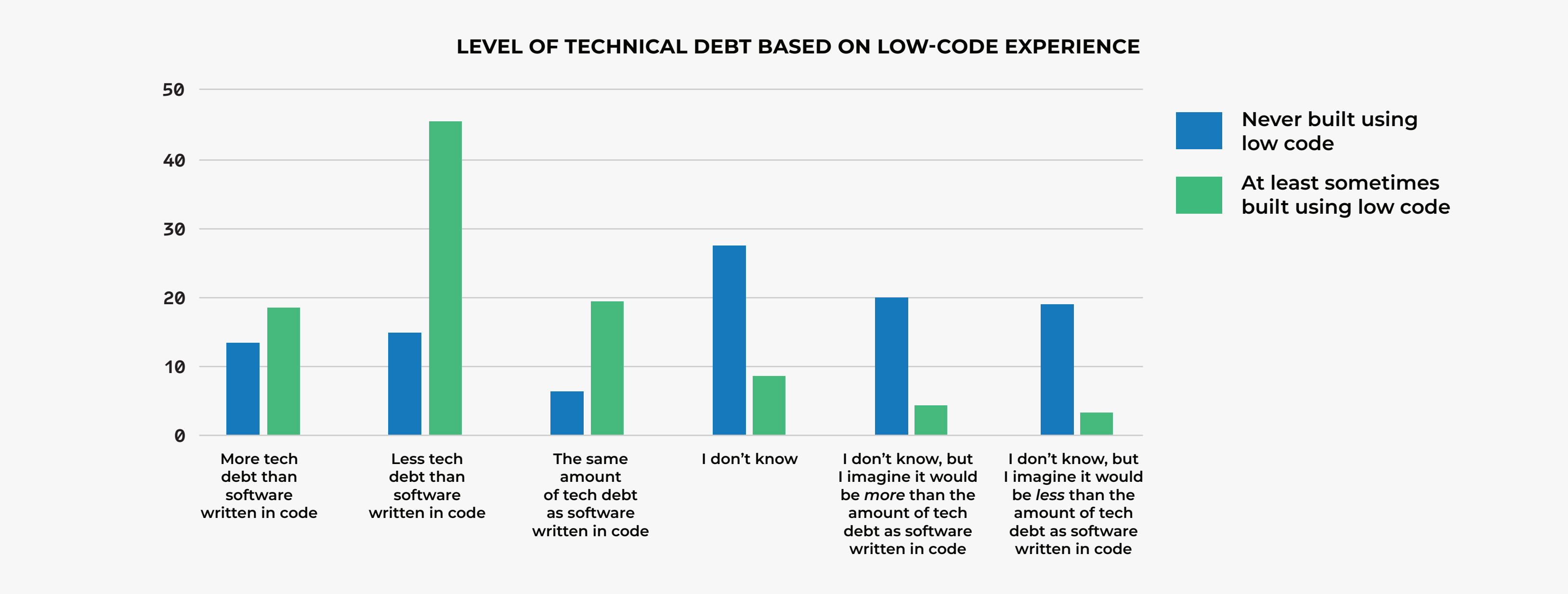
Results across all respondents (n=668):

Figure 12



Compare with responses from those who have actually built software using low-code platforms:

Figure 13



Observation:

In the judgment of low-code-experienced software professionals, the reputation of low-code platforms for high technical debt is not well earned. Almost half (45%, n=165) of low-code-experienced respondents reported that software built using low-code platforms results in less technical debt, and only 18.5% (n=68) of this group reported increased technical debt from low-code tools. This is a strong endorsement of low-code tools for long-term software maintainability.

REUSABILITY OF BUSINESS LOGIC IN SOFTWARE BUILT USING LOW-CODE TOOLS

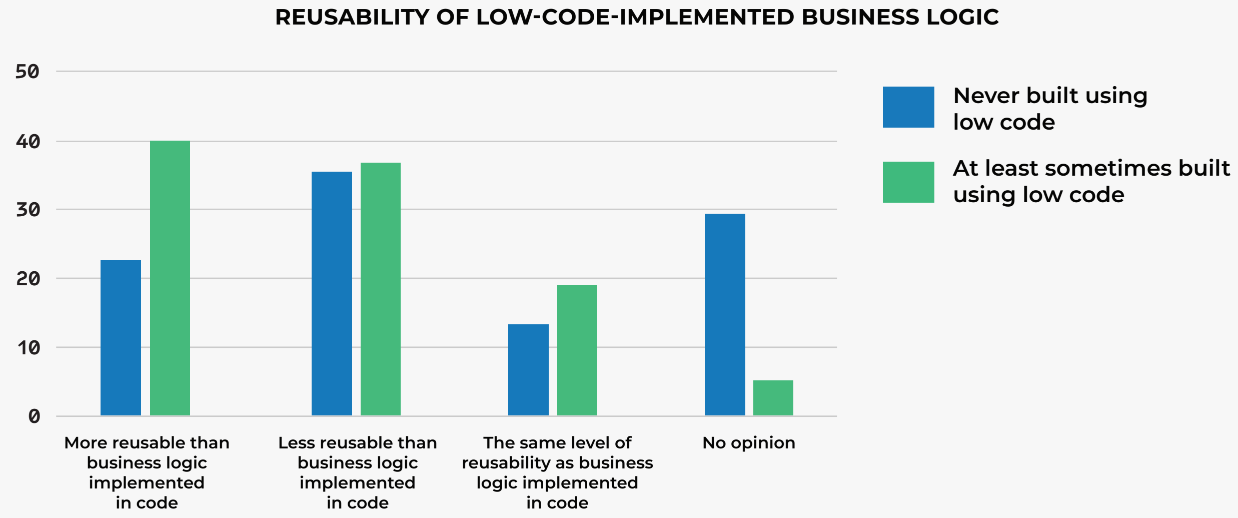
Technical debt is one way of looking at software maintainability — as a negative balance, something of a bean-counting perspective. Reusability is another, more creator-oriented way of looking at software maintainability. We wanted to learn about the quality of software generated by low-code tools from both bean-counter and creator perspectives.

So we asked:

In general, business logic implemented using low-code tools is: {More reusable than business logic implemented in code, Less reusable than business logic implemented in code, The same level of reusability as business logic implemented in code, No opinion}

Results:

Figure 14



Observations:

1. Again, the maintainability of low-code-built software exceeds its reputation. 40.2% (n=147) of respondents who have built software using low-code tools reported that the business logic implemented in these tools is more reusable than business logic implemented in code.

2. Because reusability also implies separation of concerns, these data also suggest that low-code tools can help increase encapsulation and thereby decrease coupling — perhaps the greatest of software design evils.

   This is anecdotally consistent with our experience and points to a helpful side effect of empowering non-developers: Whereas plumbers may be tempted to mix plumbing and porcelain, and may not even notice when they do, non-plumbers will not.

SPAGHETTIFICATION OF SOFTWARE BY LOW-CODE PLATFORMS

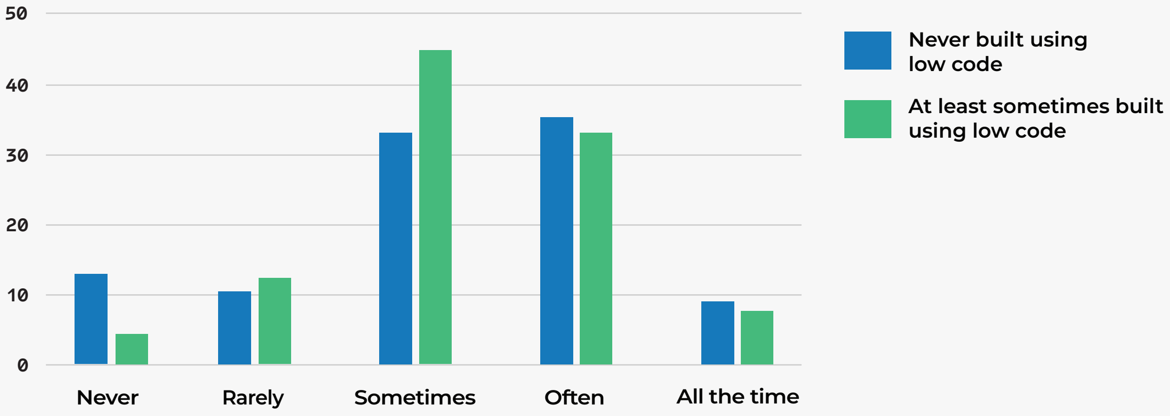
The evil inverse of encapsulation is spaghettification. An exponentially exploding execution graph may not be worth money saved by not hiring a professional coder. Since spaghetti avoidance is a technical, professional, engineering-y activity, we suspect that low-code platforms might tend toward spaghetti, at least when an empowered business user who is not a professional developer uses the tool. We wanted to know if this was actually the case.

So we asked:

Software built using low-code platforms by people who are not professional developers results in spaghetti: {Never, Rarely, Sometimes, Often, All the time}

Results:

Figure 15


PERCEIVED SPAGHETTIFICATION OF SOFTWARE BUILT ON LOW-CODE PLATFORMS

Observations:

1. Results here are mixed. Hardly anyone (3%, n=11) who has built low-code software thinks that low-code non-professional development never results in spaghetti vs. 12% (n=14) of non-low-code-builder respondents. Here, the promise apparently exceeds the reality.
2. On the other hand, negative judgments ("often" and "all the time") are more likely among non-low-code builders than low-code builders. Together with the "never" discrepancy, we suspect that non-low-code builders here are less likely to know what is really the case.

Future Research

Our survey included questions we did not have space to discuss here, which covered:

- Application complexity appropriate to low-code platforms
- Difficulty of debugging low-code vs. full-code applications
- Short term vs. long-term benefits of low-code development
- The utility of low-code platforms in avoiding preventable mistakes (e.g., bad integration practices, sloppy security, low-level coupling)
- The suitability of the title "developer" to low-code application developers

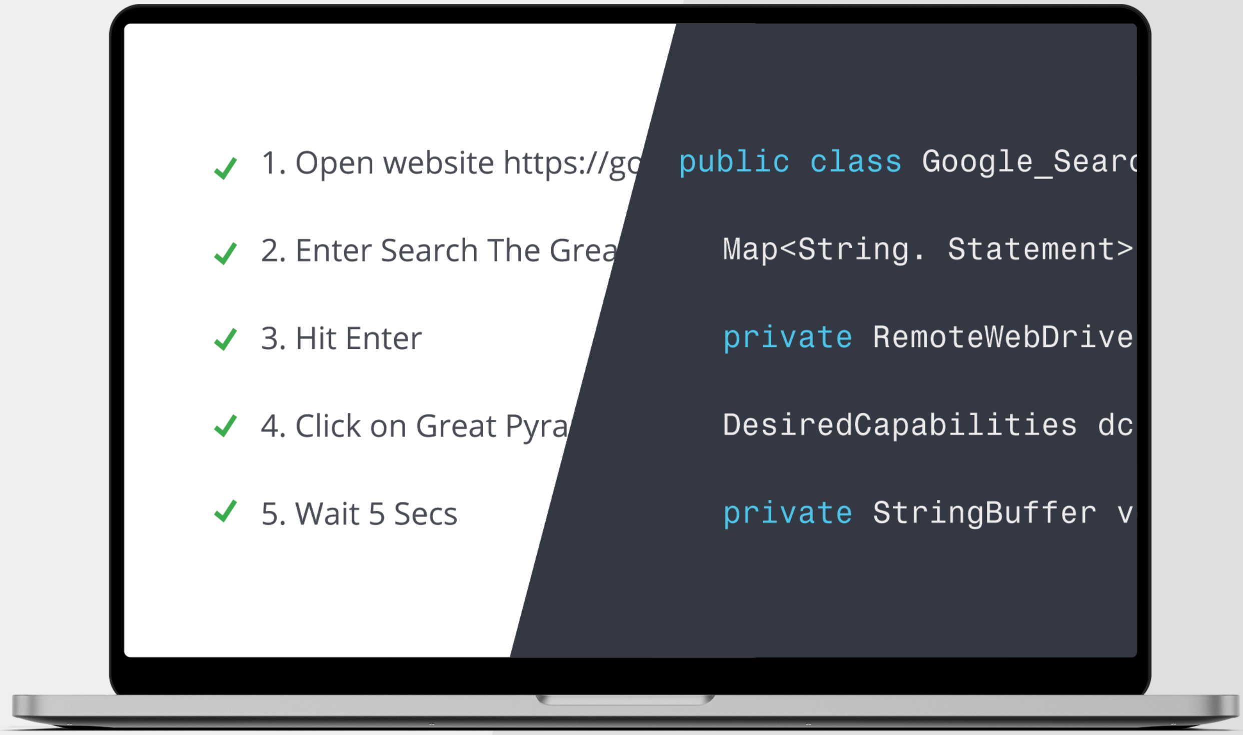
We expect to publish analyses of some of these results elsewhere on DZone.com. Since this is our first survey focused on low-code software development, and since our data show that the prevalence of low-code development is much higher than we expected, we intend to conduct significant additional research on this topic soon.

John Esposito, PhD, Technical Architect at 6st Technologies
@subwayprophet on GitHub | @johnesposito on DZone

John Esposito works as technical architect at 6st Technologies, teaches undergrads whenever they will listen, and moonlights as research analyst at DZone.com. He wrote his first C in junior high and is finally starting to understand JavaScript NaN%. When he isn't annoyed at code written by his past self, John hangs out with his wife and cats Gilgamesh and Behemoth, who look and act like their names.

Let Machines Code Your Tests. You Do The Interesting Stuff.

You create the test case. We generate the scripts. AI-driven end-to-end test automation includes powerful natural language processing capabilitites.



✓ 1. Open website https://go

✓ 2. Enter Search The Grea

✓ 3. Hit Enter

✓ 4. Click on Great Pyra

✓ 5. Wait 5 Secs

```
public class Google_Searc
  Map<String. Statement>
  private RemoteWebDrive
  DesiredCapabilities dc
  private StringBuffer v
```

Intelligent Test Management - Manage testing through constant changes, frequent upgrades and new functionality across end-to-end business processes.

Self-Configuring Reusable Test Assets - Create reusable test components customized for your any web application and add them to new test scenarios with a single click.

API Testing - Increase the breadth, scope, and velocity of your API testing and decrease the time between finding defects and resolution.

Deep-Learning and Machine Learning Capabilities - Detect changes and enable self-healing for test assets giving you critical real-time feedback to ensure long-term durability and scalability.

Learn more

SAUCELABS

CASE STUDY

Case Study: Private Health Care Service

CHALLENGE

Private Health Care Service has ambitious scaling plans, and they rely on their technology platform to help get them there. They needed a process that enables rapid application development (RAD) to replace their modern legacy systems previously supported by their business. It was necessary that the new product had a delightful user experience and was as error-free as possible to compensate for the user base who did not want to migrate from the legacy system.

As a publicly traded company, they are subject to compliance requirements (SOC2, PCI, HIPAA, etc.), and they cannot afford for their valuable patient care data to get corrupted as they rapidly scale. Also, ensuring that their systems are available 24/7/365 with over 99.9% availability was extremely critical. They were looking to hire a large QA team to be able to increase regression testing and reduce the amount of defects sent into production.

SOLUTION

Led by their CIO and with the help of an AutonomIQ Sauce Labs Evangelist, Private Health Care Service's nimble team started down their automation journey with AutonomIQ's AI-powered Codeless studio. As the engine of low-code and no-code testing in the Sauce Labs DevOps Toolchain, AutonomIQ empowered their team to automate their functional and UI testing across their entire application landscape, including their customized CRM system.

Although their core business is technology-enabled, Private Health Care Service is not a native technology company. AutonomIQ provided a solution for their team to adopt Selenium even though they did not have the technical knowledge to write the Selenium scripts themselves.

RESULTS

Private Health Care Service's IT delivery team was also able to:

- Achieve 100% regression coverage
- Increase employee productivity by 300%
- Reduce O&M (operations and maintenance) expenses by 80%

With a headcount of two, Private Health Care Service was able to develop 250+ test scripts (each script containing at least 40 steps) in four weeks, which allowed them to have full regression coverage over their entire technology ecosystem. AutonomIQ also served as a platform for the CIO to complete role-based testing for SOC internal auditing purposes.

COMPANY

Private Health Care Service

COMPANY SIZE

400+

INDUSTRY

Healthcare

PRODUCTS USED

AutonomIQ Sauce Labs Low-Code Studio

PRIMARY OUTCOME

Private Health Care Service achieved 100% regression coverage, increased employee productivity by 300%, and reduced their operations and maintenance budget by 80%.

CREATED IN PARTNERSHIP WITH

SAUCELABS

CONTRIBUTOR INSIGHTS

A Future Without Test Debt

Citizens Thrive With AI-Driven, Self-Healing Low-Code Testing

By Raj Rao, Co-Founder of AutonomIQ

As much as we saw an advancement in the complexity of software systems over the last decade, the tools used to test and deliver software quality did not keep pace. This created a tremendous backlog of candidates that shipped to software quality teams lacking the processes, headcount, and tools needed to bring the accelerated releases to market with quality. We refer to this as *test debt*.

Oftentimes, an immense amount of time and effort is spent during requirement analysis and design to ensure that software applications are compliant and delight customers. Behind the scenes of all this attention, however, is the underappreciated sweat equity invested by the software quality and deployment teams. Initiatives to shorten release cycles as part of agile development or CI/CD pipelines without setting higher quality standards have only exacerbated the impact of test debt.

The costliest form of test debt occurs when an issue is not detected, diagnosed, and debugged prior to production. It is difficult to test in production environments for many reasons — for example, the testing of production data is intrusive and may negatively impact performance. According to Forrester Research, “Firms that run Agile+DevOps as a single transformation are **35 percentage points** more likely to improve technical quality.... **72%** of firms say testers are critical to continuous delivery success.”[1]

Another costly form of test debt involves test flakiness (e.g., false negatives). In Forrester’s “The State Of Agile” report, **44%** of respondents [developers] said that skilled product owners are not available to them for collaboration. It seems that too often, something gets lost in translation from requirements to test case management. Additionally, teams end up spending excessive time on test maintenance, which cannibalizes QA sprints, forcing some teams to ship with bugs as they accept flaky pass rates. Before you know it, a test debt bubble can manifest and create an ongoing cycle of slow- or high-risk development.

If implemented correctly, the AI in modern low-code testing can take on tedious tasks as well as streamline performance, improve end-to-end visibility, make intelligent recommendations, and free engineers to focus on creative work that machines can’t do — or do nearly as well. Let’s explore how AI-driven low-code testing can help your organization solve test debt once and for all.

Measuring the Impact of Test Debt

Until organizations gain executive support for addressing test debt, it will continue to grow and potentially block the ROI of agile development. One of the most important ways to attack test debt is to combine DevOps and agile transformations. While most development professionals in the Forrester report on software quality metrics acknowledge the advantage of Agile+DevOps, only a small portion of respondents said that their organizations were planning to unify Agile+DevOps programs.

Barriers to making Agile+DevOps a reality may be found by looking closer at metrics including bugs, code quality, and churn. It’s vital to collect deeper insights about these metrics and map them to release speed and quality bottlenecks. And they may help you move the dial as it becomes clear that *development and quality belong in the same budgets*.

BUGS

How many bugs are making their way into production, causing an unpleasant customer experience? QA teams should be counting and keeping track of both fixed and unfixed bugs. Tracking the unfixed bugs allows teams to amplify their focus on key aspects of future test automation, and noting the fixed bugs helps teams measure the effectiveness of their test debt management.

CODE QUALITY

Test automation cannot guarantee code, but it can evaluate the quality of software at various levels — from individual components or modules throughout the whole system. Code quality is a measure of how well the code can respond to requirement changes, typically those related to maintainability and performance. Poorly written or poorly maintained code can lead to inefficient use of resources, difficult bugs, security vulnerabilities, and overall operational costs that can increase over time.

Development or QA teams should be tracking unit test coverage and design patterns used in the automation. Ensuring that individual units of code have been tested by at least one unit test increases coverage and reduces the number of errors present in later stages of the SDLC. In addition, taking a qualitative measure of the design pattern and applying a universal approach to all test automation will make it easier for team members to build on the existing library of test assets. Less guesswork means less test debt.

CHURN

You see churn in your test automation when the test scripts get refactored, updated, or completely replaced. Measuring churn helps QA teams recognize the constant level at which test automation assets must be redone. Having an objective view of this will help organizations plan and solve for these challenges more quickly, allowing them to take a more proactive approach in managing test debt.

Managing Test Debt With Always-On Quality Infrastructure

Managing test debt can feel like a burden. Automation engineers, QA teams, and IT delivery managers are already overwhelmed enough. More so, you eventually start seeing your test coverage decline as more test automation accumulates and tests need to be maintained. This happens because as you develop new features, you must keep developing new tests. However, your team can't automate these new tests while they're fixing existing tests, so the team grows dependent on manual testing, which is another huge time sink.

Unleash the citizen tester.

A low-code test automation tool can make a citizen tester (business analysts, administrators, project managers, and product owners) an engineer's best friend. Or it can allow a citizen developer to do their own testing. Meanwhile, engineers gain the help they really need to reduce test debt, accelerate innovation, and increase efficiency across the entire SDLC.

As the market for elite knowledge workers tightens month by month, business leaders are looking for a more manageable growth strategy. In 2020, the U.S. Bureau of Labor Statistics calculated that the unemployment rate of software professionals was 1.9%.[2] Competitive advantage in this territory will go to the organizations that drive their variable cost structures to zero, utilizing low-code technology while also incrementally scaling their coverage and reducing their exposure to change disruption.

AI-enabled automation helps achieve greater coverage and generates more test outcomes (more bugs detected, more code fixed). You can test more thoroughly and leverage the expertise of your product owners and business analysts. Far fewer errors slip through to production as a result.

Marrying Development and Testing to Divorce Test Debt

Development and testing must become parallel processes as early as during the design stage, which requires seamless collaboration between all stakeholders who need AI-driven automation to work together smartly and quickly. In this way, your organization can adopt the best parts of Agile and DevOps swiftly and cost-efficiently, enabling modern approaches to development that accelerate releases without increasing risk. These approaches include rapid application development (RAD), test-driven development (TDD), and behavior-driven development (BDD).

RAPID APPLICATION DEVELOPMENT

Low-code test automation supports heightened agility when developing and deploying new software. Using the power of AI, RAD helps to transform testing from a bottleneck to an accelerator since tests can be executed in-sprint.

In a low-code-enabled setting, although there is still an unprecedented demand for testing, the testing team is not waiting for headcount approvals in a shrinking pool of available testers. Citizen testers can leverage the software delivery team's expertise and train to a point where they can start delivering business value immediately — and not at the risk of software quality. Also, given the intelligent nature of low-code test automation, the AI knows what to test and can make recommendations before certain test assets begin to break.

TEST-DRIVEN DEVELOPMENT

Fail-first approaches to development such as TDD were created because of the incredible competitive advantage that can come from teams that fail early. In TDD, early failure reduces such risk later in the lifecycle when bugs are much harder to detect and costlier to repair. Early failure can also drive more efficient, effective iteration with built-in continuous improvement, which is especially useful for iterating through exploratory product development or working on systems of innovation.

A critical factor in optimal TDD performance is ensuring that test cases capture testing intent accurately — and within the sprint's scope of work. At this first and all-important stage of TDD, collaboration becomes key to success. UX, UI, and API development stakeholders cannot be sucked into prolonged (manual) testing. While complex builds may always involve an element of manual testing, iterative development must automate far more testing, allowing teams more time to make sure they don't fall short of the user story or miss creative opportunities to innovate great ideas.

Yet many teams are not skilled in test automation despite recognizing the need to make it work well to achieve quality at speed when shipping products in much shorter cycles. Many blockers to test automation success — from seeing ROI to reducing risk/bugs in production environments — stem from a central problem in the first step of a TDD approach: Ensure that **intent** with proper **domain knowledge** gets incorporated into the build.

Modern low-code test automation addresses this core problem by using powerful AI and natural language processing (NLP) technologies to integrate intent into TDD workflows. Consider the following quality strategies:

- **Use your words** – Developers of all types, including citizen developers (e.g., Salesforce admins), collaborate better with product owners or non-technical testers by chatting or speaking in plain English. **NLP** capabilities on low-code testing platforms such as AutonomIQ allow any user to create test cases that accurately capture intent by speaking normally.
- **Iterate much? Then test case much** – Modern apps and microservices all run on APIs, mostly back-end APIs that deal with huge numbers of integrations with public, partner, and internal endpoints that can individually or holistically affect function. Small, iterative changes to integrations may cause a domino effect that impacts many test cases, requiring a significant proliferation of regression testing. Low-code test automation platforms should offer **self-healing maintenance** of tests and test cases, ensuring that builds can be shipped without a testing bottleneck.
- **Let automation write its automation scripts** – Many development teams lack the on-hand developers or automation engineers needed to continually write and maintain automation scripts for testing workflows. AI-driven low-code testing can take NLP-generated test cases and automatically generate test scripts for automation. With AI-driven automation writing its own automation scripts for UI and API verifications and end-to-end validations, testing times can be cut in half or better while significantly reducing risk in production environments and during beta testing.

BEHAVIOR-DRIVEN DEVELOPMENT

One type of TDD that has become top of mind for many high-velocity development teams is BDD. This approach tries to focus TDD flows on addressing system behavior in hopes of creating **functional tests** quickly without losing technical and non-technical domain knowledge throughout collaboration on very complex problems.

Cucumber is one of the most popular open-source frameworks for BDD, requiring test case generation in English that, however, must follow the Gherkin syntax, which is optimized for automation tests that check on behavior tests void of logic details. In this way, Gherkin removes ambiguity from executable specs and plugs into the Cucumber framework, which principally, attempts to optimize TDD by identifying what to test, what not to test, and how much testing is needed within scope. However, as noted, a critical gap in Cucumber's approach to BDD is that it can be inefficient and ineffective to build intent into test cases when plain English must be converted into the Gherkin syntax.

Conclusion

Hiring your own AI talent is difficult and expensive. Pre-packaged AI-enabled tools solve this problem in cost-effective and scalable ways. Their effectiveness relies on how well they enable collaboration, reduce technical debt, and avoid disrupting familiar workflows.

Quality at speed is ultimately driven by making everyone a developer and every developer a tester. AI-driven low-code testing removes human error from repeatable processes, improves end-to-end visibility, makes intelligent recommendations, and blurs the line between departments with the democratization of very technical tasks. Development and testing must become parallel processes, and those that embrace AI-driven low-code development will gain a considerable competitive advantage.

[1] Forrester Research for Tricentis, "The Definitive Software Quality Metrics For Agile+DevOps: Measuring The Risk Of A Release Candidate"

[2] National Foundation for American Policy, NFAP Policy Brief, June 2020 (Table 2)

Raj Rao, Co-Founder of AutonomIQ, a Sauce Labs Company

@rajraoaiq on DZone | @rajrao1 on LinkedIn

Raj Rao currently is responsible for AutonomIQ product management, building industry solutions and customer evangelism. Prior to AutonomIQ Sauce Labs, Raj has held SVP & GM roles at CA Technologies, has been the Global Head of Software Quality at NTT Data, and has held product leadership roles at SAP, PeopleSoft (Oracle), and Broadvision.

CONTRIBUTOR INSIGHTS

Low-Code Automation Myths Debunked

By Eric Schabell, Portfolio Architect Technical Director at Red Hat

Introduction

As a professional developer, you are quite familiar with the concept of coding applications and building out the infrastructure to deliver that code, and you understand the complexities involved when a team works to meet delivery deadlines. You've spent years becoming proficient at your calling, and then suddenly one day you're confronted with a new technology platform referred to as a low-code automation platform. It's here to deliver solutions with a minimal amount of coding by developers and provides the flexibility for business users to develop their own solutions.

You might think this story above won't happen to you, but the latest low-code magic quadrant report from Gartner states otherwise:

"By 2023, over 50% of medium to large enterprises will have adopted an LCAP as one of their strategic application platforms."

Time to face the facts — sooner or later you're going to be faced with a low-code automation platform. As a developer, there are a few myths around low-code platforms, and if we're honest with ourselves, anything that tries to provide solutions without professional developers is concerning, right?

What Is Low-Code Automation?

Before we get too carried away worrying about these platforms, let's take a closer look at what exactly a low-code automation platform does. While there are simple definitions to be found around low-code automation platforms, for the purpose of this article, we'll combine a few to define low-code automation platforms for the rest of this article as follows:

"Low code is a software development approach requiring minimal coding to build applications and processes. A low-code automation platform uses visual tooling with simple drag-and-drop features and configuration instead of extensive coding languages."

This simple definition creates a baseline that low-code automation is a platform of visual and configuration tooling that lets a user design applications or processes. The results generate a running solution with little professional developer intervention.

Low-Code Myths

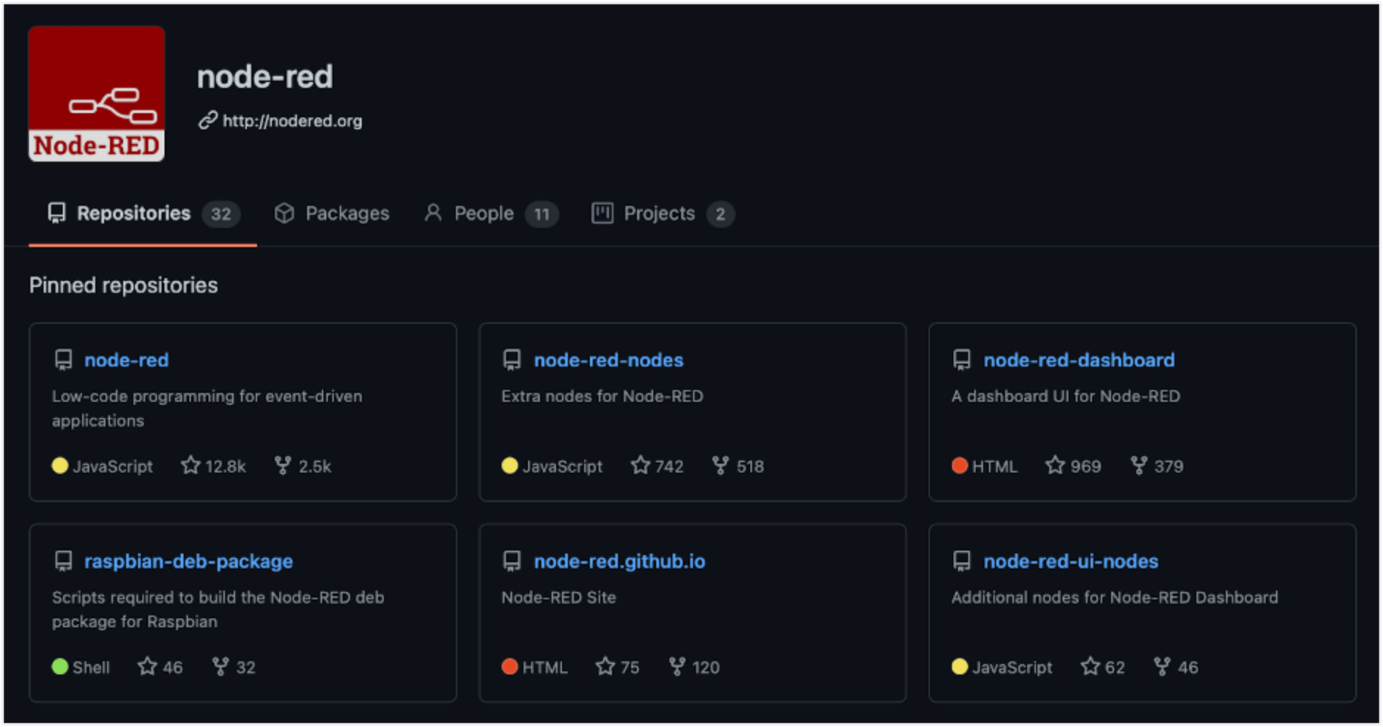
To understand what low-code automation stirs in professional developers, we'll take a look at a few of the more common myths surrounding it. These myths could just as easily be called fears or deep concerns. The following are a few common myths for developers that you might hear on a break by the coffee machine.

LOW-CODE AUTOMATION ELIMINATES THE NEED FOR PROFESSIONAL DEVELOPERS

The first reaction to hearing that low code is coming to your organization is that it's going to mean a reduced need for professional developers. Nothing could be further from the truth. While the way that applications and processes are constructed will change with less of a focus on the nuts and bolts of programming, the resulting generated artifacts continue to need attention. Professional developers remain crucial to managing application development at scale, integration of components, and general continuity of the application development landscape over time. A general shift in focus for the developer is the natural evolution with the occasional dive into the traditional coding arena to ensure low-code-generated projects are successful.

LOW-CODE AUTOMATION ACCELERATES DEVELOPMENT SPEED

Along the same lines as the previous myth, this one seems like the natural result of using low-code automation. The reality is that initial speed gains on the front side of projects where drag-and-drop designing delivers components faster balances out with the customizations and enhancements needed to ensure the applications meet your needs. Professional developers will be kept busy ensuring that the business users of the low-code automation platforms are seeing the desired end results in their applications.



node-red

http://nodered.org

Repositories 32 | Packages | People 11 | Projects 2

Pinned repositories

node-red
Low-code programming for event-driven applications
● JavaScript ☆ 12.8k 2.5k

node-red-nodes
Extra nodes for Node-RED
● JavaScript ☆ 742 518

node-red-dashboard
A dashboard UI for Node-RED
● HTML ☆ 969 379

raspbian-deb-package
Scripts required to build the Node-RED deb package for Raspbian
● Shell ☆ 46 32

node-red.github.io
Node-RED Site
● HTML ☆ 75 120

node-red-ui-nodes
Additional nodes for Node-RED Dashboard
● JavaScript ☆ 62 46

Image source: GitHub project page Node-RED

LOW-CODE PLATFORMS ARE PROPRIETARY

For those used to the open-source model of development, fear of vendor lock-in due to proprietary low-code automation platforms is a real thing. It's even more of a concern in tooling that generates applications from closed source and proprietary methods that are not easy to come to grips with.

As always, where there is a need, open-source projects rise to fill the hole. While there are many variations and projects in the low-code open-source arena, there is one option highlighting that open source can be an option. This open-source project, shown in the figure above, provides tooling for wiring together hardware devices, APIs, and online services in new and interesting ways. Furthermore, it's open to extensions, new features being added, community support, and all the freedom one expects from an open-source project.

LOW-CODE PLATFORMS ARE NOT FLEXIBLE

Whether using an open-source or proprietary low-code automation platform, the impression many developers have is that generated code components are not flexible. One expects generated components to need modifications, customized business logic, or integration additions to ensure they comply with existing API standards in the organization.

There's always a need to modify generated components or to add integration logic to the application, ensuring conformity with existing architectural choices. It's not very often that you get to build applications and deploy them into a brand-new architecture due to legacy platform choices, existing product issues, and other architectural problems. This is where professional developers shine, ensuring that the customization of applications or integration points can be done in a consistent, secure, and scalable fashion.

Conclusion

While the idea of reducing the amount of coding being done by professional development teams is strange at first, the myths discussed in this article show that things are not as bad as one might think. The job of a professional developer is not going away — it's evolving into something that just might become more productive using the tools low-code automation platforms provide. In some cases, development might accelerate with low-code tools, but in others, it's going to take a professional developer with the insights to understand how to manage the integration, complexity, and growing solution architecture that low-code automation platforms deliver.

Using sound judgement and hard-won experiences, professional developers can ensure that generated solutions can be managed, extended if necessary, and all done in a safe, scalable fashion. An important fact is understanding that the use of open-source technologies does not have to end with low-code automation platforms. There are many options in the open-source ecosystem for you to adopt into your development toolbox during your low-code journey.

Eric D. Schabell, Portfolio Architect Technical Director at Red Hat

@eschabell on DZone | @ericschabell on Twitter | www.schabell.org

Eric is Red Hat's Portfolio Architect Technical Director. He's renowned in the development community as a speaker, lecturer, author, and baseball expert. His current role allows him to share his deep expertise of Red Hat's open-source technologies and cloud computing. He brings a unique perspective to the stage with a professional life dedicated to sharing his deep expertise of open-source technologies and organizations.

CONTRIBUTOR INSIGHTS

Python and Low-Code Development

Smooth Sailing With Jupyter Notebooks

By Steven Lott, Writer, Python Guru & Retiree

If you ride on a sailboat in a steady breeze, it glides through the water effortlessly. Few things can compare to crossing a bay without the noise and commotion of a thundering internal combustion engine.

The dream of no-code and low-code development is to effortlessly glide from problem to solution. Back in the '80s, no-code/low-code development was called "end-user computing." Since the invention of the spreadsheet, we've had a kind of low-code computing. The technologies continue to evolve.

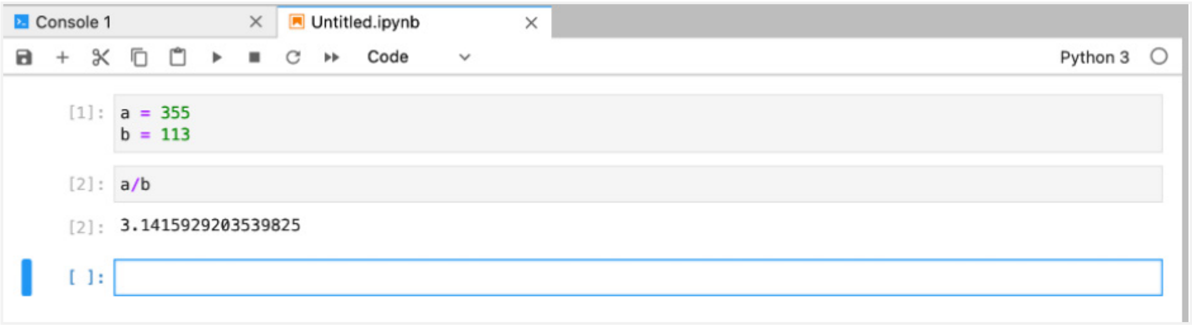
Let's take our boat out of the slip and sail around a little. We'll look at two ways Python is commonly used to create no-code and low-code solutions to problems.

The first thing we'll look at is using JupyterLab to create solutions to problems with minimal code. I liken this to using winches to help lift the sails. It's not an internal combustion engine, but it is a machine that helps us manipulate heavy, bulky items like sails and anchors. The second thing we'll sail past is using Python as an integration framework to knit together tools and solutions that aren't specifically written in Python themselves. In this case, we're going to be writing integration code, not solution code. This is how sailboats work; given a hull and some masts, you'll have to select and set the sails that will make the boat move.

JupyterLab

As a developer, and as a writer about Python, I rely on JupyterLab a lot. It's often the tool I start with because I get handy, pleasant spreadsheet-like interaction. The "spreadsheet" feature that I'm talking about is the ability to change the value of one cell, and then recompute the remaining cells after that change. This lets me create elegant, interactive solutions where the bulk of the interaction is handled by the notebook's internal model: a lattice of inter-dependent cells.

Figure 1



```
[1]: a = 355
     b = 113
[2]: a/b
[2]: 3.1415929203539825
[ ]:
```

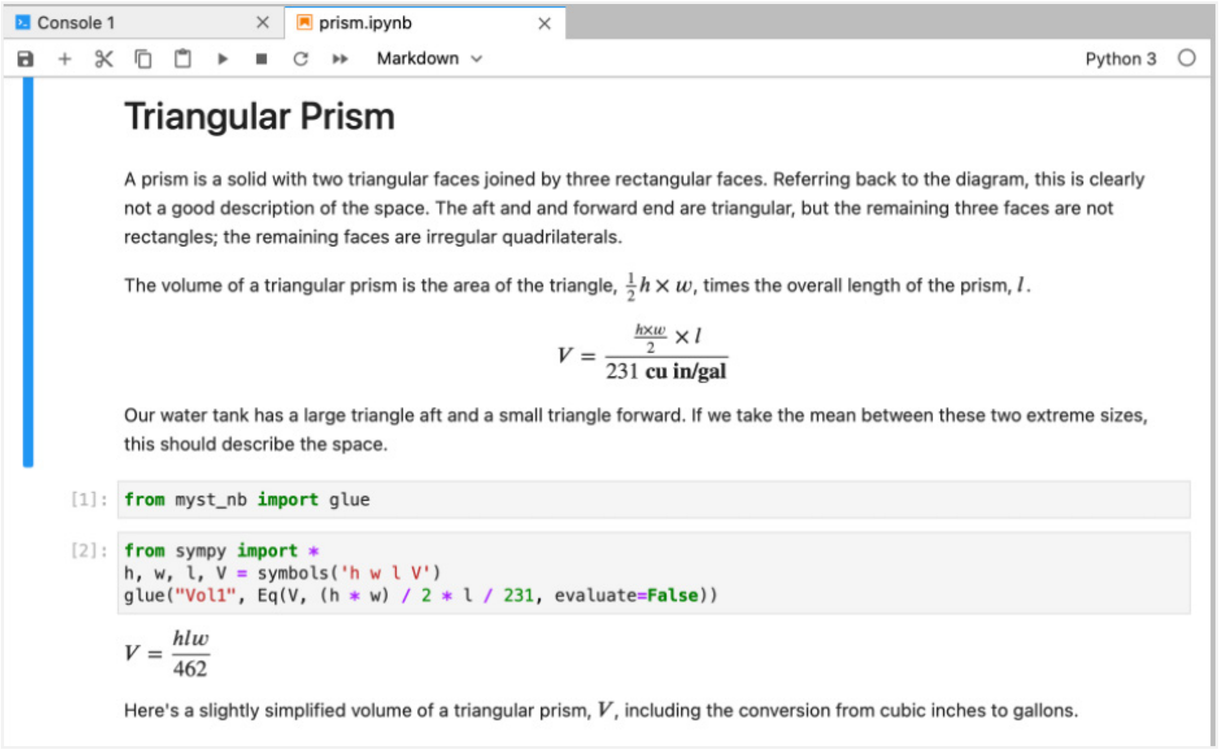
In this picture, we can see the computation of cell 2 depends on prior results in cell 1. I consider this "low code" because there's a vast amount of code we don't need to write. The Jupyter Notebook provides us an interactive framework that's robust and reliable. More important than that, the framework fits the way a lot of people have grown to use computers by putting values into some cells and checking results in other cells. The big difference between Jupyter and a spreadsheet is Jupyter lets us write extensions and expressions in Python.

Recently, I had a boat-related problem crop up that was perfect for this kind of low-code processing. The tank under the pointy part of the boat (the "V-berth") has a fairly complex shape. The question really is: "How big is it?" The problem is access; I have to make a series of approximations and models.

I really need a spreadsheet-like calculator, but I need a lot more mathematical processing than is sensible in a spreadsheet. Consequently, let me apologize to any math-phobic readers. This example involves a lot of math. For folks who don't like math, think of using a spreadsheet where you never looked at the formula in a given column. A notebook can (and often does) hide the details. This specific example doesn't try to hide the details.

Here's an example: https://github.com/slott56/replacing-a-spreadsheet. I want to lift up a few key features of this low-code development approach. I created three notebooks, each of which had a common structure. They all have a collection of measurements as the input. As the output, it reports a computed volume. The rest of the cells provide some background to help me make sure the math is right.

Figure 2



Triangular Prism

A prism is a solid with two triangular faces joined by three rectangular faces. Referring back to the diagram, this is clearly not a good description of the space. The aft and and forward end are triangular, but the remaining three faces are not rectangles; the remaining faces are irregular quadrilaterals.

The volume of a triangular prism is the area of the triangle, $\frac{1}{2}h \times w$, times the overall length of the prism, l.

$$V = \frac{\frac{h \times w}{2} \times l}{231 \textbf{ cu in/gal}}$$

Our water tank has a large triangle aft and a small triangle forward. If we take the mean between these two extreme sizes, this should describe the space.

[1]:

```
from myst_nb import glue
```

[2]:

```
from sympy import *
h, w, l, V = symbols('h w l V')
glue("Vol1", Eq(V, (h * w) / 2 * l / 231, evaluate=False))
```

$$V = \frac{hlw}{462}$$

Here's a slightly simplified volume of a triangular prism, V, including the conversion from cubic inches to gallons.

Here's the input cell's content in the "prism.ipynb" notebook. This is cell 8.

```
measured = {
    # Forward triangle, in inches
    "h_f": 8,
    "w_f": 10 + Rational(1, 2),

    # Aft triangle, in inches
    "h_a": 27,
    "w_a": 48,

    # Overall length from forward to aft, in inches.
    "l_fa": 46,
}
```

This is a Python dictionary that has a number of input values. This is a bit more complex-looking than spreadsheet cells, making it solidly "low-code" not "no-code."

The output is computed in cell 10, providing a single numeric result with a format like the following:

```
'50 85/88 gallons'
```

This shows me that the volume of space, given the measurements, is just shy of 51 gallons.

The best part about this data is there are two kinds of changes I can make. The most important change is to the measurements, which leads to recomputing the notebook. Anyone who can use a spreadsheet can alter the numbers to the "measured" dictionary to recompute the volume.

The other change I can make is to adjust some of the underlying assumptions. This is a more nuanced change to the model that is also implemented in the notebook. I find that for a wide variety of problems, a notebook is the place to start. It lets me gather data, formulate alternative approaches to algorithms and data structures, and work out candidate solutions that are richly interactive.

The JupyterLab project is like a boat with three masts and dozens upon dozens of sails for all conditions and situations. There are a lot of features that can be used to create interactive solutions to problems. The idea is to write only the code that's directly related to the problem domain and leverage the notebook's capabilities to present the solution so that people can make decisions and take actions.

In addition to the notebook for a low-code interactive user experience, we can look at Python as an engine for integrating disparate applications together.

Python as Integration Engine

Python's standard library includes modules to help us work with OS resources, including other applications. Rather than build or modify existing code, we can treat applications as opaque boxes and combine them to create an integrated solution. To a limited extent, integration applications is what shell scripts do. There's a world of difference, however, between integration with a shell script and integration with Python. Shell scripting involves the almost impossible-to-understand shell programming language. (See this article on replacing shell scripts with Python for more thoughts on this.)

When we integrate applications with Python, we can easily introduce additional computations and transformations. This can help to smooth over gaps, remove manual operations, and prevent potential errors.

I'm a fan of code like the following:

```
command = [
    "markdown_py", "-v", "-e", "utf-8"
]

temp = options.input.with_suffix(".temp")
output = options.input.with_suffix(".html")
params = dict(CSS=str(options.style), TITLE=options.title)

with temp.open('w') as temporary, options.input.open() as source:
    subprocess.run(command, stdout=temporary, stdin=source)
```

This is part of a larger and more complex script to publish a complex document written in markdown. It has a lot of code examples, which are a lot easier to read in HTML format. I must do some pre-processing of the markdown and some post-processing of the HTML. It seems easiest to execute the `markdown_py` command from inside a Python script, avoiding a complex python-bash-python kind of process.

Since I'm not modifying the underlying applications, I find this fits with a low-code approach. I'm using the source application (`markdown_py`) for the thing it does best — adjusting the inputs and outputs using Python.

Conclusion

We can use Python in a variety of ways. It's a programming language, so we can build code. More importantly, we can use the vast number of pre-built Python libraries to create low-code solutions. We can use a Jupyter Notebook as a low-code way to create a sophisticated interactive experience for users, and we can use Python to integrate other applications.

Sailing isn't effortless. The boat glides only when the sails are set properly, and we keep the rudder in the right position. Just as skill and expertise are required to make a boat move, so too is careful attention needed to write the minimal Python code to solve an information processing problem.

Steven Lott, Writer, Python Guru & Retiree

@slott on DZone | @s_lott on Twitter | https://slott-softwarearchitect.blogspot.com

Steven has been programming since the 70s, when computers were large, expensive, and rare. As a former contract software developer and architect, he worked on hundreds of projects from very small to very large. He's been using Python to solve business problems for over 20 years. His titles with Packt Publishing include Python Essentials, Mastering Object-Oriented Python, Functional Python Programming, Python3 Object-Oriented Programming, and Python for Secret Agents. Steven is currently a technomad who lives in various places on the east coast of the US.

CONTRIBUTOR INSIGHTS

What Is Low- and No-Code Development?

Understanding the Benefits, Challenges, and Leading Use Cases for Low Code

By Zachary Blitz, Business Analyst at Warburg Pincus

Low-code and no-code development is the method for enabling the automation of business processes for both technical and non-technical developers. It allows for quick setup and deployment of applications for both small businesses and enterprise-grade companies. Individuals do not require technical skills to improve processes or create lasting value for an organization.

There have been many instances in the world today in which businesses and individuals have an idea for an application, but they can't develop it themselves because they aren't proficient in coding. They can hire a software developer to build it for them, but oftentimes that path is too expensive. This creates a bottleneck that has prevented businesses and individuals from creating value for their customers and consumers. However, with the evolution of low-code and no-code platforms, this bottleneck can be solved.

Low-code platforms enable programmers and non-programmers to develop completely customized applications using minimal lines of code, while no-code platforms enable applications to be built without writing a single line of code. Through graphical user interfaces and model-driven logic, a user can build app components using methods such as drag-and-drop. Data between different systems can be intertwined without using an API or running into any technical barriers. In addition to creating applications, low-code and no-code platforms can also be used to launch websites and build workflow automations.

Benefits and Challenges

Like all technology-related decisions, there are both benefits and challenges associated with using low-code and no-code tools in your organization. However, the scalability and immediate change these platforms enable outweigh many of their disadvantages.

BENEFITS

According to Appian, a few benefits of using low-code platforms include higher productivity, decreased costs, better risk management, and ease of use. Organizations do not have to be reliant on just technology groups to improve processes. Workflows can be scheduled by super-users across non-technical groups in an organization, therefore, allowing for faster adoption of process automation.

Benefit	Description	Example
Higher productivity	• More applications can be built in less time • Manual processes are transitioned to automated processes	Consolidating data from different Excel files with the same format into one summary report
Decreased costs	• Does not require hiring more full-time developers or employees • One vendor can provide solutions across an entire organization	Enabling highly skilled workers to create reports versus hiring consultants

(Table continues on the next page)

Benefit	Description	Example
Better risk management	• Data quality checks can be implemented • Process improvements can help reduce human error	Creating a workflow to check data quality of a system
Ease of use	• Non-technical workers can build applications from inception to completion • Functionality includes drag-and-drop	Designing a website mockup
Improved agility	• A change in process does not require a developer • Teams can better respond to customer and regulatory needs	Automating a repetitive task such as scheduling emails can be done through a low-code platform

CHALLENGES

On the contrary, a few notable disadvantages include vendor lock-in, workflow maintenance, and initial permission access issues. Once software gets included within an organization's application ecosystem, there becomes downstream application dependencies of that software. Certain processes become reliant on that software and switching costs enormously rise.

Challenge	Description	Example
Vendor lock-in	• Processes that are reliant on a single vendor increase the costs associated with switching vendors and solutions • Your source code is tied to a single vendor	Implementing software that captures data used for reporting makes it hard to switch to a different software
Workflow maintenance	• Any changes to your systems could have downstream effects and break a workflow • Non-technical users may not understand how to debug a workflow	A faulty email address could cause an error on an email scheduling tool, requiring someone without knowledge of the application to debug
Initial permission access issues	• Any workflow that touches a new system must be given permission, which: – Can create a time bottleneck – Needs approval by every system administrator	Anyone trying to retrieve data from a database needs permission to that system

The amount of time savings that can be created from low-code platforms outweigh their disadvantages. Making a website from scratch, designing a data quality check, and automating reporting are just a few examples of tasks that will make one think why they didn't switch to a low-code platform sooner. All the time saved can now be spent on more strategic initiatives instead of manual, repetitive tasks.

Use Cases for Low- and No-Code Development

The goal of this section is to show you how to bring the benefits described in the earlier section to your organization. In order to understand these benefits, readers will find the practical application of these benefits extremely helpful. By reading about website building/UI design, platform integration, and QA testing and data reconciliations, one should be able to analogize these use cases to the systems and platforms.

WEBSITE BUILDING/UI DESIGN

How many times have you or your business ever wanted to make a website but were not knowledgeable in front-end development languages such as HTML, CSS, or JavaScript? Learning these languages is no longer a technical barrier in building a website. Figure 1 on the following page is an example of what a drag-and-drop platform looks like for developing custom business apps.

Figure 1: Custom business apps UI/UX outline - Drona HQ

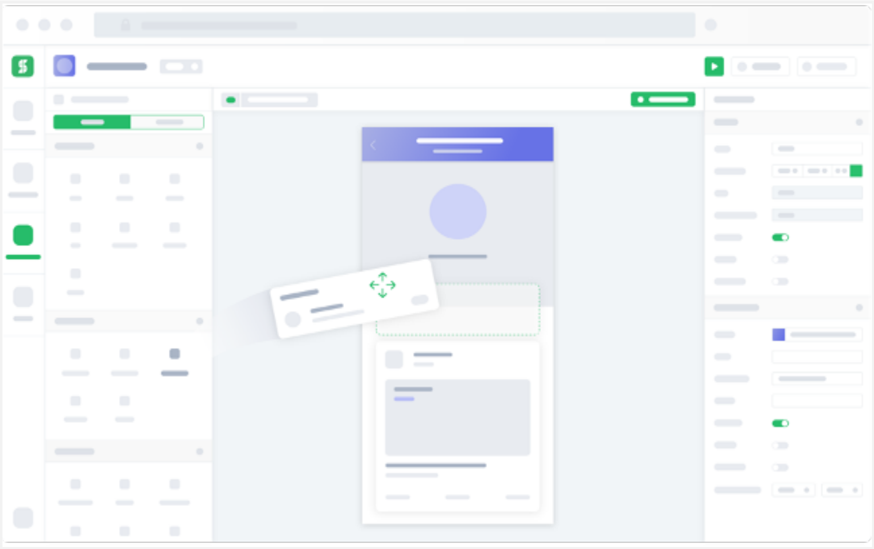


Image obtained from Drona HQ

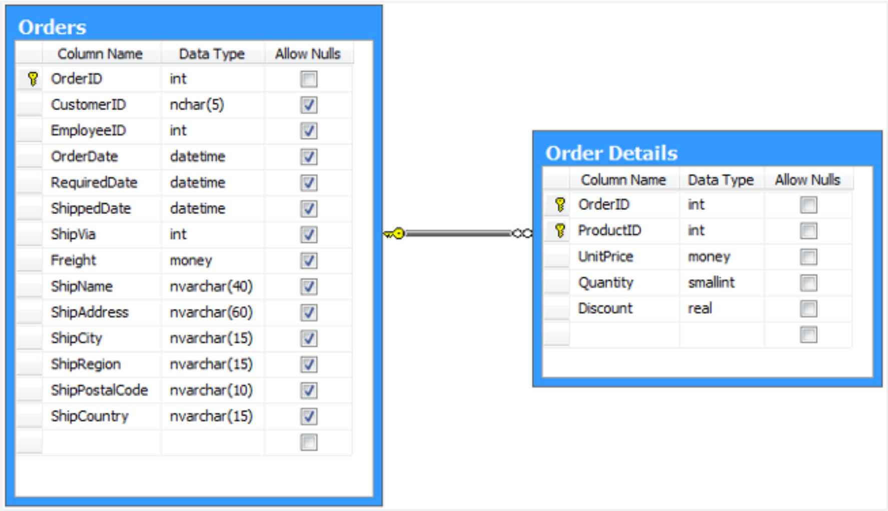
There are many low- and no-code platforms that specialize in user experience (UX) and user interface (UI) design. Additionally, these applications have the ability to hook into the back-end of systems and display any custom data that you would ever want to show. According to a TechTarget Network article on low-code use cases and industry examples, user interfaces usually need to work across various devices and platforms, from desktop applications to mobile and web apps. Having the ability to customize the UX across multiple devices from one platform is a huge time saver and reduces the need to hire pure technical development talent.

PLATFORM INTEGRATION

Do you or your business ever find yourself interested in analyzing/manipulating data from multiple systems but can't seem to merge the datasets without manually downloading data into an Excel file? Low-code and no-code platforms allow you to hook into the back end of certain systems and software to directly retrieve, manipulate, and export data to whatever format you need. This can be crucial for analytics and reporting purposes across multiple systems.

For example, as shown in Figure 2, let's say you house order data in one system and order detail data in another system. Using a unique identifier, like `OrderID`, you can combine the datasets in a low-code tool and figure out what products are typically bought in what cities. This can be done by linking the two tables and aggregating the count of the `ProductIDs` when grouping by `ShipRegion`.

Figure 2: Linking data between two systems based on a unique identifier



Orders

Column Name	Data Type	Allow Nulls
OrderID	int	☐
CustomerID	nchar(5)	☑
EmployeeID	int	☑
OrderDate	datetime	☑
RequiredDate	datetime	☑
ShippedDate	datetime	☑
ShipVia	int	☑
Freight	money	☑
ShipName	nvarchar(40)	☑
ShipAddress	nvarchar(60)	☑
ShipCity	nvarchar(15)	☑
ShipRegion	nvarchar(15)	☑
ShipPostalCode	nvarchar(10)	☑
ShipCountry	nvarchar(15)	☑
		☐

Order Details

Column Name	Data Type	Allow Nulls
OrderID	int	☐
ProductID	int	☐
UnitPrice	money	☐
Quantity	smallint	☐
Discount	real	☐
		☐

Source: Code on Time

This is similar to a SQL data `pull` and `join`; however, someone who does not understand SQL still has the ability to perform this query. You can schedule a workflow that sends an email with this report after ingesting data from these two systems versus needing a technical professional to run a SQL query on a local machine.

QA TESTING AND DATA RECONCILIATION

In addition to building UX/UI and platform integrations, low- and no-code tools can be instrumental in supporting data governance and reducing any regulatory risk. For example, let's say you have two different systems collecting the same type of data. Customer personal data is in one system and customer order data is in another system, with a customer name being captured in both. If the integration between the platforms is not entirely sufficient, you could end up with different values for the same data. For example, someone named Thomas Jonah Smith could be named "Thomas Smith" in one system and "Thomas J. Smith" in another system.

When combining datasets and aggregating by name, Thomas Jonah Smith will be represented as two different people because of the slight difference in name. To mitigate this risk, especially if such data is shown to customers, having a low-code platform automate QA for certain checks could be extremely helpful. Similar to how we described platform integration, ingesting two datasets, merging them, and comparing the values allows all businesses — from small- to enterprise-level — to achieve better governance over their data.

Conclusion

Low-code and no-code automation platforms can have a near immediate positive impact on your organization. For companies of all sizes, having a platform that improves the efficiency of business processes can ultimately save an enormous amount of time and resources. Improving UX/UI, reconciling data, and integrating platforms will lead to higher productivity and better data governance, helping businesses transform at pace with their industry peers. Because they allow both programmers and non-programmers to develop applications, low-code and no-code platforms will enable individuals and businesses to create value for their consumers and customers.

REFERENCES

- https://appian.com/low-code-basics/benefits.html
- https://www.dronahq.com/low-code/
- https://searchsoftwarequality.techtarget.com/tip/Review-these-9-low-code-use-cases-and-industry-examples
- https://codeontime.com/print/learn/sample-applications/order-form/understanding-the-project

Zachary Blitz, Business Analyst at Warburg Pincus

@zachblitz8 on DZone

Zachary Blitz is currently a Business Analyst in the Portfolio Analysis group at Warburg Pincus. Prior to Warburg Pincus, he worked as a Financial Services Consultant at Ernst & Young. He graduated from the University of Michigan in 2019 with a Bachelor of Science in Information and a minor in Economics. Outside of work, he enjoys the New York Knicks, running, traveling, gaming, and filming.

CONTRIBUTOR INSIGHTS

Best Practices for Adopting Low-Code and No-Code Platforms

How to Select the Right Platform for Your Team and Manage for Success

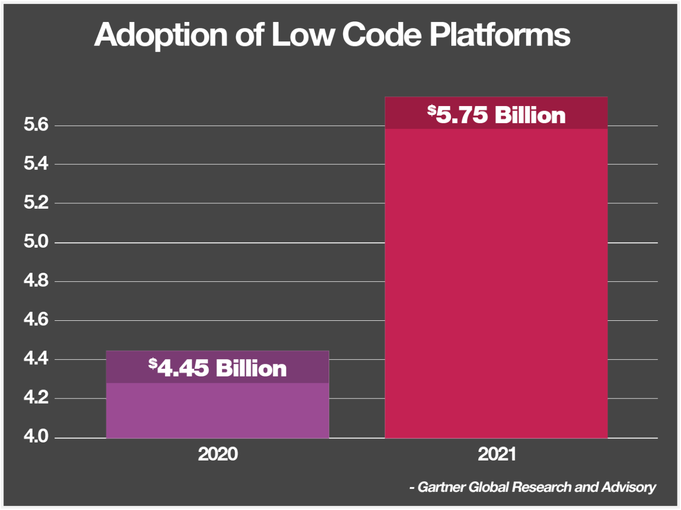
By Amy Groden-Morrison, Vice President at Alpha Software Corporation

When many developers think about no-code and low-code development tools, they often think of citizen developers and line-of-business experts creating simple insecure apps not suited for the enterprise. But that's no longer true. Low-code and no-code apps are now vital for businesses of any size looking to survive and thrive post-COVID.

With companies needing to support extended remote work and streamline business activities amid staff shortages, they're looking to mobile apps as one answer. Streamlining traditional iOS and Android app development is required to mobilize all these processes cost-effectively.

Gartner has become extremely bullish on low code, predicting the low-code application platforms market alone to grow from nearly $4.45 billion in 2020 to more than $5.75 billion in 2021. This growth will result from the economic fallout of the COVID-19 pandemic, as well as increasing demands for digital transformation.

Today's modern low-code and no-code software can build secure, cross-platform apps in days or even hours versus months, and some platforms even offer much of the flexibility and customization of native development. According to No-Code Census, these platforms make application development nearly five times faster, five times more affordable, and five times easier.



As a result, Gartner projects that by 2024, low-code application tools will be used for more than 65% of all application development, in both enterprises and smaller companies. Gartner adds that 66% of large companies will use a minimum of four low-code platforms. "The economic consequences of the COVID-19 pandemic have validated the low-code value proposition," explains Fabrizio Biscotti, research vice president at Gartner, in the report. "Low-code capabilities that support remote work function, such as digital forms and workflow automation, will be offered with more elastic pricing since they will be required to keep the lights running."

But if you're new to low-code and/or no-code, it can be tough to choose the right software platform. And once you've selected one, you may find it hard to deploy and manage properly in your organization. However, adopting a new app development platform needn't be that tough, as I'll explain in the rest of this article.

How to Choose the Right No-Code and Low-Code Platforms

The first step is to choose the right platform — one that fits your business' needs and existing skill sets. There are more than 300 no-code and low-code vendors, so you need to carefully vet them to make sure the one you select is right for you, says

Mayuresh Kulkarni, director of management consulting at KPMG US, one of the "Big Four" worldwide professional services networks. They suggest you choose one that meets the following requirements:

- Available features that fit your immediate business objectives
- Cross-platform apps that can be used on multiple browsers and mobile operation systems
- Integration with internal and external business applications and data sources
- Product training that scales with your use
- Healthy, active user community for learning and sharing
- Innovative product roadmap that will scale to support future growth and align with your industry's needs

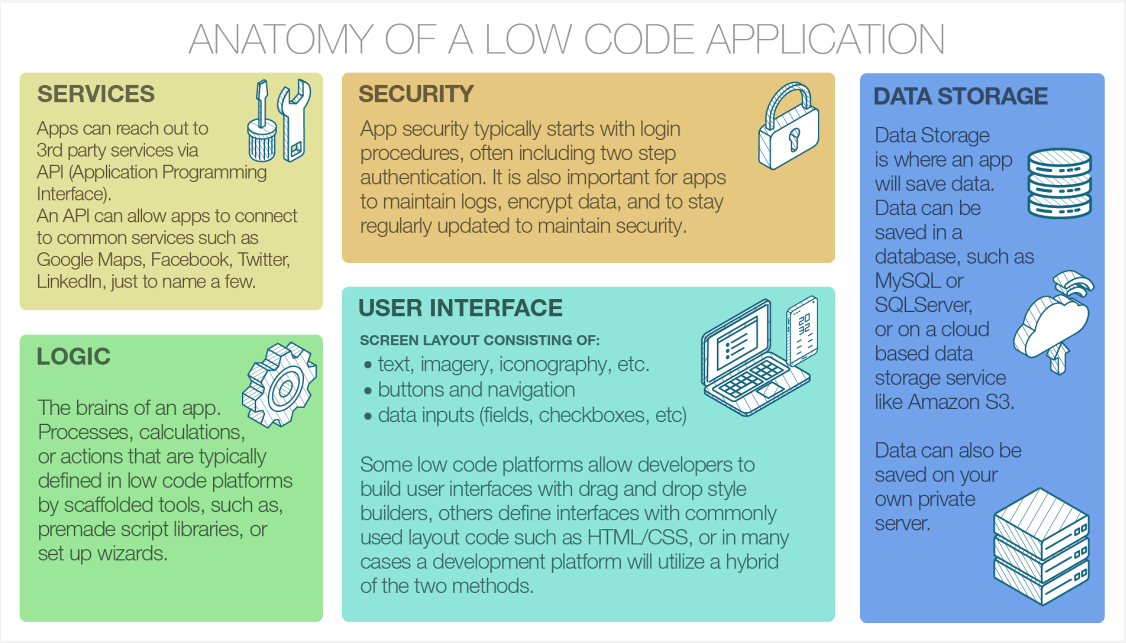
You should also consider who in your organization will be using the tools and what they'll be using them for. Will it be only for citizen developers to write simple, department-specific applications, or will IT use it for writing complex, enterprise-wide applications? Or will it be used by both? Selecting a platform geared for the right skill level of your users is critical for successful development and product adoption.

Another consideration is the remote working environment and where workers will be using the app. Will the app be used in areas without a strong Wi-Fi or cell signal? Will it be used in new construction sites or emergency situations where there won't be any internet at all? Do you want a smart app that saves on data costs? If these are important criteria for your company, a platform with robust offline operation will be a requirement.

Additional challenges surround data integration, data governance, and connections to systems of record. Will integration with multiple data sources and other applications be required, or are you planning only to build standalone applications? If the app will need to integrate with other data sources and applications, determine how easily the platform can accomplish this, and whether integrations are built right into the platform using APIs and other tools. Finally, make sure that the platform offers the security and compliance requirements for your business, its data governance policies, and the industry in which it operates.

Deploying and Managing Your No-Code and Low-Code Platforms

Once you've selected the best low-code platform for your organization, the real work begins — managing it so that it accomplishes exactly what you need. That involves choosing proper business tasks to mobilize and ensure that developers and end users are actually adopting the software platform and apps throughout your organization. KPMG's Kulkarni says you should first identify "hotspots" that are ideally suited for no and low code. They explain: "Some examples of hotspots are manual functions needing automation, paper-based processes that can be digitized, and convoluted workflows. It is imperative for the hotspots to be sufficiently concentrated, but you don't want these use cases to be too narrow."



ANATOMY OF A LOW CODE APPLICATION

SERVICES

Apps can reach out to 3rd party services via API (Application Programming Interface).
An API can allow apps to connect to common services such as Google Maps, Facebook, Twitter, LinkedIn, just to name a few.

SECURITY

App security typically starts with login procedures, often including two step authentication. It is also important for apps to maintain logs, encrypt data, and to stay regularly updated to maintain security.

DATA STORAGE

Data Storage is where an app will save data. Data can be saved in a database, such as MySQL or SQLServer, or on a cloud based data storage service like Amazon S3.

Data can also be saved on your own private server.

LOGIC

The brains of an app. Processes, calculations, or actions that are typically defined in low code platforms by scaffolded tools, such as, premade script libraries, or set up wizards.

USER INTERFACE

SCREEN LAYOUT CONSISTING OF:

- text, imagery, iconography, etc.
- buttons and navigation
- data inputs (fields, checkboxes, etc)

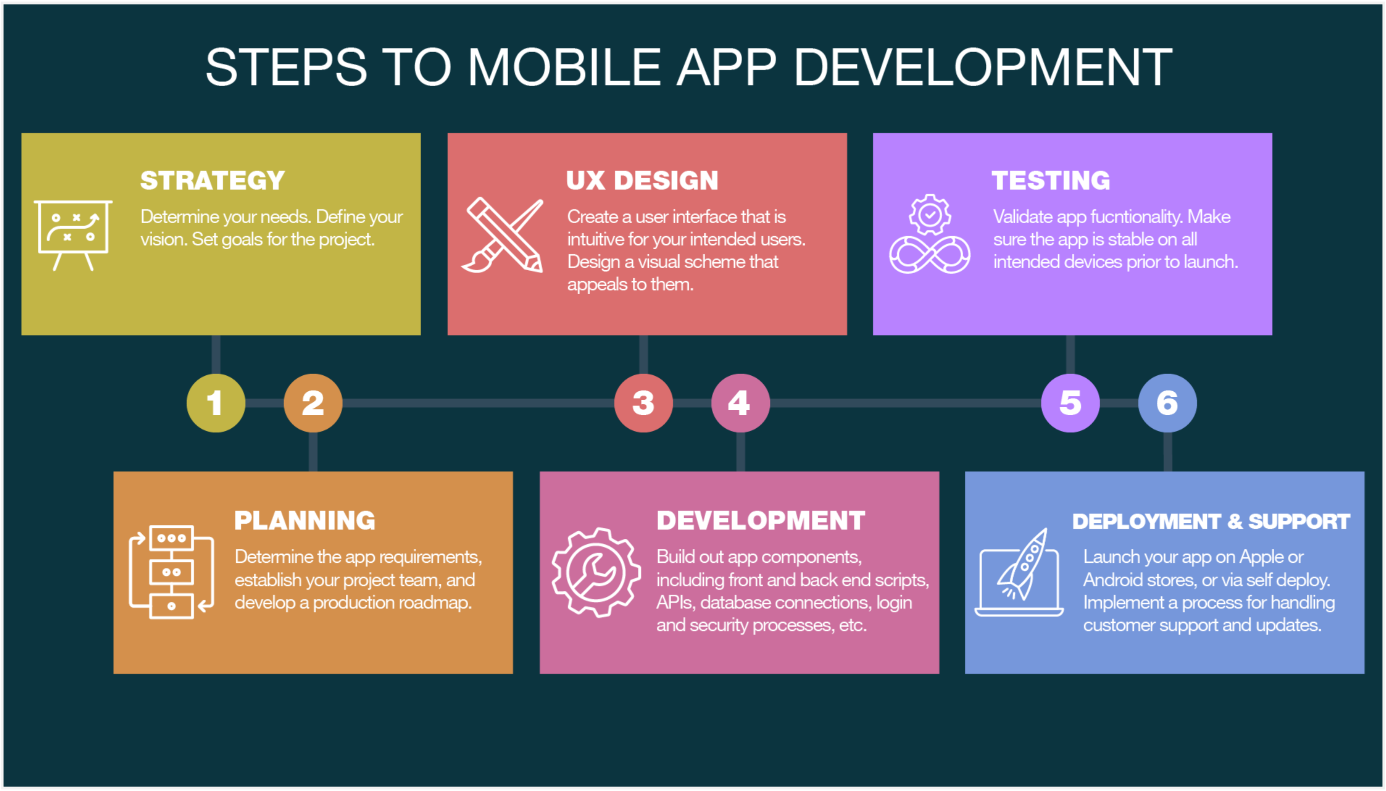
Some low code platforms allow developers to build user interfaces with drag and drop style builders, others define interfaces with commonly used layout code such as HTML/CSS, or in many cases a development platform will utilize a hybrid of the two methods.

Image: All low code apps are different but there are some basic similarities

Some examples include safety or vehicle inspections, field dispatch and customer service calls, patient data input, equipment tracking with RFID or bar code scanning, and more.

Low-code platforms, because they allow you to quickly build straightforward applications, are also ideal for the following tasks, according to TechTarget:

- **Web portals.** No-code and low-code development eliminates manually coding HTML and back-end components. You'll be able to build a variety of portals with consistent, common interfaces.
- **Line-of-business systems.** Line-of-business experts know best what they need — and the platform gives them the tools to build systems without relying on professional developers.
- **Digitized business processes.** The platforms are ideal for creating end-to-end digitized workflows, such as streamlining capital request applications.
- **Mobile apps.** Because the systems let you build apps from existing components, they're ideal for creating mobile apps. They can develop apps for Android and iOS from a common codebase, saving tremendous amounts of time. In fact, Forrester says, "current usage indicates that these platforms can propel software development to 10 times the speed of traditional processes."
- **Microservices applications.** Microservices can be strung together to build larger, complex applications. No code and low code are ideal for building those small components.
- **IoT-based apps.** Many IoT apps take data gathered by sensors and use that information to automatically take actions. An agricultural application that uses data from moisture and temperature sensors to automatically control irrigation and indoor lighting for greenhouse crops is one straightforward example where a low-code platform would excel.



STEPS TO MOBILE APP DEVELOPMENT

1 STRATEGY
Determine your needs. Define your vision. Set goals for the project.

2 PLANNING
Determine the app requirements, establish your project team, and develop a production roadmap.

3 UX DESIGN
Create a user interface that is intuitive for your intended users. Design a visual scheme that appeals to them.

4 DEVELOPMENT
Build out app components, including front and back end scripts, APIs, database connections, login and security processes, etc.

5 TESTING
Validate app fucntionality. Make sure the app is stable on all intended devices prior to launch.

6 DEPLOYMENT & SUPPORT
Launch your app on Apple or Android stores, or via self deploy. Implement a process for handling customer support and updates.

Image: Before you build your low code app, you will want to consider these important development steps

While no-code and low-code development offer considerable benefits, the software must be incorporated carefully into your organization, warns Chris Johannessen and Tom Davenport in the Harvard Business Review. The authors say the platforms can potentially make "shadow IT" problems worse — with citizen developers using unsanctioned tools that produce apps that aren't secure, don't work properly, or don't fit into an enterprise's overall computing infrastructure. To solve that problem, IT needs to oversee which platforms are sanctioned, who can use them, and how they can be used.

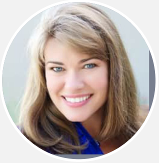
They recommend: "The best situation may often be a hybrid citizen/professional development model, in which the user develops 80% of the model, and hands it off to the developer for polishing. Or the user may develop the initial application using a graphic interface tool, and then give it to a developer to program it in Python or some other more scalable language… We've seen organizations where one system developer supports ten or more citizen developers."

Wrapping it All Up

What does all this mean for your organization? No matter its size, no-code and low-code software are likely in your company's future. In fact, Gartner predicts 75% of large enterprises using at least four low-code development tools by 2024. These platforms can increase productivity, build powerful applications quickly, and help speed digital transformation. Just make sure you use them for the right tasks and incorporate them into your organization's technology infrastructure with IT in the lead.

REFERENCES

1. Gartner, "Gartner Forecasts Worldwide Low-Code Development Technologies Market to Grow 23% in 2021," February 16, 2021
2. Forbes, "The Rise of Low-Code App Development," Ilker Koksal, April 29, 2020
3. VentureBeat, "7 tips for managing low-code/no-code adoption in your enterprise," Mayuresh Kulkarni, July 3, 2021
4. TechTarget Search Software Quality, "What is low-code? A guide to enterprise low-code app development," Stephen J. Bigelow, March 15, 2021
5. Forrester Blog, "Why You Need to Know About Low-Code, Even If You're Not Responsible for Software Delivery," John Rymer, Vice President, Principal Analyst, August 8, 2018
6. Harvard Business Review, "When Low-Code Development Works – and When It Doesn't," Chris Johannessen and Tom Davenport, June 22, 2021



Amy Groden-Morrison, Vice President at Alpha Software Corporation

@AmyatAlphaSW on DZone

Amy Groden-Morrison has held marketing leadership roles at TIBCO Software, Spotfire, RSA Security and Ziff-Davis. Her past accomplishments include launching a company on the NYSE, rebranding a NASDAQ-listed company amid a crisis and building programs to achieve 30%+ growth at tech startups. She currently advises early-stage tech startups, is an avid sailor, and holds an MBA from Northeastern University.

CONTRIBUTOR INSIGHTS

No-Code and Low-Code Development: The Ultimate Dev Time-Saver

By Applying Low Code, Save 90% or More of Your Time

By Thomas Hansen, CEO at ServerGardens.com

If you ask 100 developers what no code is, you're likely to get 100 different answers. However, what is often agreed upon is that no-code development offers the ability to create software without needing dozens of software developers working — often for months at a time — on software before delivering a working solution. Essentially, this means that no code is **the process of automating software systems**.

How Does No Code Work?

Yet again, there are as many answers to this question as there are people with opinions about the concept — I can only answer this question by explaining how I implemented a low-code and no-code software system myself.

Internalized Standardization Processes

Standards are imperative if you want to implement your own no-code solution. If you can implement internal standards regarding how your code is built, the computer can then make assumptions about how the code works. The reason for this is because of a little-known fact: **Your computer doesn't understand code!**

While I understand this may sound a bit absurd, it is fundamentally true. If it wasn't true, we wouldn't need interpreters or compilers, and the computer could simply execute your code directly without needing to change it before execution. However, if your computer can assume that all of your `HTTP PUT` endpoints are updating a database entity, it no longer needs to understand the code, and it can generate an entity UI interface, wrapping the input arguments your endpoint requires.

If the `PUT` assumption above is true, however, you could use tools such as Swagger to extract metadata about your endpoints and create a GUI, wrapping all your `PUT` endpoints. The same is true for all CRUD operations. Typically, we use `PUT` for update, `POST` for create, `DELETE` for delete, and `GET` for read.

The beauty of this approach is that as a general architectural design pattern, regardless of whether or not you're interested in no code, implementing such internal standards makes your code more easily understood. Once a human developer understands one endpoint, they have effectively understood all endpoints that the API exposes.

Based on that conclusion, no code is simply the natural side effect of applying a good design to your software, implementing conventions and rules that everyone can follow to understand the system. In fact, whether or not your existing codebase can be "no-coded" arguably becomes the definition of how much quality it possesses. If your code doesn't align with these rules and conventions, it's likely difficult to understand. If it's difficult to understand, then it's oftentimes badly architected. But if it is built upon a good architecture and able to wrap these standardization constructs, it grows organically without any ceilings, similarly to how biological organisms grow, exclusively based upon standardization mechanisms.

This is the reason I chose "Server Gardens" as the name of my company I must confess — simply because good standards and conventions, allowing for correct assumptions, results in less cognitive load to the point where the computer can automatically "grow" your code. When you are armed with internal standards, the process is more similar to how biological organisms grow than how software is typically engineered.

Metadata

Metadata is possibly the most under-appreciated paradigm we've ever had in regard to the art of creating software. Metadata, simply put, is "data about data." For instance, when you send a text, the message is the data. Metadata are things such as "how many characters did your message contain," "what time was the message sent," "which phone did you use," etc. Hence, metadata is an axiom describing properties about the data you're interested in.

When it comes to your codebase, metadata consists of components such as "which database connection is this code relying upon" and "what input does this endpoint require." For existing software systems, Swagger is a great tool for extracting metadata. Once you have metadata, you can automatically scaffold `POCO` classes and methods around your code, consuming your code where no code grows up from the ground almost organically.

Now, metadata capabilities from tools like Swagger become pale in comparison to what a specific metadata-designed programming language gives you. In the previous section, we said, "*Your computer does not understand code.*" If you can solve this problem, **your code becomes metadata**.

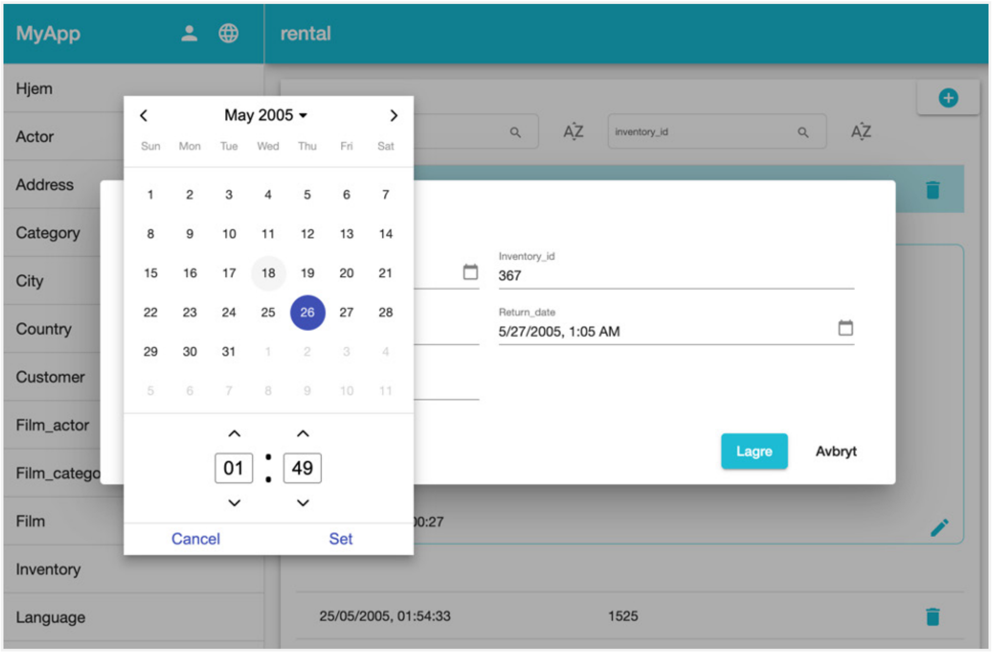
Once the entirety of your codebase exclusively consists of metadata, your computer can create 100% accurate assumptions about every single line of code in your codebase. If you can achieve this, the computer can easily answer questions like, "Does this method log?" and "Which database tables is this code using?" These questions are impossible to answer using metadata extractors such as Swagger, but they are implicitly given if you use a metadata programming language.

For an example, check out my own creation: Hyperlambda. Hyperlambda, of course, is at the core of Magic and the reason why Magic can do what it does. Fundamentally, it's just a tree structure, which is Turing complete, and your computer is capable of executing without interpreting or compiling. This way of thinking turns the entirety of your codebase into "metadata."

Once the computer has access to every pixel of metadata, your computer now "understands code" and can draw 100% accurate assumptions about the code; thus, it can now create automated processes wrapping, your codebase and solving every problem imaginable.

Combining Metadata and Standards

To better understand the true power of meta data, the following screenshot originates from a system created that starts by reading meta data from the database. Database meta data consists of information like, "What are the names of your tables?" and "What are the primary keys of your tables?" Next, it generates a web API, wrapping your tables in Hyperlambda. Finally, it assembles a complete Angular front end, wrapping the web API. Basically, the software system in the following screenshot was, in its entirety, created without a human being adding as much as a single line of code to it.



Inventory_id
367

Return_date
5/27/2005, 1:05 AM

0:27

25/05/2005, 01:54:33 1525

The software system in the screenshot was created in **one second**. The system consists of roughly 3,200 lines of back-end code and some additional 30,000 lines of front-end Angular code. The average human software developer can produce roughly 550 lines of code manually per month. Ignoring all other aspects, the above software system would have required one human software developer roughly 60 months to complete. **The computer created it in one second**.

Wrapping Up

So is no code just a bit of excitement, or is it our undeniable future? To answer that question: Let's address what the above software system does *not* do. Primarily, it does not end up with a result resembling exactly what you want — and unless its back-end code is generated around a 100% perfectly applied "standard," it kind of freezes, stops, and is unable to proceed. It requires assumptions to be correct if it is to succeed. And more often than not, it also requires a bit of manual coding after you are done.

As the inventor of this system, I have obviously used it a lot. But after the initial "wow," you will probably realize as I did, that it's not really a threat to your job because once the computer has done its job, it will require (some) manual coding afterward. And even if it did everything, there is still deployment, orchestrating multiple systems together, and enterprise architecture left for the human to do — in addition to database designs required for it to function.

In summary, allow me to answer one final question:

> Q: Will no code steal your job?
>
> A: Only 80% of the most boring parts!

Regardless of how intuitive our no-code and low-code tools become, the end result will always need a human touch. No and low code is about empowering the end user. Once you realize that simple fact, you also realize you have nothing to fear from it.

Thomas Hansen, CEO at ServerGardens.com

@mrgaia on DZone | Creator of polterguy.github.io

Autodidact software developer with 40 years of experience, since the age of 8, and 20+ years as a professional software developer, creating enterprise software for dozens of companies around the world. Now the CEO of ServerGardens.com, a company exclusively creating no-code and low-code tools.

CONTRIBUTOR INSIGHTS

A Low-Code Approach to Building Integrations

Low Code Brought the Age of Hyperautomation. What Will Low-Code Integration Bring?

By Roland Alston, Thought Leadership Blogger & Brand Journalist at Appian Corporation

The COVID-19 crisis accelerated the already blistering pace of digital transformation. Which is another way of saying organizations that can quickly integrate applications, systems, and data to adapt to constant change will outperform companies that can't.

But if your IT organization is like most, you probably have a backlog of projects and initiatives that never ends. The business has needs. And your team? Well, it probably has trouble keeping up, what with everything else they must do to keep operations humming. In fact, research shows that now:

- 50% of all new app development requests end in failure.
- 15% never get started.
- 20% get delivered but don't meet expectations.
- 15% get started but never get delivered.

Low-code integration is the opposite of that. For starters, developers can accelerate the application integration process with easy-to-use drag-and-drop visual design tools that speed time to market and business impact without compromising quality, security, or user experience. Yes, you could choose to buy off-the-shelf solutions to meet short-term business needs. But integration is always complex and quickly becomes an issue. Silos of data and process start to build up. And before you know it, you've created a bigger problem than you had in the first place.

The perfect solution for this painful conundrum is an enterprise-class, low-code automation platform. Unlike throwback rapid app development tools, modern low-code platforms simplify the entire software development lifecycle with pre-built, no-code integration and templates that reduce the time it takes to connect with databases, cloud, robotic process automation (RPA), artificial intelligence (AI), and application programming interfaces (APIs) from hours or days to just minutes.

From Idea to Business Impact Faster

Traditional software development tools essentially allow developers to build industrial-strength integrations by writing numerous lines of complex code. But the thing about low-code integration is that it allows developers and non-developers alike to build powerful applications by using a mouse to draw lines and boxes and create different kinds of decision nodes unlimited by the user's coding skills.

This kind of remarkable agility allows users to create powerful applications and integrate them with data sources, web services, and legacy systems with remarkable speed. Case in point: Ryder Systems, Inc., a commercial truck fleet management and supply chain solutions company that consistently ranks among FORTUNE's World's Most Admired Companies. The company's 18-person development team was quickly able to develop and deploy over 40 applications that improved customer experience and boosted employee productivity.

"Even within a couple of weeks, we are able to show [the business user] a prototype of what this product would look like," said Ramesh Sundaresan, VP and Division CIO. "It's much easier for us to get their feedback at a much earlier stage and make the necessary changes before we get to the end." Ryder has been able to cut rental transaction times in half, reducing a process that used to take 20 minutes down to about 10 minutes. Their customers are thrilled, with a tenfold increase in customer satisfaction index responses and a 10% increase in satisfaction scores.

3 Ways Low-Code Integration Speeds Business Impact

1. **Simplified Design:** Instead of coding integrations, you configure an integration and test to see if it works. Everything you build with low code is reusable. So if you build an integration or define a rule for an application today, adding that same component into a different application tomorrow is a snap.
2. **Simplified Collaboration:** For developers and non-developers alike, low code is a productivity multiplier that makes it easier for line-of-business execs and IT to collaborate to optimize the business impact of application integrations, making it easier to keep business and IT on the same page.
3. **Simplified Deployment:** Modern low-code platforms include DevSecOps capabilities that make deploying, integrating, and managing applications simple. Want to deploy that new application to users on web, mobile, and tablets with a single click? Need to change, update, or otherwise customize a business-critical application on a dime with no downtime and no interruption to your business operations? It's all possible with low-code integrations.

A FASTER, EASIER WAY TO INTEGRATE THIRD-PARTY DATA FOR DIGITAL SERVICES

Experts say the hardest thing about integrating applications is data because developers must know where it is, access it, get permissions, and more. But the best low-code platforms turn data into drag-and-drop objects, automatically identify data elements, and make them easy to select and integrate into a new workflow — even when you're selecting and combining elements from multiple data sources.

A leading vendor in the low-code space calls this data-driven functionality "low-code data," a feature that allows platform users to do complex data-workflow integrations by dragging and dropping, configuring, and seamlessly connecting to data in an intuitive and simple way. Imagine, for example, reaping the benefits of a platform that allows users to pull data from a wide range of systems, and add AI to any workflow to quickly extract and classify data from massive quantities of incoming documents. This is precisely the kind of productivity boost offered with low code.

GETTING THE MOST OUT OF LOW-CODE INTEGRATION

Enter software development kits (SDKs). These allow low-code developers to extend a platform's integration capabilities. Low-code vendor Appian, for example, offers two SDKs — Connected System Plug-ins and Component Plugins — that make it easy for users to create branded, low-code integrations and ship them with their logo to the Appian AppMarket.

Be sure to look for a low-code platform that comes with a broad selection of connected systems, which allows users to integrate with legacy systems and services with the speed and simplicity of point-and-click functionality. Also look for a broad selection of plug-ins that allow graphical integrations with enterprise systems and third-party services.

TURNING THE IT PRESSURE COOKER INTO A STRESS-FREE ZONE

This is a battle that many organizations have no hope of winning. Integrating custom software with existing business systems can burn through massive amounts of time and money. But it would sure be cool if developers could easily integrate applications and customize code as needed. So what if there was a way to quickly connect custom applications to legacy systems and also easily integrate with essential office productivity suites, enterprise applications, and DevSecOps tools?

Oh, and what if there was also a way to integrate more applications in less time, and provision and stage cloud-native apps and deploy them when and where you needed them — to the cloud or on-premises? That's the magic of low-code integration. Case in point: Iccrea Cooperative Banking Group, the largest collective bank in Italy, released 13 applications in a single year using low code with just nine developers and cut the time it takes to introduce new products in half.

"Building applications is easier," said Giovanni Gallucci, the head of IT process automation. "Our product experts do not require super technical, specialist training to build custom applications." Their new applications have led to dramatic business benefits, such as a 75% decrease in fraud management processing time, where they have been able to cut a five-month backlog for managing fraud cases down to zero. In their anti-money laundering area, they improved Service Level Agreements (SLAs), and are now able to respond to affiliated banks within 10 days instead of the SLA of 45 days.

Low-Code Integrations: Easy, Secure, and Productive

The top low-code platforms come with integration designer functionality that allows users to configure and test integrations and create an environment that speeds the process of integrating with external systems and third-party data in minutes.

In the wake of the pandemic, many organizations are increasingly turning to non-developers to build, scale, and integrate critical applications. But research shows this “citizen developer” trend could also pose serious data, security, and integration challenges for large organizations, according to a recent Appian IT survey. Here’s the math:

- 73% worry about non-developers using the wrong data in critical applications.
- 69% said they are concerned about compromised security.
- 58% reported integration issues in applications built by non-developers

But there’s another takeaway from the survey. Over 70% of IT shot callers trust low code to help them do a better job of securing non-developer applications and easing the pressure on overburdened IT departments. So if you’re looking to level-up your approach to integrations with a low-code platform, be sure to look for one with a broad selection of adapters to integrate applications with databases, services, and leading software products such as SAP, Salesforce, MS Dynamics, and more.

Finally, there are two existential challenges businesses are focusing on right now: figuring out how to automate critical workflows faster and adapting to massive change on a dime. Everything else pales in comparison. Perhaps it’s time for low-code integration to start getting the same attention.

Roland Alston, Thought Leadership Blogger & Brand Journalist at Appian Corporation
@rolandalston on DZone | @roland-alston-0894431 on LinkedIn

I’m a thought leadership blogger, ghostwriter, and content creator focused as all get-out on evangelizing human-centric innovation and getting across complex ideas in simple language. As a writer, I lean into storytelling focused on helping business and IT leaders solve problems, create customer value and drive business outcomes.

ADDITIONAL RESOURCES

Diving Deeper Into Low and No Code

BOOKS

Low-Code Application Development with Salesforce

By Enrico Murru

On the forefront of the low-code era, Salesforce offers dynamic functionality for developing, designing, and deploying high-quality apps. Learn how these offerings can be used to improve team productivity, streamline workflows, and ultimately, create solutions that offer rich user experiences — all without fully diving into programming complexities.

Mastering No-Code: No-Code AI Apps (Create Your Own Machine Learning Apps)
PAUL E. LOVE

Mastering No-Code: No-Code AI Apps

By Paul E. Love

Low-code tools enable organizations to create apps without relying entirely on individuals with programming backgrounds, but until recently, teams couldn't integrate AI with their low-code solutions. However, advancements in the space have sparked new machine learning capabilities across low- and no-code platforms. This book shows how no-code machine learning gives users of all technical proficiency levels the ability to develop their own AI-enabled apps.

TREND REPORTS

Modern Web Development

DZone's research focused on how developers shape successful web applications, and in particular, how to determine where storage and computation should occur. In this Trend Report, you'll examine critical web development design choices; explore the latest development tools and technologies; and learn about building a modern, performant, and scalable web application.

The Rise of Continuous Testing

Continuous testing (CT) isn't just about automation or CI/CD pipelines. CT involves testing throughout the SDLC — automating the appropriate tests, implementing the proper policies, and ensuring teams have effective test automation frameworks in place. Read this Trend Report to learn more through original research and expert contributor articles.

REFCARDS

End-to-End Test Automation Essentials: Evolving End-to-End Testing in the Age of No Code

End-to-end (E2E) tests are often neglected, due in large part to the effort required to implement them. Automation solves many of these problems by ensuring a consistent, production-like test coverage of the system. In this Refcard, learn the fundamentals of E2E test automation through test coverage, integration, and no-code options.

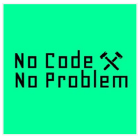
Microservices and Workflow Engines: Getting Started With Agile Business Process Automation

Automating business processes can help meet success metrics like increased team agility, faster time-to-market, and improved customer service. But existing dependencies between your systems and teams can make process automation more challenging to achieve and maintain. This Refcard covers ways to address such challenges with a microservice architectural style and workflow engine for orchestration, plus key techniques for microservice design, communication, and state management.

PODCASTS

High Tech-Low Code

Listen to this podcast aimed at helping educate the software industry at large about low-code technologies — how these solutions are disrupting and transforming markets ranging from finance and healthcare to utilities and retail. Hosts speak with leaders from companies like Microsoft, Digital Ventures, and Tech Women Today.



No Code No Problem

This podcast features conversations with creators building companies in the no-code space and covers various topics ranging from APIs and machine learning to mobile and web app design. Explore how no-code tools are changing the software and technology industries in 50+ episodes available across nine listening platforms.