2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR) | 978-1-7281-8710-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/MSR52588.2021.00018

An Empirical Study of Developer Discussions on Low-Code Software Development Challenges

Md Abdullah Al Alamin*, Sanjay Malakar†, Gias Uddin*, Sadia Afroz†, Tameem Bin Haider†, Anindya Iqbal†
*University of Calgary, †Bangladesh University of Engineering and Technology

***Abstract*—Low-code software development (LCSD) is an emerging paradigm that combines minimal source code with interactive graphical interfaces to promote rapid application development. LCSD aims to democratize application development to software practitioners with diverse backgrounds. Given that LCSD is relatively a new paradigm, it is vital to learn about the challenges developers face during their adoption of LCSD platforms. The online developer forum, Stack Overflow (SO), is popular among software developers to ask for solutions to their technical problems. We observe a growing body of posts in SO with discussions of LCSD platforms. In this paper, we present an empirical study of around 5K SO posts (questions + accepted answers) that contain discussions of nine popular LCSD platforms. We apply topic modeling on the posts to determine the types of topics discussed. We find 13 topics related to LCSD in SO. The 13 topics are grouped into four categories: Customization, Platform Adoption, Database Management, and Third-Party Integration. More than 40% of the questions are about customization, i.e., developers frequently face challenges with customizing user interfaces or services offered by LCSD platforms. The topic "Dynamic Event Handling" under the "Customization" category is the most popular (in terms of average view counts per question of the topic) as well as the most difficult. It means that developers frequently search for customization solutions such as how to attach dynamic events to a form in low-code UI, yet most (75.9%) of their questions remain without an accepted answer. We manually label 900 questions from the posts to determine the prevalence of the topics' challenges across LCSD phases. We find that most of the questions are related to the development phase, and low-code developers also face challenges with automated testing. Our study findings offer implications for low-code practitioners, platform providers, educators, and researchers.**

***Index Terms*—Low-Code, Issue, Challenge, Empirical Study.**

I. Introduction

LCSD is a new paradigm that enables the development of software applications with minimal hand-coding using visual programming with graphical interface and model-driven design. LCSD embodies End User Software Programming [36] by democratizing application development to software practitioners from diverse backgrounds [17]. By facilitating automatic code generation, the low-code development tools allow developing production-ready applications with minimal coding. It addresses the gap between domain requirement and developers' understanding that is a common cause of delayed development in many applications with complex business logic. The benefits of using LCSD platforms also include flexibility and agility, fast development time allowing quick response to market demands, reduced bug-fixing, lower deployment effort, and easier maintenance. Hence, the industry of low-code development is gaining popularity at a rapid pace. According to Forrester report [47], the LCSD platform market is expected to be $21 Billon by 2022. According to Gartner report, by 2024, around 65% of large enterprises will use LCSD platforms to some extent [61].

To date, there are more than 200 LCSD platforms, offered by almost all major companies like Google [21] and Salesforce [49]. Naturally, LCSD has some unique challenges [48]. Wrong choice of LCSD application/platforms may cause a waste of time and resources. There is also concern about the security/scalability of LCSD applications [26]. With interests in LCSD growing, we observe discussions about LCSD platforms are becoming prevalent in online developer forums like Stack Overflow (SO). SO is a large online technical Q&A site with around 120 million posts and 12 million registered users [35]. Several research has been conducted to analyze SO posts (e.g., big data [7], concurrency [2], blockchain [59], microservices [9]). However, we are aware of no research that analyzed LCSD discussions on SO, although such insight can complement existing LCSD literature – which so far has mainly used surveys or controlled studies to understand the needs of low-code practitioners [20, 27, 3, 26].

In this paper, we report an empirical study to understand the types of challenges and topics in LCSD developer discussions in SO by analyzing all 4.6K SO posts related to the top nine LCSD platforms at the time of our analysis (according to Gartner). We answer three research questions:

RQ1. What types of topics are discussed about LCSD in SO? Given LCSD is a new paradigm, it is necessary to learn about the types of topics LCSD practitioners discuss in a technical Q&A site like SO. Therefore, we apply topic modeling algorithm LDA [13] on our dataset of 4.6K posts. We find a total of 13 LCSD topics which are grouped into four categories: Customization of LCSD UI and Middleware, LCSD Platform Adoption, LCSD Database Usage, and Third-Party Integration. A majority of the (40%) questions are asked about the diverse challenges developers face while attempting to customize the user interface (UI) or a service/form provided by an LCSD platform. This is due to the fact that LCSD platform features are inherently heavy towards a graphical user interface (GUI) in a drag and drop environment. As such, any customization of such features that are not directly supported by the LCSD platforms becomes challenging.

RQ2. How are the topics distributed across the LCSD life cycle phases? Our findings from RQ1 show the unique nature

978-1-7281-8710-5/21/$31.00 ©2021 IEEE
DOI 10.1109/MSR52588.2021.00018

of challenges LCSD developers face, like customization issues. Given the considerable attention towards LCSD support by software vendors/platforms, the success of the platforms/SDKs can benefit from their effective adoption into the various stages of a software development life cycle (SDLC). For example, if testing of LCSD application cannot be done properly, it is difficult to develop a reliable large-scale LCSD application. We, therefore, need to understand whether and how LCSD developers are discussing the adoption of tools and techniques in LCSD topic across different SDLC phases. We randomly sampled 900 questions from our dataset and manually analyzed the types of LCSD challenges developers discussed in the questions. For each question, we label the SDLC phase for which the developer noted the challenge. We found that more than 85% of the questions revolved around development issues, and it is more or less consistent across all the four topic categories. We also find that testing can be challenging for LCSD applications due to the graphical nature of the SDKs, which can be hard to debug.

RQ3. What LCSD topics are the most difficult to answer? Our findings from the above two research questions show that LCSD developers face challenges more unique to LCSD platforms (e.g., Customization topics) as well as similar to other domains (e.g., Database topics). Therefore, it can be useful to learn what topics are more difficult to get the right answer to and whether the popularity of the topics can suffer due to the observed difficulty. We compute a suite of popularity and difficulty metrics for each topic, like the view count and the percentage of questions without an accepted answer. We find that questions related to the topic "Dynamic Event Handling" from the Customization topic category are the most difficult (to get an accepted answer) but also the most popular.

To the best of our knowledge, ours is the first empirical study of LCSD and platforms on developer discussions. The findings would help the research community with a better focus on the specific LCSD areas. The practitioners can be prepared for difficult areas. Relevant organizations will be able to design more effective and usable tools for LCSD, increasing their usability. All stakeholders can work together for improved documentation support. The LCSD vendors can support increased customization of the LCSD middleware and UI to make the provided features more usable.

Replication Package: The code and data are shared in https://github.com/disa-lab/LowCodeEmpiricalMSR2021

II. Background

What is an Low-code Application? To cater to the demand of the competitive market, business organizations often need to quickly develop and deliver customer-facing applications. LCSD platform allows the quick translation of the business requirement into a usable software application. It also enables citizen developers of varying levels of software development experience to develop applications using visual tools to design the user interface in a drag-and-drop manner and deploy them easily [32]. LCSD is inspired by the model-driven software

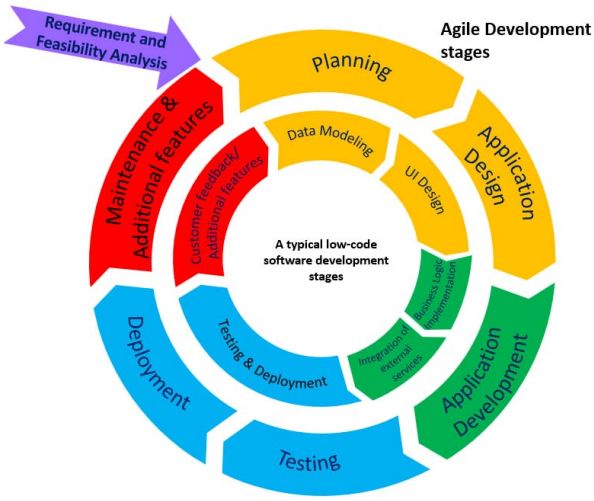

Fig. 1: Agile methodologies in traditional vs LCSD development

principle where abstract representations of the knowledge and activities drive the development, rather than focusing on algorithmic computation [48]. LCSD platforms aim to abstract away the complexity of testing, deployment, and maintenance that we observe in traditional software development. Some of the most popular low-code platforms are Appian [4], Google App Maker [21], Microsoft Powerapps [39], and Salesforce Lightning [49].

Development Phases of an LCSD Application. A typical LCSD application can be built in two ways [48]: 1) "UI to Data Design", where developers create UI and then connect the UI to necessary data sources, or 2) "Data to UI" where the design of the data model is followed by the design of the user interfaces. In both approaches, application logic is implemented, and then third party services and APIs are integrated. APIs are interfaces to reusable software libraries [44]. A major motivation behind LCSD is to build applications, get reviews from the users, and incorporate those changes quickly [60]. As such, the agile development methodology [12] and LCSD can go hand in hand because the fundamental principle and objective are customer satisfaction and continuous incremental delivery. The inner circle of Figure 1 shows the important development phases of an LCSD application, as outlined in [48]. The outer circle of Figure 1 shows the phases in a traditional agile software development environment. As LCSD platforms take care of many of the application development challenges, some of the agile application development phases have shorter time/execution spans in LCSD compared to traditional software development.

III. Study Data Collection and Topic Modeling

In this Section, we discuss our data collection process to find LCSD related posts (Section III-A). We then discuss the details of the topic modeling (Section III-B).

A. Data Collection

We collect LCSD related SO posts in three steps: (1) Download SO data dump, (2) Identify LCSD related tag list, and (3) Extract LCSD related posts from the data dump based on our selected tag list. We describe the steps below.

Step 1: Download SO data dump. We downloaded SO data dump [18] of June 2020. We used the contents of Post.xml file, which contained information about each post like the post's unique ID, type (Question or Answer), title, body, associated tags, creation date, view-count, etc. Our data dump included posts from July 2008 to May 2020 and contained around 58,544,636 posts. Out of them, 33.4% are questions, 66.6% are answers, and 17.4% questions had accepted answers.

Step 2: Identify low-code tags. We need to identify the tags that are related to LCSD in order to extract low-code related posts from SO discussions. To find relevant tags, we followed a similar procedure used in prior work [1, 2, 59, 30]. At Step 1, we identify the initial low-code related tags and call them T_{init}. At Step 2, we finalize our low-code tag list following related work [7, 62]. Our final tag list T_{final} contains 19 tags from the top nine LCDPs. We discuss each step in details below.

(1) Identifying Initial low-code tags. The SO posts do not have tags like "low-code" or "lowcode". Instead, we find that low-code developers use a LCSD platform name as a tag, e.g., "appmaker" for Google Appmaker [21]. Hence, to find relevant tags, first we compile a list of top LCSD platforms by analysing a list of platforms that are considered as the market leaders in Gartner [57], Forrester [47], related research work [48], and other online resources like PC magazine [37]. We find nine LCSD platforms are consistently mentioned in the above resources: Zoho Creator [63], Google App Maker [21], Salesforce Lightning [49], Quickbase [40], Outsystems [40], Mendix [34], Vinyl [58], Appian [4], and Microsoft Powerapps [39]. We thus focus on the discussions of the above nine LCSD platforms in SO. We find one tag per LCSD platform as the name of the platform (e.g., "salesforce-lightning"). However, upon close inspection of the tags in SO, we found that developers used more than one tag for some of the nine LCSD platforms. For example, "Microsoft Powerapps" has multiple tags (e.g., "powerapps", "powerapps-formula", "powerapps-canvas"). At the end of both quantitative analysis and manual validation by four authors, we found a total of 16 tags for the nine LCSD platforms. We refer to these 16 tags as T_{init}.

(2) Finalizing low-code related tags. Intuitively, there might be more variations to tags of nine LCSD platforms other than those in T_{init}. We use heuristics from previous related works [7, 62] to find other relevant tags. First, we denote our entire SO dump data as Q_{all}. Second, we extract all the questions Q that contain any tag from T_{init}. Third, we create a candidate tag list $T_{candidate}$ using all the tags found in questions Q. Fourth, we select significantly relevant tags from $T_{candidate}$ for our LCSD discussions. Following related works [7, 62], we compute significance and relevance for each tag t in $T_{candidate}$ with respect to Q (our extracted questions that has T_{init} tag) and Q_{all} (i.e., our data dump) as follows,

$$(Significance) \quad S_{tag} = \frac{\#\ of\ ques.\ with\ the\ tag\ t\ in\ Q}{\#\ of\ ques.\ with\ the\ tag\ t\ in\ Q_{all}}$$

$$(Relevance) \quad R_{tag} = \frac{\#\ of\ questions\ with\ tag\ t\ in\ Q}{\#\ of\ questions\ in\ Q}$$

A tag t is significantly relevant to LCSD if the S_{tag} and R_{tag} are higher than a threshold value. We experimented with a wide range of values of S_{tag} and R_{tag}. We found relevant tag set for LCSD for S_{tag} = 0.2 and R_{tag} = 0.005 . These values are consistent with related work [7, 2]. The final tag list T_{final} contains 19 significantly relevant tags.

Step 3: Extracting low-code related posts. An SO question can have at most five tags, and we consider a question as low-code related question if at least one of its tag is in our chosen tag list T_{final}. Based on our T_{final} tag set, we found a total of 7,302 posts from our data dump. There were 51.3% Questions (i.e., 3,747) and 48.7% Answers (i.e., 3,555) and among them 16.9% Questions (i.e., 1,236) had accepted answers. SO has a score-based system (upvote and downvote) to ensure the questions are in proper language with necessary information (code samples and error messages), not repeated or off-topic. Here is an example for a question with score "-4" where a practitioner is making an API related query in Powerapps($Q_{61147923}$)[1] platform. However, it is not clear what the practitioner is asking as the question is poorly written and without any clear example. In order to ensure good quality discussions, we excluded questions that had a negative score which resulted in 6,982 posts containing 51.5% Questions (i.e., 3,597) and 48.5% Answers (i.e., 3,385). Following previous research [7, 46, 10], we excluded unaccepted answers and only considered accepted answers for our dataset. Hence, our final dataset B contained 4,785 posts containing 3,597 non-negative scored questions and 1,188 accepted answers.

B. Topic Modeling

We produce LCSD topics from our extracted posts in three steps: (1) Preprocess the posts, (2) Find optimal number of topics, and (3) Generate topics. We discuss the steps below.

Step 1. Preprocess the posts. For each post text, we remove noise following related works [1, 7, 10]. First, we remove the code snippets from the body, which is inside <code></code> tag, HTML tags such as (, <a>, etc), and URLs. Then we remove the stop words such as "the", "is", "are", punctuation marks, numbers, non-alphabetical characters using the stop word list from MALLET [33], NLTK [31], and our custom low-code specific (i.e., LCSD platform names) stop word list. After this, we use porter stemmer [41] to get the stemmed representations of the words e.g., "wait", "waits", "waiting", and "waited" - all of which are stemmed to base form "wait".

[1] Q_i and A_i denote a question Q or answer A in SO with an ID i

Step 2. Finding the optimal number of topics. After the prepossessing, we use Latent Dirichlet Allocation [13] and the MALLET tool [33] to find out the LCSD related topics in SO discussions. We follow similar studies in Software engineering research using topic modeling [5, 6, 62, 7, 1]. Our goal is to find the optimal number of topics K for our dataset B so that the *coherence* score, i.e., encapsulation of underlying topics, is high. We use Gensim package [43] to determine the coherence score following previous works [56, 45]. We experiment with different values of K that range from {5, 6, 7, .., 29, 30} and for each value, we run MALLET LDA on our dataset for 1000 iterations [7]. Then we observe how the coherence score is changing with respect to K. We pick the topic model with the highest coherence score. Choosing the right value of K is important because, for smaller values of K, multiple real-world concepts merge, and for a large value of K, a topic breaks down. For example, in our result, the highest coherence score 0.50 for $K = 7$ and $K = 13$. We choose $K = 13$ as it captures our underlying topics better. For $K = 7$, we find that it merges the topics "Dynamic Event Handling", "Dynamic Content Display", and "Dynamic Form Controller". MALLET also uses two hyper-parameters, α and β, to distribute words and posts across the topics. Following the previous works [7, 2, 8, 46], we use the standard values $50/K$ and 0.01 for hyper-parameters α and β in our experiment.

Step 3. Generating topics. Topic modeling is a method of extracting a set of topics by analysing a collection of documents without any predefined taxonomy. Each document has a probability distribution of topics, and every topic has a probability distribution of a set of related words [10]. We produced 13 topics using the above LDA configuration on our dataset B. Each topic model offers a list of top N words and a list of M posts associated with the topic. In our settings, a topic consists of 30 most frequently co-related words, which represents a concept. Each post had a correlation score between 0 to 1, and following the previous work [59], we assign a document with a topic that it correlates most.

IV. Empirical Study

We answer three research questions by analyzing LCSD discussions in SO. The RQ1 aims to understand the types of topics discussed in SO about LCSD (Section IV-A). The RQ2 aims to understand how each topic is discussed across different stages of low-code SDLC (software development life cycle) (Section IV-B). The RQ3 offers insight into the challenges of LCSD developers across the observed LCSD topics based on the difficulty of getting answers (Section IV-C).

A. What types of topics are discussed about LCSD? (RQ1)

1) Approach: We get 13 low-code related topics from our LDA topic modeling, as discussed in Section III. We use card sorting [19] to label these topics following previous works [7, 2, 62, 46, 1]. In open card sorting, there is no predefined list of labels. To label a topic, we used the top 30 words for the topic and a random sample of at least 20 questions that are assigned to the topic. Four of the authors participated in the labelling process. Each author assigns a label for each topic and discusses with each other until there is an agreement. The authors reached an agreement after around 15 iterations of meetings over Skype and email and labeled the 13 topics from the LDA output discussed in Section II. After the labeling of the topics, we grouped them into higher categories. For example, UI Adaptation and Dynamic Form Controller are related to UI design, and so we group them into a group named UI. In the same way, Dynamic Event Handling, Dynamic Content Display, and Dynamic Content Binding topics are related to the middleware feature of low-code development platforms, and so we put these three topics into Middleware sub-category. We repeat this process until we can not find any more higher-level group. For example, the above mentioned two categories UI and Middleware belong to the application customization task where the developers customize the UI or the business logic of the application according to their need. Hence, we put them under a high-level category named Customization. Similarly, we put Access Control & Security into Configuration sub-category. Then we put Configuration sub-category and Client Server Comm & IO topic under Platform Adaptation high-level category.

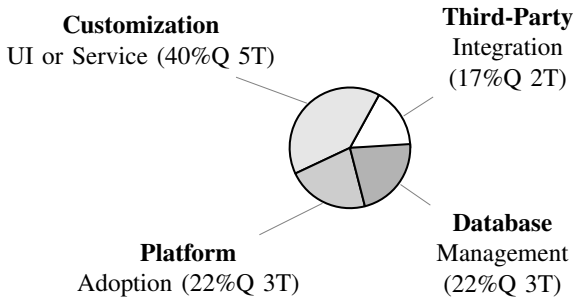

Fig. 2: Distribution of questions (Q) and topics (T) per topic category

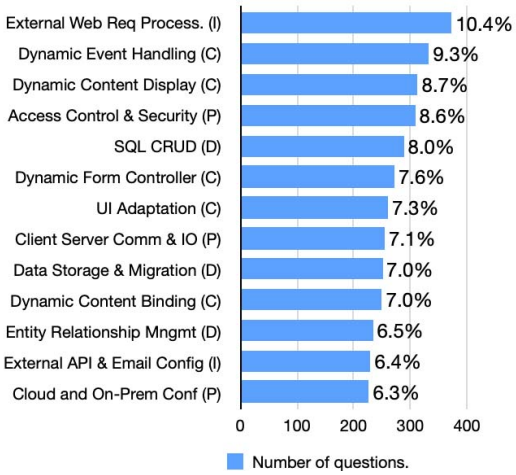

Fig. 3: Distribution of questions by low-code Topics (C = Customization Category, I = Integration, P = Platform Adoption, D = Database)

2) Results: We find 13 LCSD topics that we group into four high-level categories: **Customization, Platform Adoption,**

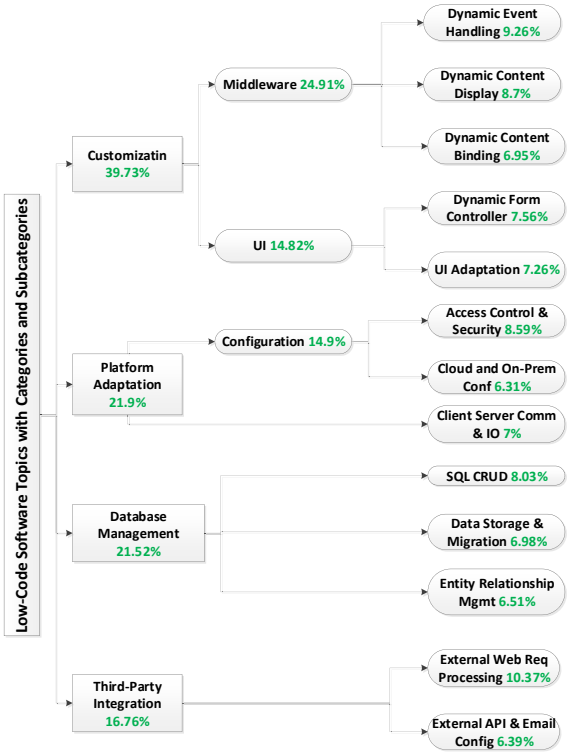

Fig. 4: Low-code topics categories and sub-categories

Database Management, and Third-Party Integration. Figure 2 shows the distribution of questions and topics among the four high-level categories. The *Customization* category covers the highest percentage of questions and number of topics (40% questions and five topics), followed by *Platform Adoption* (22% questions and three topics), *Database Management* (33% questions and three topics), and *Integration* (17% questions and two topics). Figure 3 sorts the 13 topics based on number of questions. The *"External Web Req Processing topic"* covers the most questions regarding queries and discussions related to the integration of third party services/APIs (10.4%). The *"Dynamic Event Handling"* has the second most questions (9.3%) related to the implementation of business logic.

In Figure 4, we group the 13 topics into four high-level categories. The categories are sorted according to the number of questions belonging to them. For example, the topmost category *Customization* has the most number of questions under them. Each category may consist of some sub-categories. For example, *Customization* category contains two sub-categories: (1) *Middleware*, and (2) *UI* . Each sub-category contains one or more topics. For example, Middleware sub-category consists of three topics: (1) Dynamic Event Handling, (2) Dynamic Content Display, and (3) Dynamic Content Binding. Each sub-category and topic are organized according to their distribution of questions. For example, under Customization (40%) category, *Middleware* (25%) is followed by *UI* (15%) sub-category. In the same way, "Dynamic Event Handling" (9.3%) topic is followed by "Dynamic Content Display" (8.7%) topic based on their question distribution.

• **Customization** is the largest category, with 40% of the SO questions and five topics. It contains discussions about business logic implementation, input and form validation, linking the UI to the backend storage via dynamic content binding, a drop-down menu with predefined value, formatting date and time, drop-down widgets, etc. It contains two sub-categories of topics: (1) *Middleware* sub-category covers discussions on the middlewares that provide support for system integration, the connection between UI and storage layer, etc., and (2) *UI* sub-category contains discussions on drag-and-drop UI and form design and also customization of UI components.

(i) **Middleware (25% questions)** sub-category contains three topics: (1) *Dynamic Event Handling (9.26%)* has discussions about handling user interaction events, accessing input value after form submission ($Q_{43096166}$), and rendering chart in the canvas($Q_{56154215}$). (2) *Dynamic Content Display (8.70%)* is about dynamically displaying items on the page ($Q_{53648077}$), displaying content based on previous action, and creating and accessing gallery from multiple data sources ($Q_{51764889}$). (3) *Dynamic Content Binding (6.95%)* is about updating views when some other values get changed ($Q_{59932262}$) and building process based on some values on the form ($Q_{61282976}$).

(ii) **UI (15% questions)** sub-category contains two topics: (1) *Dynamic Form Controller (7.56%)* contains discussions related to the design of forms with predefined values and the implementation of multi-select and customized drop-down values ($Q_{44013975}$), adding event-listeners to text widget ($Q_{46038130}$), etc. (2) *UI Adaptation (7.26%)* is about designing and customizing the user interface, resizing screen, etc.($Q_{34515865}$).

• **Platform Adoption** is the second-largest category, with 22% of the questions. It contains discussions about generic query on LCSD platform features and support, role management, SDLC management tools (e.g., scrum, agile), cloud setup and configuration, deployment issues, etc. The category has three topics. The topic, *Client Server Comm & IO (7.09%)*, contains discussions on client-server architecture ($Q_{54900592}$), debugging server-side scripts ($Q_{55283256}$), and general debugging queries on error messages or unexpected output ($Q_{50936643}$). The other two topics are grouped under Configuration sub-category.

(i) **Configuration (15% questions)** contains discussions on LCSD platform configuration on access control and cloud-based setup. There are two topics: (1) *Access Control & Security (8.59%)* is about discussion on role-based access control to tasks ($Q_{51431318}$), configuration of existing authentication mechanism (Azure Active Directory configuration) ($Q_{61734680}$). (2) *Cloud and On-Prem Conf (6.31%)* contains discussion on the proper configuration parameters and guidelines to connect to the cloud/on-prem databases ($Q_{55207558}$ and $Q_{45740520}$).

• **Database** is tied to the second biggest category, with 22% of the questions. It contains discussions about database connection, SQL CRUD operations, import/export existing data, etc. There are three topics: (1) *SQL CRUD (8.03%)* contains dis-

cussions on SQL query ($Q_{49051500}$, $Q_{59852901}$) and table joining ($Q_{37707699}$), (2) *Data Storage & Migration (6.98%)* discusses the upload and storing of files on the server ($Q_{49666940}$), moving files from one platform to another ($Q_{55726281}$), conversion of large CSV files to excel sheet ($Q_{50977178}$), etc., (3) *Entity Relationship Management (6.51%)* contains discussion on relational database design ($Q_{51881224}$) and platform support-/limitations on relational database ($Q_{58935331}$).

• **Integration** is the smallest category, with 17% of the questions and two topics. It contains discussions about email server configuration, integration of external services, OAuth, fetching and parsing data, etc. It contains two topics: (1) *External Web Req Processing (10.37%)* contains posts regarding API integration, parsing and debugging the responses of REST APIs ($Q_{21314917}$), and OAuth ($Q_{56873258}$), some general query on networking protocols such as HTTP, REST API ($Q_{48628269}$), etc., (2) *External API & Email Config (6.39%)* is about configuration, sending or forwarding emails ($Q_{34085695}$), configuration error ($Q_{31501424}$), how to use a generic programming language to send an email ($Q_{36341976}$), creating managing calendar events ($Q_{46738962}$), etc.

Summary of RQ1. We found 13 topics in our SO dataset relevant to low-code software development. The topics belong to four categories: Customization, Platform Adoption, Database, and Integration. Customization category has the most number of questions, followed by Platform Adoption, Database, and Integration. Out of the topics, "External Web Request Processing" topic under the Integration category constitutes the highest number of questions (10.4%), followed by the topic "Dynamic Event Handling" (9.3%) under the Customization category.

B. How are the topics distributed across the LCSD life cycle phases? (RQ2)

1) Approach: Agile software development methodology [12] has six SDLC phases: (1) Requirement Analysis & Planning, (2) Application Design, (3) Implementation, (4) Testing, (5) Deployment, and (6) Maintenance. Our final dataset *B* contained 3,597 questions. We needed at least 347 questions to produce a statistically significant sample, with a 95% confidence level and 5 confidence interval. As noted in Fig. 5, some phases have a low number of questions. Thus we wanted to find more examples of those phrases in our sample. As such, we analyzed 900 randomly sampled questions. The labelling of each question to determine the precise SDLC phases was conducted by several co-authors in joint discussion sessions spanning over 80 person-hours. Eventually, we manually labelled 916 questions, out of which 16 were considered invalid because they did not have any specific LCSD discussions. For example, a new practitioner is tasked with finding the right LCSD platform during the planning stage of his/her LCSD application. The practitioner queries, "Are there any serious pitfalls to Outsystems Agile Platform?" ($Q_{3016015}$). We thus assign the SDLC phase for it as "Requirement Analysis & Planning". Another question asks,

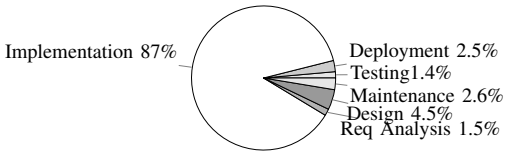

Fig. 5: Distribution of questions (Q) per SDLC phase

TABLE I: Distribution (frequency) of LCSD topics per SDLC phase. Each colored bar denotes a phase (Black = Requirement Analysis & Planning, Green = Application Design, Magenta = Implementation, Red = Testing, Blue = Deployment, Orange = Maintenance)

Topics	Development Phases Noted in #Questions
Customization (351, 39%)	
Dynamic Event Handling	73 7 1
Dynamic Content Display	2 78 1
Dynamic Form Controller	2 67
UI Adaptation	1 59 1 1 2
Dynamic Content Binding	5 49 2
Platform Adoption (189, 21%)	
Access Control & Security	4 3 42 2 14 12
Client Server Comm & IO	54 2 1
Cloud and On-Prem Conf	8 12 28 1 4 2
Database (214, 23.7%)	
SQL CRUD	1 4 74
Data Storage & Migration	3 63 2
Entity Relationship Mngmt	1 5 58 3
Integration (145, 16.1%)	
Ext Web Req Processing	3 99
Ext API & Email Config	41 1 1

"Google App Maker app not working after deploy" ($Q_{42506938}$). We label the SDLC phase as "Deployment".

2) Results: Figure 5 shows that the Implementation phase is found in 87% of the 900 questions we studied, followed by Design (4.5%), Maintenance (2.6%) and Deployment (2.4%) phases. This is not surprising, given that SO is a technical Q&A site and developers use the forum to find solutions to their technical problems. Table I shows the distribution of our 13 LCSD topics over six low-code SDLC phase based on our analysis of 900 SO questions. Besides the dominance of development phase related questions across all topics, we find the presence of other phases (e.g., testing, deployment) in customization and platform adoption topics.

Requirement Analysis & Planning (14, 1.5%). Requirement analysis is the process of developing software according to the expectation of the users. During planning, the feasibility, timeline, dependability, potential complexity/risks are analyzed and planned by paying attention to the operational aspects. The LCSD platforms usually provide requirement management tools that allow developers to collect data, customize checklists, import user stories into sprint plans. At this phase, the developers usually face enquiries about cost, learning curve, LCSD platform's support for faster application development, deployment, and maintenance features to choose the right platform for their business need. For example, in this popular question, a new practitioner is asking for some drawbacks on some potential pitfalls for a particular LCSD platform, e.g., "Are there any serious pitfalls to Outsystems Agile Platform?" ($Q_{3016015}$). A developer from that platform

provider suggests using the platform to build an application and decide for himself as it is hard to define what someone might consider a pitfall.

Application Design (40, 4.5%). In this phase, the design specification is made based on the requirements of the application. All the key stakeholders review and approve this considering application architecture, modularity, and extensibility. The LCSD developers face challenges regarding data storage design, drag and drop UI design, using on-prem data-sources with the LCSD platform (e.g., "Can AppMaker be used with SQL Server" ($Q_{55220499}$)), migrating existing data to LCSD platform ($Q_{46421271}$), designing a responsive web page (e.g., "Incorporating responsive design in App Maker" ($Q_{52744026}$)).

Implementation (785, 87.3%). At this phase, actual application development begins. LCSD developers face a wide range of challenges when trying to customize the UI, implement business logic, integrate third-party modules, debug and test the implemented functionalities, read the incomplete or incorrect documentation, etc. Developers ask application customization and UI customization questions like "How do I change timezone in AppMaker Environment?" ($Q_{47731051}$), drop-down menu customization (e.g., "powerapps: populate drop down list from another datasource" ($Q_{40159662}$)). There are many queries (17%) regarding external services or API integration. Many of these questions are relevant to a task-based tutorial on how to use a REST API and process its response (e.g., "How to upload files and attachments to the subject record using REST API?" ($Q_{61143493}$)). The root cause of many of these issues is incomplete or incorrect documentation. For example, in $Q_{46241015}$, a practitioner is querying about the integration of Zoho CRM and says that s/he does not understand the sample code. In $Q_{34510911}$, a practitioner asks for a sample code to convert a web page to a PDF. Clearly, the documentation is not sufficient for a smooth transition for entry-level practitioners.

Testing (14, 1.5%). The testing process of LCSD varies from traditional software. It usually takes less testing in LCSD platforms because the platform management team test and monitors the modules provided. Unit testing carries less importance than tradition development because the components are already unit tested, and developers usually integrate those using drag-and-drop fashion. Many platforms provide custom unit testing features for the application logic code added by the developers. The platform providers recommend running a security audit to check if there is a potential data exposure. Most of the issues faced in this phase are related to browser compatibility, lack of proper documentation to run automated tests, test coverage, issues using the third party functional testing tools like Selenium ($Q_{61210424}$), etc. Practitioners make general queries about running tests on low-code platforms ($Q_{46669690}$) and errors while running test ($Q_{47254010}$).

Deployment (22, 2.5%). LCSD platforms aim to make the *deployment and maintenance* phase smooth. Many platforms provide Application Life-Cycle Management tools to develop, debug, deploy, and maintain the staging and production server. However, LCSD developers still face challenges regarding deployment configuration issues ($Q_{46369742}$), version control, DNS configuration, performance issues, accessibility issues (i.e., sharing public URL of the application ($Q_{44136328}$, $Q_{53884162}$)), DNS configuration, etc. For example, in this discussion, a developer was having trouble accessing the app after deployment (e.g., "Google App Maker app not working after deploy" ($Q_{42506938}$)). In the accepted answer, a community member provides a detailed description to achieve that and points out the lack of *Official Documentation* for such a crucial task. There are a few queries about deploying application with custom URL, i.e., the domain name (e.g., "How to make friendly custom URL for deployed app" ($Q_{47194231}$)). In this case, it was difficult because the platform did not have native support.

Maintenance (24, 2.6%). At this stage, the application is released and needs maintenance support. The users sometimes find bugs that were not caught before and sometimes want new features that may spawn a new software development life cycle in agile or incremental development methodology. In this phase, developers face challenges regarding bugs in a low-code platform and difficulties in using different application maintenance features provided by the platform, such as event monitoring, collaboration, application design reusability, etc. Different LCSD platforms provide features on developers' role management, dashboard, and event monitoring. For example, in this question, a practitioner queries about the feasibility of role-based access control in an LCSD platform: "PowerApps : Implementing Role Based Security In Your PowerApps App" ($Q_{52762374}$). LCSD developers also find it difficult to determine/update versions of LCSD platforms ($Q_{45209796}$).

Summary of RQ2. We randomly sampled 900 questions from our dataset and manually examined the SDLC phase of the questions. We found an overwhelming majority of (85%) implementation phase-related questions in SO. Non-coding questions are generally discouraged in SO, so we found very few questions related to other phases (e.g., requirement analysis, deployment, etc.). We also find that LCSD developers find testing to be challenging for LCSD application due to the graphical nature of the SDKs, which can be hard to debug.

C. What LCSD topics are the most difficult to answer? (RQ3)

1) Approach: For each topic, we compute the difficulty of getting answers under a topic using three metrics: (1) Percentage of questions without any answer at all, (2) Percentage of questions without an accepted answer, (3) Average median time needed to get an accepted answer. In the same way we use the following four popularity metrics: (1) Average number of views, (2) Average number of favourites, (3) Average score, and (4) Average number of answers. Then we aim to determine the correlation between the difficulty and popularity of the topics. The first two difficulty metrics and the first three popularity metrics are used in several previous studies [7, 1, 2]. We use the Kendall Tau correlation measure [25] to find the correlation between topic popularity and topic difficulty. SO

TABLE II: Low-code software development topics, their popularity, and difficulty

Topic	Category	Popularity score				Difficulty score		
		Avg view	Avg fav.	Avg score	Avg #ans	W/O any ans.	W/O acc. ans.	Med. Hrs acc.
Dynamic Event Handling	Customization	833	1.23	0.54	0.86	37.2%	75.9%	9.8
Ext. Web Req Processing	Integration	785	1.24	0.62	1.04	23.3%	68.1%	15.7
Ext. API & Email Config	Integration	764	1.23	0.83	0.91	34.3%	70.4%	12.7
Dynamic Content Display	Customization	722	0.86	0.48	0.99	17.6%	65.2%	14.8
Cloud and On-Prem Conf	Adoption	578	1.18	0.85	1.01	23.8%	66.5%	16.8
Dynamic Form Controller	Customization	566	0.85	0.45	1.07	15.1%	57.4%	4.2
UI Adaptation	Customization	536	0.95	0.46	0.88	30.3%	68.2%	6.1
Dynamic Content Binding	Customization	507	1.16	0.36	0.94	22.4%	67.2%	24.9
Entity Relationship Mngmt	Database	485	1.09	0.48	0.93	29.5%	62.4%	6.9
Data Storage & Migration	Database	472	1.07	0.60	0.89	25.1%	69.3%	14.8
Client Server Comm & IO	Adoption	408	1.11	0.54	0.86	31.8%	67.8%	12.7
SQL CRUD	Database	359	1.04	0.45	0.92	22.1%	60.2%	7.6
Access Control & Security	Adoption	301	0.97	0.49	0.93	26.2%	69.9%	12.6
Average		**572**	**1.1**	**0.5**	**0.9**	26%	**67%**	12.3

TABLE III: Correlation between the topic popularity and difficulty

coefficient/p-value	View	Favorites	Score
% without acc. ans.	0.154/0.51	0.348/0.10	0.379/0.08
Hrs to acc. ans.	0.077/0.77	0.400/0.06	0.431/0.04
% without any answer	0.077/0.77	0.374/0.08	0.275/0.20

does not provide the data across a time-series for all metrics such as view count, score, etc. As such, our analysis offers as of time insight.

2) Results: Table II shows the topic difficulty using the three metrics, as noted above. These metrics allow us to understand the difficulty of getting a working solution [7, 1]. The *Dynamic Event Handling* under Customization has the highest average view count and the highest percentage of questions (76%) without an accepted answer. *Dynamic Content Binding* from the same category also has the highest median hours to get an accepted answer. Many questions in this topic relate to business logic customization in an LCSD platform, which is not familiar to other developers. For example, $Q_{51443599}$ asks, "How to add more data an array of objects in a lightning component?". This question has been asked around two years ago, viewed around 11K times and still active. It has three answers, but none of them is marked as accepted. Whereas, *Dynamic Form Controller* under Customization is the least difficult topic concerning the percentage of the questions without an accepted answer (57%) and median time (4.2 hours) to get a solution. It is because questions related to form design and validation have good *community support*. The same is also true for *SQL CRUD*, which has around 7.6 median hours for accepted answers.

The *External Web Req Processing* under *Integration* has the highest percentage of questions and the highest average favourite count. The *Cloud and On-Prem Conf* under *Platform* adoption have the highest average score. The discussions are about setting and maintaining cloud configuration and migrating on-prem data to the server in this topic. For example, in $Q_{44727285}$, a practitioner is asks "Migrate Salesforce data from one org to another", and in $Q_{3016015}$, a new practitioner is making a general low-code platform related query "Are there any serious pitfalls to Outsystems Agile Platform?". This question has a very high score because lots of new low-code platform developers have faced this issue. *Access Control & Security* under Platform Adoption is the least popular topic in terms of average view count. In this topic, the questions are about deploying the application, its security and access control. For example, in $Q_{17886545}$, a practitioner says, "Can't Connect to SalesForce in C#". He also explains there is a security token error, and in the documentation, it is not mentioned how to configure that. Many of these questions are not general, and so it has a low average view-count.

Correlation between topic difficulty and popularity. In Table III, we present nine correlation measures using three difficulty and popularity metrics from Table II. Our result shows no statistically significant correlation between the popularity and difficulty metrics since for eight out of nine correlation values are > 0.05. Therefore, we cannot say that the least popular topics are the most difficult ones and vice versa. For example, Access Control & Security is one of the least popular topics but considered among the most difficult topics (Table II). However, this observation does not hold for Dynamic Event Handling, which is the most popular and among the most difficult topics.

Summary of RQ3. We compute seven popularity and difficulty metrics for each topic using metrics such as view count, percentage of questions without an accepted answer, etc. We find that questions related to the topic "Dynamic Event Handling" from the customization category are the most difficult (to get an accepted answer) but also the most popular (average view count). Overall, we do not see any significant correlation (positive/negative) between topic difficulty and popularity metrics.

V. Discussions

In this section, we discuss the evolution of low-code and LCSD related discussions with respect to different topics. Then we discuss the implications of our findings.

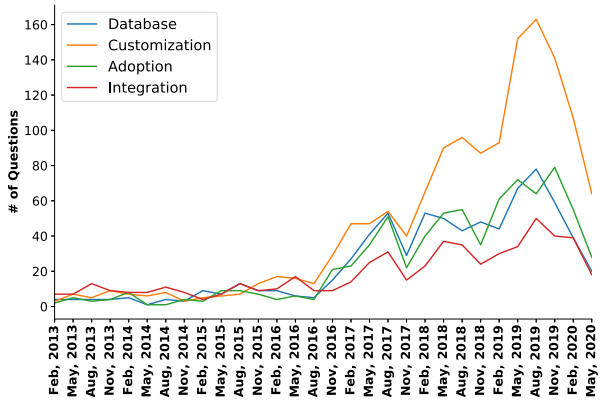

Fig. 6: Low-code topic category evolution over time.

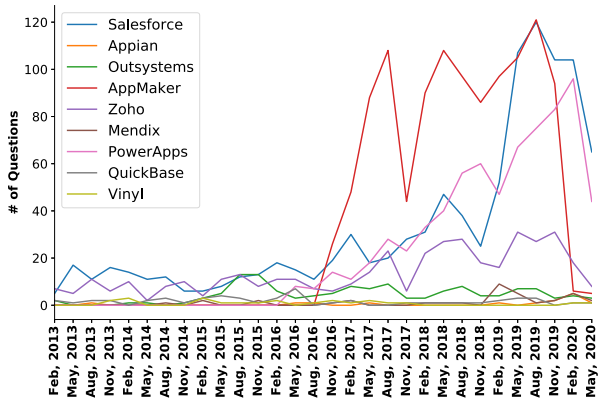

Fig. 7: Low-code Platforms evolution over time

A. Evolution of LCSD topics

We measure the growth of our four high-level topic categories over time to better understand the evolution of LCSD. We measure the absolute growth, i.e., the total number of questions in a category over time. Figure 6 shows that all four of our topic categories are increasing monotonically. This trend indicates that the LCSD approach is gaining more community attention over time, especially after 2016.

We further analyze two sudden swings in the number of questions. First, we find an increase of questions in every category after mid-2016, especially for questions about the *Customization* category. Google released App Maker [21] for public use in 2016, which introduced many discussions on LCSD customization. Figure 7 confirms it and shows a spike in questions about Google App Maker during that time. The second case is that at the beginning of 2020, there is a sharp decline in SO discussions. In Jan 2020, Google announced that they would no longer release new features for Google App Maker and discontinue it by 2021 [22]. It created unrest among the developer community as they were trying to verify this information ($Q_{59947680}$) and to explore alternatives (e.g., $Q_{59985750}$). Figure 7 also shows the sudden drop in the number of questions asked about Google App Maker starting Jan 2020.

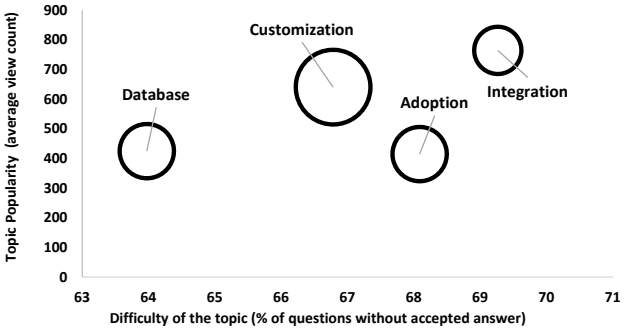

Fig. 8: Low-code topic categories popularity vs. difficulty

B. Implications of Findings

This study can help the low-code community to focus on the pressing issues on the LCSD paradigm. We discuss the implications of our study findings by stakeholders below.

LCSD Platform Providers. In order to better understand the issues of LCSD, we present a bubble chart 8 that presents the positions of low-code categories in terms of popularity vs. difficulty. In this study, we use the average number of view-count and percentage of questions without accepted answers as a proxy for the topic category popularity and difficulty, respectively [1]. The size of the bubble depends on the number of questions for that particular topic category. Figure 8 shows that *Integration* is the most popular as well as most difficult topic category. On the other hand, *Database* remains the least difficult category due to the superior database support by LCSD platforms. As shown in Figure 8, *Customization* is the largest and prevalent low-code topic category. Many new practitioners make queries regarding LCSD platforms, learning resources, basic application and UI customization, and how to get started with this new emerging technology. We find that *Documentation* related queries are both very popular and difficult. Our findings also suggest that many practitioners still face challenges during testing and debugging. Consequently, many of the questions on this topic remain unanswered. It reveals that to ensure smooth adoption of the LCSD platform, the platform providers should provide better and effective documentation and provide learning resources to reduce entry-level barriers and smooth out the learning curve.

LCSD Practitioners/Developers. LCSD abstractions and the platform's feature limitations sometimes make it very difficult to customize and debug. Our finding shows that the practitioners find third party service *Integration* and *Platform Feature* category most difficult. It provides valuable insights for project managers to manage resources better (i.e., human resources and development time). LCSD platform enables practitioners with diverse experience to contribute to the development process even without a software development background. However, our finding shows that practitioners find debugging, application accessibility, and documentation challenging. Hence, the practitioners should take the necessary

steps to understand the tradeoffs LCSD platforms' features deeply. The project manager should adopt specific strategies to learn to customize, debug, and test the application.

LCSD Researchers & Educators. We find that the LCSD paradigm's challenges can be different from traditional software development [48]. Researchers can focus on the most popular and difficult topic category *Integration* and develop a set of metrics to automatically detect documentation quality on third-party service APIs. Simultaneously, researchers can study how to provide better tools for practitioners to customize the application. Security is an open research opportunity for such platforms as a security vulnerability in such platforms or frameworks could compromise millions of applications and users [29]. Researchers can come up with better testing approaches to ensure faster development and dependability. Educators can also benefit from the results presented in Figure 8 to prioritize their focus on different topics such as *Database, Customization, and Third-party API Integration*.

VI. Threats to Validity

Internal validity threats relate to the authors' bias while conducting the analysis. We mitigate the bias in our manual labeling of topics and LCSD phases by consulting the labels among multiple authors. Four of the authors actively participated in the labelling process. The third author reviewed the final labels and refined the labels by consulting with the first author. **Construct Validity** threats relate to the errors that may occur in data collection like, identifying relevant LCSD tags. To mitigate this, we examine all the tags that we find in the low-code related questions. Then we expanded our tag list using state-of-art approach [7, 1, 2, 46]. Another potential threat is the topic modeling technique, where we choose K = 13 as the optimal number of topics for our dataset B. This optimal number of topics have a direct impact on the output of LDA. We experimented with different values of K following related works [1, 7]. We used the coherence score and manual examination to find K's optimal that gives us the most relevant and generalized low-code related topics. **External Validity** threats relate to the generalizability of our findings. Our study is based on data from developers' discussion on SO. However, there are other forums LCSD developers may use to discuss. Nevertheless, we believe using SO's data provides us with generalizability because SO is a widely used Q&A platform for developers. To ensure good quality discussion, we only use posts with non-negative scores. However, we also believe this study can be complemented by including discussions from other forums, surveying and interviewing low-code developers.

VII. Related Work

Research on low-code development. LCSD is a relatively new technology, and there are only a few research works in this domain. There is some research on how this emerging technology can be used in different software applications [20] or for automating business process in manufacturing [60]. Sipio et al. [17] present the benefits and future potential of LCSD by sharing their experience of building a custom recommendation system in the LCSD platform. Kourouklidis et al. [27] discuss the low-code solution to monitor the machine learning model's performance. Sahay et al. survey LCDP and compare different LCDPs based on their helpful features and functionalities [48]. Khorram et al. [26] analyse commercial LCSD platforms and present a list of features and testing challenges. Ihirwe et al. [24] analyse 16 LCSD platforms and identifies what IoT application-related features and services each platform provides. All these research works compare LCSD platforms and their support on the different types of applications [3]. To the best of our knowledge, ours is the first empirical study of LCSD and platforms based on developer discussions.

Topic Modeling in Software Engineering. Our motivation to use topic modeling to understand LCSD discussions stems from existing research in software engineering that shows that topics generated from textual contents can be a good approximation of the underlying *themes* [15, 50, 52]. Topic models are used recently to understand software logging [28] and previously for diverse other tasks, such as concept and feature location [16, 38], traceability linking (e.g., bug) [42, 6], to understand software and source code history evolution [23, 54, 53], to facilitate code search by categorizing software [55], to refactor software code base [11], as well as to explain software defect [14], and various software maintenance tasks [51, 50]. The SO posts are subject to several studies on various aspects of software development using topic modeling, such as what developers are discussing in general [10] or about a particular aspect, e.g., concurrency [2], big data [7], chatbot development [1]. We are aware of no previous research on understanding the LCSD discussions in SO.

VIII. Conclusions

LCSD is a new paradigm that enables the development of software applications with minimal hand-coding using visual programming. We present an empirical study that provides insights into the types of topics low-code developers discuss in Stack Overflow (SO). We find 13 low-code topics in our dataset of 4.6K SO posts (question + accepted answers). The posts are collected based on 19 SO tags belonging to the popular nine LCSD platforms during our analysis. We categorize them into four high-level groups, namely Customization, Platform Adoption, Database, and Integration. Our findings reveal that developers find the external API Integration topic category the most challenging and the Database category least difficult. Dynamic Event Handling is the most popular, as well as the most challenging topic. We find a severe lack of good tutorial based documentation that deters the smooth adaptation of LCSD. We hope that all of these findings will help various LCSD stakeholders (e.g., LCSD platforms, practitioners, SE researchers) to take necessary actions to address the various LCSD challenges. Since the growth indicates that this technology is likely to be widely adopted by various companies for their internal and customer-facing applications, platform providers should address the prevailing developers' challenges.

REFERENCES

[1] Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem, and Emad Shihab. Challenges in chatbot development: A study of stack overflow posts. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 174–185, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375177. doi: 10.1145/3379597.3387472. URL https://doi.org/10.1145/3379597.3387472.

[2] Syed Ahmed and Mehdi Bagherzadeh. What do concurrency developers ask about? a large-scale study using stack overflow. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450358231. doi: 10.1145/3239235.3239524. URL https://doi.org/10.1145/3239235.3239524.

[3] Ana Nunes Alonso, João Abreu, David Nunes, André Vieira, Luiz Santos, Tércio Soares, and José Pereira. Towards a polyglot data access layer for a low-code application development platform. *arXiv preprint arXiv:2004.13495*, 2020.

[4] appian. Appian platform overview. Available: https://www.appian.com/. [Online; accessed 5-January-2021].

[5] Rajkumar Arun, Venkatasubramaniyan Suresh, CE Veni Madhavan, and MN Narasimha Murthy. On finding the natural number of topics with latent dirichlet allocation: Some observations. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 391–402. Springer, 2010.

[6] Hazeline U Asuncion, Arthur U Asuncion, and Richard N Taylor. Software traceability with topic modeling. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 95–104. IEEE, 2010.

[7] Mehdi Bagherzadeh and Raffi Khatchadourian. Going big: A large-scale study on what big data developers ask. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 432–442, New York, NY, USA, 2019. ACM.

[8] Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. Mining questions asked by web developers. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 112–121, 2014.

[9] Alan Bandeira, Carlos Alberto Medeiros, Matheus Paixao, and Paulo Henrique Maia. We need to talk about microservices: an analysis from the discussions on stackoverflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 255–259. IEEE, 2019.

[10] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 19(3):619–654, 2014.

[11] Gabriele Bavota, Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk, and Andrea De Lucia. Methodbook: Recommending move method refactorings via relational topic models. *IEEE Transactions on Software Engineering*, 40(7):671–694, 2014.

[12] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.

[13] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3 (4-5):993–1022, 2003.

[14] Tse-Hsun Chen, Stephen W. Thomas, Meiyappan Nagappan, and Ahmed E. Hassan. Explaining software defects using topic models. In *9th working conference on mining software repositories*, pages 189–198, 2012.

[15] Tse-Hsun (Peter) Chen, Stephen W. Thomas, and Ahmed E Hassan. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*, 21(5): 1843–1919, 2016.

[16] Brendan Cleary, Chris Exton, Jim Buckley, and Michael English. An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering*, 14:93–130, 2009.

[17] Claudio Di Sipio, Davide Di Ruscio, and Phuong T Nguyen. Democratizing the development of recommender systems by means of low-code platforms. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–9, 2020.

[18] Stack Exchange. Stack exchange data dump . Available: https://archive.org/details/stackexchange, 2020. [Online; accessed 5-January-2021].

[19] Sally Fincher and Josh Tenenberg. Making sense of card sorting data. *Expert Systems*, 22(3):89–93, 2005.

[20] Meg Fryling. Low code app development. *J. Comput. Sci. Coll.*, 34(6):119, April 2019. ISSN 1937-4771.

[21] googleappmaker. Google App Maker platform overview. Available: https://developers.google.com/appmaker. [Online; accessed 5-January-2021].

[22] googledisc. Google App Maker will be shut down on January 19, 2021. https://workspaceupdates.googleblog.com/2020/01/app-maker-update.html. [Online; accessed 5-January-2021].

[23] Jiajun Hu, Xiaobing Sun, David Lo, and Bin Li. Modeling the evolution of development topics using dynamic topic models. In *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering*, pages 3–12, 2015.

[24] Felicien Ihirwe, Davide Di Ruscio, Silvia Mazzini, Pierluigi Pierini, and Alfonso Pierantonio. Low-code engineering for internet of things: A state of research. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381352. doi: 10.1145/3417990.3420208. URL https://doi.org/10.1145/3417990.3420208.

[25] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1):81–93, 1938.

[26] Faezeh Khorram, Jean-Marie Mottu, and Gerson Sunyé. Challenges & opportunities in low-code testing. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381352. doi: 10.1145/3417990.3420204. URL https://doi.org/10.1145/3417990.3420204.

[27] Panagiotis Kourouklidis, Dimitris Kolovos, Nicholas Matragkas, and Joost Noppen. Towards a low-code solution for monitoring machine learning model performance. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–8, 2020.

[28] Heng Li, Tse-Hsun (Peter) Chen, Weiyi Shang, and Ahmed E. Hassan. Studying software logging using topic models. *Empirical Software Engineering*, 23:2655–2694, 2018.

[29] Guanjun Lin, Sheng Wen, Qing-Long Han, Jun Zhang, and Yang Xiang. Software vulnerability detection using deep neural networks: a survey. *Proceedings of the IEEE*, 108(10):1825–1848, 2020.

[30] Mario Linares-Vásquez, Bogdan Dit, and Denys Poshyvanyk. An exploratory analysis of mobile development issues using stack overflow. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 93–96. IEEE, 2013.

[31] Edward Loper and Steven Bird. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[32] lowcodewiki. Low-code development platform . Available: https://en.wikipedia.org/wiki/Low-code_development_platform. [Online; accessed 5-January-2021].

[33] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. *http://mallet. cs. umass. edu*, 2002.

[34] mendix. Mendix platform overview. Available: https://www.mendix.com/. [Online; accessed 5-January-2021].

[35] Stack Overflow. *Stack Overflow Questions*. https://stackoverflow.com/questions/, 2020. Last accessed on 14 November 2020.

[36] John Pane and Brad Myers. *More Natural Programming Languages and Environments*, pages 31–50. Springer, 10 2006. ISBN 978-1-4020-4220-1. doi: 10.1007/1-4020-5386-X_3.

[37] pcmag. The Best Low-Code Development Platforms. Available: https://www.pcmag.com/picks/the-best-low-code-development-platforms. [Online; accessed 5-January-2021].

[38] Denys Poshyvanyk, Yann-Gaël Guéhéneuc, Andrian Marcus, Giuliano Antoniol, and Václav T Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering*, 33(6):420–432, 2007.

[39] powerapps. Microsoft power apps platform overview. Available: https://powerapps.microsoft.com/en-us/. [Online; accessed 5-January-2021].

[40] quickbase. Quickbase platform overview. Available: https://www.quickbase.com/product/product-overview. [Online; accessed 5-January-2021].

[41] C Ramasubramanian and R Ramya. Effective pre-processing activities in text mining using improved porter's stemming algorithm. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(12):4536–4538, 2013.

[42] Shivani Rao and Avinash C Kak. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *8th Working Conference on Mining Software Repositories*, page 43–52, 2011.

[43] Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[44] Martin P. Robillard, Eric Bodden, David Kawrykow, Mira Mezini, and Tristan Ratchford. Automated API property inference techniques. *IEEE Transactions on Software Engineering*, page 28, 2012.

[45] Michael Röder, Andreas Both, and Alexander Hinneburg. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining*, pages 399–408, 2015.

[46] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, 21(3):1192–1223, 2016.

[47] John R Rymer, Rob Koplowitz, and Salesforce Are Leaders. The forrester wave(tm) low-code development platforms for ad&d professionals, q1 2019. 2019.

[48] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178. IEEE, 2020.

[49] salesforce. Salesforce platform overview. Available: https://www.salesforce.com/in/?ir=1. [Online; accessed 5-January-2021].

[50] Xiaobing Sun, Bin Li, Yun Li, and Ying Chen. What information in software historical repositories do we need to support software maintenance tasks? an approach based on topic model. *Computer and Information Science*, pages 22–37, 2015.

[51] Xiaobing Sun, Bixin Li, Hareton Leung, Bin Li, and Yun Li. Msr4sm: Using topic models to effectively mining software repositories for software maintenance tasks. *Information and Software Technology*, 66:671–694, 2015.

[52] Xiaobing Sun, Xiangyue Liu, Bin Li, Yucong Duan, Hui Yang, and Jiajun Hu. Exploring topic models in software engineering data analysis: A survey. In *17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 357–362, 2016.

[53] Stephen W. Thomas, Bram Adams, Ahmed E Hassan, and Dorothea Blostein. Modeling the evolution of topics in source code histories. In *8th working conference on mining software repositories*, pages 173–182, 2011.

[54] Stephen W. Thomas, Bram Adams, Ahmed E Hassan, and Dorothea Blostein. Studying software evolution using topic models. *Science of Computer Programming*, 80(B):457–479, 2014.

[55] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *6th international working conference on mining software repositories*, pages 163–166, 2009.

[56] Gias Uddin and Foutse Khomh. Automatic summarization of api reviews. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 159–170. IEEE, 2017.

[57] P Vincent, K Lijima, Mark Driver, Jason Wong, and Yefim Natis. Magic quadrant for enterprise low-code application platforms. *Retrieved December*, 18:2019, 2019.

[58] vinyl. Vinyl platform overview. Available: https://zudy.com/. [Online; accessed 5-January-2021].

[59] Zhiyuan Wan, Xin Xia, and Ahmed E Hassan. What is discussed about blockchain? a case study on the use of balanced lda and the reference architecture of a domain to capture online discussions about blockchain platforms across the stack exchange communities. *IEEE Transactions on Software Engineering*, 2019.

[60] Robert Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52:376–381, 01 2019. doi: 10.1016/j.ifacol.2019.10.060.

[61] Jason Wong, Mark Driver, and Paul Vincent. Low-code development technologies evaluation guide, 2019.

[62] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, 2016.

[63] zohocreator. Zoho Creator platform overview. Available: https://www.zoho.com/creator/. [Online; accessed 5-January-2021].