SUMMARY  Mining software repositories allow software practitioners to improve the quality of software systems and to support maintenance based on historical data. Such data is scattered across autonomous and heterogeneous information sources, such as version control, bug tracking and build automation systems. Despite having many tools to track and measure the data originated from such repositories, software practitioners often suffer from a scarcity of the techniques necessary to dynamically leverage software repositories to fulfill their complex information needs. For example, answering a question such as “What is the number of commits between two successful builds?” requires tiresome manual inspection of multiple repositories. As a solution, this paper presents a conceptual framework and a proof of concept visual query interface to satisfy distinct software quality related information needs of software practitioners. The data originated from repositories is integrated and analyzed to perform systematic investigations, which helps to uncover hidden relationships between software quality and trends of software evolution. This approach has several significant benefits such as the ability to perform real-time analyses, the ability to combine data from various software repositories and generate queries dynamically. The framework evaluated with 31 subjects by using a series of questions categorized into three software evolution scenarios. The evaluation results evidently show that our framework surpasses the state of the art tools in terms of correctness, time and usability.

key words: software analysis, mining software repositories, visual query interfaces, static grammar

1. Introduction

Software engineering landscape has significantly changed over the last decade with the ever-increasing complexity of software products as well as the growth of the development teams into hundreds of people. With the increase in team size and tool support, the amount of data that software practitioners exposed to have grown to a point, where it is cumbersome to examine them manually. As a result, recent years have witnessed extensive studies on mining software repositories for extracting data related to software systems in which the information seekers can find out various types of valuable information [1], [2].

Software data is distributed across multiple heterogeneous software repositories such as version control systems (VCS), issue tracking systems, build-automation systems or any additional quality assurance platforms. Despite having many tools to track and measure such data, software practitioners often suffer from a scarcity of the techniques necessary to leverage these potentially powerful information sources towards fulfilling their complex information needs [3], [4]. For example, version control systems, issue tracking systems, etc. provide a plethora of features of their own. However, answering a query such as “What is the number of commits between two successful builds?” requires tiresome manual inspection of repositories associated with multiple technical systems.

Existing approaches that enable the querying and integration of different information sources often do not allow developers to formulate dynamic queries and obtain real-time results. This work, therefore, devised a conceptual framework and a proof-of-concept visual query interface that enables software practitioners to dynamically query for various diverse evolutionary aspects of a software system. It facilitates queries related to code quality, development history, build-information, as well as to defect and issue management. It is not constrained to fulfilling the information needs of software practitioners, but essentially it empowers software quality improvements in software projects. The query interface built by adhering to the underline concepts of data mashups, which allows users to create query workflows visually by dragging and dropping the selected components. The component integration logic based on a set of grammar rules, which is kept hidden behind the visual query interface. To rationalize the integration, we employ Goal-Question-Metrics (GQM) approach [5], which permits to quantify the goals in a meaningful way by exploiting a set of questions and related software metrics. This paper intends to address the following research questions (RQs):

- RQ1: How can we provide a framework with an interactive query interface to facilitate real-time software analysis by exploiting heterogeneous software artifacts?
- RQ2: When software practitioners use such a framework to perform software analysis tasks, can we witness an improvement over the state-of-the-art regarding accuracy, time efficiency, and usability?

The rest of the paper is structured as follows. We highlight the Related work in Sect. 2. Next, we present our Approach in Sect. 3 followed by the proof-of-concept implementation in Sect. 4. Then the Evaluation of our query framework is covered in Sect. 5. We conclude the paper with the conclusion and future work in Sect. 6.
Sillito et al. described 44 questions that developers ask when examining information about large-scale software development activities. Some of these questions are mainly focused on information that is directly related to source code, while others concentrate on a wide range of software analyses. On the other hand, the information needs of software practitioners are also exceedingly dynamic.

2. Related Work

Software repositories such as version control repositories, bug tracking repositories, and other Continuous Integration (CI) tools are highly data-intensive and undergo frequent changes. This dynamicity, and the diversity of such information, is the foundation for many kinds of valuable software analyses. On the other hand, the information needs of software practitioners are also exceedingly dynamic.

2.1 Information Needs of Software Practitioners

Information needs during software development activities have been examined in several previous studies. Some studies are mainly focused on information that is directly related to source code, while others concentrate on a wide range of questions about large-scale software development activities. Sillito et al. described 44 questions that developers ask when programming [6]. The research conducted by De Alwis and Murphy [7] recognized 36 questions that mainly associated with source code. This identification of questions was based on literature, blogs as well as their own experiences as software practitioners. 21 types of questions have identified and reported in the study conducted by Ko and colleagues by shadowing professional developers at work [8]. LaToza et al. utilized their personal experience as developers and proposed 19 problems associated with information needs. The results were presented based on a survey of specialized developers about the significance of each issue [9]. LaToza and Myers then clustered these questions into 21 categories and 94 distinct questions [10]. That was an eye-opener to build new software analysis tools and programming languages to help developers to find answers for their daily information needs. The study of Roehm et al. [2] reports on an observational study of professional developers from software companies, investigating how developers comprehend software with a particular emphasis on the strategies, information needs, and the available tools. They found that wherever possible, developers seem to prefer strategies that avoid comprehension, due to both time and mental effort needed. It was further observed that state-of-the-art tools in program comprehension are either unknown or rarely utilized. In addition to the question catalogs, Ghezzi and Gall [11] presented a set of usage scenarios in the context of software analysis.

2.2 Information Integration

Voinea and Talea [12] presented a framework enabling the integration of data extraction from version control repositories with data analysis. They have used the framework to find answers to software engineering questions addressing a specific project. Fritz and Murphy [13] introduced an information fragment model together with a prototype tool that helps to automate the composition of different kinds of information. It allows developers to choose how to display the composed information easily. Li and Zhang [14] described how to apply Semantic Web techniques to integrate object-oriented software engineering data from different sources. They also show how the combined data can facilitate answering complex queries about large-scale software projects through a case study on Eclipse. SQA Mashup [15] integrates information from various Continuous Integration tools such as GitHub, Jenkins, and SonarQube. This idea has later been enhanced for rule-based activity profiles for continuous integration environments [16].

In contrast, our approach has several noteworthy differences. First, our method is not limited to satisfy the information needs of software developers. Instead, it could be used by any interested practitioner such as developer, tester or project manager. Second, we move a step further by allowing them to create dynamic queries and obtain the results on-the-fly, which has not thoroughly addressed in the software analysis domain [17]. Finally, we rationalize the data integration by employing GQM approach.

3. Approach

This research work aims at leveraging software artifacts to facilitate real-time software analysis by utilizing the large amounts of interesting and potentially useful information stored in various artifacts. We first investigate the potential challenges in fulfilling the complex information needs of software practitioners and the challenges associated with developing a query framework for leveraging software artifacts. Then we present our approach to data integration by employing GQM approach concerning the characteristics and sub-characteristics of ISO/IEC 25010 quality model. Next, we explain our approach of measuring quality sub-characteristics. Measuring is done by identifying a set of associated goals, questions, and metrics for each sub-characteristic. Finally, we provide an overview of the visual query framework, which was developed as a proof-of-concept to overcome the above mentioned potential challenges. Figure 1 presents an overview of this research.

3.1 Challenges

Leveraging software artifacts to fulfill the information needs of software practitioners has grown into two crucial technical and conceptual challenges: real-time, up-to-date data extraction from repositories and meaningful integration of data originated from such repositories.

Real-time data extraction: Until recently, data extraction from the repositories and analysis has been cumbersome due to several reasons such as limited access to repositories and the complexity of data extraction [3]. Modern day software artifacts enriched with service-oriented architecture (SOA), which provides a reasonable easiness in real-time data extraction. Software repositories typically undergo a series of changes during the lifetime of software projects. For instance, during the development of a software product, version control repositories and other continuous...
integration repositories are subject to various kinds of frequent changes. Therefore, real-time, up-to-date information extraction from software repositories is mandatory for better software analyses.

Linking data between software artifacts: Though the data is available on service-oriented software artifacts, bringing data together and making sense of it becomes increasingly difficult. This difficulty has identified as an essential future direction in mining software repositories [18]. For example, fulfilling the information needs such as “What is the contribution of a developer working on a project” is not straightforward. Rather it is a stepwise process, which consists of answering two sub-questions: “Number of commits of the developer in the project” and “Number of issues fixed by the same developer”. To obtain such information, it is required to extract and combine information from a version control repository and a bug repository. Likewise, many software repositories need to be mined and combined to answer questions like “What is the contribution style in a given project?” [12], “What is the maintenance risk of a given developer leaving a given project?” [12], “Which files were affected by a given bug?” [19], “How many developers worked on an entity?” [19], “How was the effort distributed among them?” [19]. In this paper, the problem of interlinking the data available in various artifacts is tackled by adhering to a service-oriented approach together with a set of static grammar rules, which describes in Sect. 4.1.

There are several challenges associated with developing a query framework to leverage large-scale multiple software repositories. In particular, it requires accessing them programatically, establishing an infrastructure to download data and applying meaningful operations on it. As a result of software analytics tasks do not always follow a simple structure, it makes the manual process tedious for software researchers and practitioners. Therefore, query-framework based approaches to software analytics should provide an acceptable degree of expressiveness when it comes to solving more complex analytics tasks. Our approach aims at providing a framework that could improve the expressiveness of the analytics queries written by software practitioners and researchers. The expressiveness is expressed in terms of Flexibility, Generality, and Extensibility.

Flexibility: In the context of software analytics, novel frameworks need to be purposely designed to enable flexible querying of data. That means, for example, it should not limit to a particular set of queries. Much existing work does not allow ad-hoc queries. The approaches that facilitate ad-hoc queries are limited to a pre-defined data sources, where the querying and analysis scope is limited.

Generality: Query frameworks should not be specific to a particular information need. Much existing work either operate on a specific data sources or a specific portion of data sources. Modern software analytics research focuses on a diverse range of analytics such as finding the impact of continuous integration on code reviews [20], finding identical code fragments in Github and stack overflow [21], and finding the influence of contributors’ involvement in build status [22]. Therefore, novel frameworks are needed to accomplish such manifold requirements.

Extensibility: Incorporating a new data source to broaden the analysis spectrum should be straightforward without breaking the underpinning design rationale of the framework. For example, several recent studies have focused on subsuming non-traditional data sources such as video tutorials. To the best of our knowledge, existing domain-specific frameworks do not explicitly facilitate such extensions. On the other hand, achieving flexibility (i.e., allowing ad-hoc queries) and generality (i.e., not being limited to a particular information need) also requires data extraction, integration, and analysis of multiple heterogeneous data sources.

![Fig. 1 Overview of the approach](image-url)
3.2 Goal-Oriented Software Quality

Software measurements are useful only if they measure what is exactly needed by software practitioners, rather than measuring what is convenient to measure. Determining what is required (i.e., attributes to measure) depends on the objectives of software practitioners. Therefore, regardless of the benefits of having a general collection of software measures that could apply to software quality assessments, it has been proven that goal-based software measurements are much more successful in practice [23]. The fundamental assumption of goal-based measurement is that measures must directly associate with the organizational goals. In that way, GQM [5] identify a firm goal, refines this goal into a series of questions, and defines a set of metrics that should provide the information to answer these questions. In this paper, the goals are determined to meet a specific quality sub-characteristic in ISO/IEC 25010 [24] model, which is an extension of ISO/IEC 9126 [25] quality model. ISO/IEC quality models provide an international standard that has been agreed upon by the community and several countries [26]. It defines a standard set of features that are directly associated with software product quality and therefore used as the frame of reference for communication about software product quality in our approach. ISO/IEC 25010 product quality model classifies quality attributes into eight characteristics: functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability, and portability. Each quality characteristic consists of a collection of sub-characteristics. For example, Performance efficiency is a major quality characteristic in the ISO/IEC 25010 model, of which Resource utilization is a sub-characteristic.

Once the goals are defined at a conceptual level, we then examine the operational questions (e.g., Q1, Q2, ..., Qn) that must be answerable to achieve the goals. However, there can be situations where the operational questions are at a comparatively abstract level. We tackled this problem by breaking down the questions into sub-questions (e.g., SQ1, SQ2, ..., SQn). A similar approach has been presented by Washizaki et al. [27]. Then the measurements have been firmed up with the specific metrics (e.g., M1, M2, ..., Mn) required formulating the answers to those questions. Figure 2 showcases the tree structure starting from high-level quality characteristics down to measurement units (i.e., software metrics).

Different metrics have different impacts on sub-questions. Therefore, in our approach, relative weights are assigned to the metrics based on its importance in answering the sub-question. Similarly, weights are assigned to sub-questions based on its relevance to addressing the high-level question. A similar procedure is repeated to the questions if a particular goal needs to be reached by answering more than one question. However, there is no consistently established methodology to assign weights to individual indicators (i.e., Metrics, Sub-questions and Questions) before combining them into a composite indicator.

![Fig. 2 Employing GQM to quality characteristics](image)

Different weights may assign to individual indicators based on the significance level of the indicator in the considered problem domain as well as the domain of interest of the information seeker (i.e., developer, QA engineer, project manager). Even though the weight assignment is slightly out of the scope of this paper, we emphasize the importance of keeping weighting process explicit and transparent since it has a significant impact on the final result. Equations (1), (2) and (3) describes the calculation and weight assigning process.

\[
vG_i = \sum_{i=1}^{n} (vQ_i \times wQ_i)
\]

\[
vQ_i = \sum_{i=1}^{n} (vSQ_i \times wSQ_i)
\]

\[
vSQ_i = \sum_{i=1}^{n} (vMi \times wMi)
\]

where:

- \(vG_i\) = quantitative value obtained for Goal,
- \(\omega Q_i\) = weight of the Question,
- \(vQ_i\) = quantitative value obtained for Question,
- \(\omega SQ_i\) = weight of the Subquestion,
- \(vSQ_i\) = quantitative value obtained for Subquestion,
- \(\omega M_i\) = weight of the Metric,
- \(vM_i\) = quantitative value obtained for Metric.

Software metrics are commonly characterized at micro-level. Therefore, metrics should be aggregated to provide insights into the evolution at the macro-level [28]. According to the Eq. (3), aggregated metrics values offer the answers to the sub-questions. Similarly, as per the Eqs. (2) and (1), aggregated values of sub-questions provide answers to the questions, and finally, aggregated values of questions define the goals operationally.
3.3 Measuring Quality Sub-Characteristics

This section overviews our approach to measure quality sub-characteristics concerning a high-level goal. To showcase the concept clearly, we picked a single quality sub-characteristic from each quality characteristic defined in the ISO/IEC 25010 model. Then we meaningfully decomposed the sub-characteristic into a set of questions, and finally, the questions are boiled down to a measurable set of software metrics. Table 2 evidently presents how we decomposed quality attributes into measurable metrics. However, the idea was to materialize our concept of decomposing quality attributes to metrics as a research contribution, but not to present a complete set of decomposition for all the quality attributes.

For example, consider the first entry in Table 2. It refers to ISO/IEC 25010 quality characteristic Functional Suitability and one of its sub-characteristics Functional Correctness. As described previously, quality sub-characteristic Functional Correctness can be measured by defining an associated goal: to evaluate the degree of closeness of a measured or calculated quantity to its actual value from the quality assurance engineer’s viewpoint. To reach this goal one of the main questions to be answered would be What is the degree of systematic and random errors in software?. This question is at a relatively abstract level and directly measuring it is not straightforward. Therefore, it has been broken down into two sub-questions: What is the failure ratio between the releases? and What is the success rate in test cases?. The sub-questions are at a fine-grained level and therefore can be answered by utilizing a set of software metrics: Number of build failures, unsuccessful builds between two releases and Unit test success. In that way, the measured values for such metrics provide means to assess the quality characteristic Functional Suitability.

4. Interactive Query Interface for Dynamic Software Analysis

In software analytics projects, the overall validity of the findings limited by the quality of data extraction. For this reason, timely and precise data extraction considered as one of the key factors in software analytics projects [29]. Another key factor to be considered is that software artifacts frequently co-change with time. Therefore, timely data extraction is essential in performing timely analyses. In this research, as a proof-of-concept, an interactive visual query interface (SA-FLY) has been developed to facilitate real-time data extraction and meaningful integration. Figure 3 provides an overall view of SA-FLY platform and its three main constituents: service tank, pipe engine, and query interface.

Service Tank: Service tank acts as a service repository, where the users can select the services based on their information needs. It binds continuous integration tools by exposing their functionalities via standard RESTful service interfaces. RESTful interfaces are stick to the architectural style of representational state transfer (REST), which widely used among web services community. For example, commit data is available on GitHub and Bitbucket repositories. Jenkins provides build-related information such as build date and build versions. Jira offers bug-related data such as bug fixes, assignees, and assigned dates.

Pipe Engine: Pipe engine contains two main components: data extractors and operators. Data extractors act as web service clients that are used to extract data from CI tools. For example, we have implemented data service operators such as Get Commits, Get Builds and Get Issues to retrieve the details such as commits, build details, and coding issues of a software project. For instance, Get Commit injects commit details such as commit message, committer name, email, and commit date from the repository. Such operators act as web service clients, and they extract data by using the REST endpoints of the CI tools. A set of low-level operators has been developed for each data source to filter and integrate the extracted data as per the requirements of the user. For example, the Filter operator can be used to filter the data based on a condition specified by the user. Filtering data by a specific date range, filtering the commits of a particular developer and filtering the issues fixed by a specific developer are examples of that nature. The filtered data can be saved, and it can be further utilized to construct relationships between the data sets. The extracted data will be integrated using pipe operators. CI tools provide JSON data in different hierarchical manner. The structure of the JSON (JavaScript Object Notation) data plays a major role in developing the query integration logic. Due to that limitation, it is not possible to dynamically use the low-level operators such as SELECT or FILTER as common operators. To tackle that problem, we have developed different sets of dedicated operators for diverse data sources.

Query Interface: Currently SA-FLY provides a customizable GUI and a default GUI. The Customizable GUI is one of the leading contributions of this approach as it allows software practitioners to create the queries by dragging and dropping components dynamically. SA-FLY users need to have a basic understanding in the software analysis domain since a query needs to be generated in the form of a workflow. The components are available in the left side panel of SA-FLY, which can be categorized as data extractors and data operators as described above. To obtain the results correctly, such components have to be appropriately connected to each other.

4.1 Data Extraction and Integration with SA-FLY

One of the main contributions of this paper is to provide a mechanism to integrate the data originated from heterogeneous software repositories. To address this issue, we adhered to a service-oriented approach together with a static rule set to define the connections. As described previously, all the operators and data extractors are designed as web service clients. The data extractors get data from REST end-
points of the various continuous integration tools in JSON format, which allows us to resolve the issues arising due to various data formats. However, the next challenge is to provide a mechanism for service composition to integrate the extracted data. However, traditional service composition is considered as a time consuming and error-prone process, which requires experts in both composition languages and existing standards. In this paper, we tackle the problem of service composition, by adhering to a light-weight, programming-free and process-centric technique. The composition language is hidden behind the graphical interface of the query platform, which exploits the RESTful nature of CI tools and provides a service composer to enable semi-automated integration of software quality related data. In summary, SA-FLY consists of a custom-made composition language in terms of a static grammar and an infrastructure for the service offerings. It is capable of accessing various software artifacts to extract the necessary data, perform the required processing steps, compose into workflows, and execute over the Internet.

The query is generated in the form of a workflow (i.e., by pipelining data extractors and operators), which accomplishes a particular information need of a software practitioner. For example, consider answering a question like “What is the number of commits of a developer in a project?”. To answer this question, the SA-FLY user needs to generate a workflow to represent the query. First, it is required to extract the commit information from a version control repository by specifying a particular project using a data extractor registered in SA-FLY followed by an operator to loop through all the commits. Then the commits of a particular developer need to be filtered out by using a filter operator. Finally, the number of commits of the developer needs to be calculated by using a count operator. Figure 4 presents a screen-shot of the generated workflow. An interactive video and a screen-shot showing how SA-FLY works can be found in public Github repository https://github.com/chaman-pub/SA-FLY.

In this approach, contrasting to fully-fledged workflow models, the queries can only be generated by adhering to three of the classic workflow patterns: Sequence, Parallel split and Synchronization. For composing queries, all data extractors and operators are defined directly on an underlying logical representation, a static grammar. However, the users do not need to know the underlying formalism because they are only exposed to the visual interface, which consists of the canvas, data extractors, and operators. The static grammar is an integral part of SA-FLY that has been defined manually. Static grammar consists of the production rules to combine data extractors and operators into a meaningful workflow. Ideally, the static grammar rules follow a syntax similar to the Extended Backus-Naur Form but, here we show it using a simplified syntax. Table 1 represents a
Table 2  Our approach to employ GQM to measure ISO/IEC 25010 quality characteristics

<table>
<thead>
<tr>
<th>Characteristic and Sub-characteristic</th>
<th>Goal</th>
<th>Question</th>
<th>Sub-question</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Suitability</td>
<td>Evaluate the degree of closeness of a measured or calculated quantity to its actual value</td>
<td>What is the degree of systematic and random errors in software?</td>
<td>What is the failure ratio between the releases?</td>
<td>No: of build failures</td>
</tr>
<tr>
<td>Functional Correctness</td>
<td>Evaluate the ability of software to maintain a specific level of performance in case of failures</td>
<td>How well can a system continue to operate in the event of a failure?</td>
<td>What are the builds that resulted in failures?</td>
<td>No: of build failures</td>
</tr>
<tr>
<td>Reliability</td>
<td>Evaluate the ability of software to maintain a specific level of performance in case of failures</td>
<td>What is the usability of a software in different environments?</td>
<td>Is the software thoroughly tested before releasing?</td>
<td>Code coverage</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>To measure the effectiveness and efficiency of a system in meeting the goals and expectations of its users</td>
<td>What is the effectiveness of a software in determining the success rate in test cases?</td>
<td>What are the error rates between failure builds?</td>
<td>No: of build failures</td>
</tr>
<tr>
<td>Usability</td>
<td>Maintain appropriate personal allocation throughout the project</td>
<td>How has a particular developer been utilized in a project?</td>
<td>What is the contribution of a developer to a project?</td>
<td>No: of commits of a developer to a project</td>
</tr>
<tr>
<td>Operability</td>
<td>Identify the attributes of software systems that bear on the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified.</td>
<td>What is the difficulty level in analyzing software?</td>
<td>How big is the source code?</td>
<td>Lines of code</td>
</tr>
<tr>
<td>Performance Efficiency</td>
<td>Maintain appropriate personal allocation throughout the project</td>
<td>How has a particular developer been utilized in a project?</td>
<td>What is the contribution of a developer to a project?</td>
<td>No: of commits of a developer to a project</td>
</tr>
<tr>
<td>Resource utilization</td>
<td>To characterize the effort required to install the software</td>
<td>What is the amount of effort needed to install a software in multiple environments?</td>
<td>What is the number of smallest installable parts available in a software?</td>
<td>No: of unit tests</td>
</tr>
<tr>
<td>Portability</td>
<td>Ability to prevent a harm/ unauthorized access, whether accidental or deliberate, to programs or data</td>
<td>How is a software resistant to, or protected from, harm or third party access?</td>
<td>Who is the responsible person to originate the harm?</td>
<td>The corresponding person for the build failure</td>
</tr>
<tr>
<td>Installability</td>
<td>The ability of different software to communicate and exchange data in an accurate, effective, and consistent manner</td>
<td>What is the ability of a system to work with other systems without special effort on the part of the developer?</td>
<td>What are the similar projects a particular developer is working on?</td>
<td>No of projects for a developer</td>
</tr>
</tbody>
</table>
part of the grammar file unique to VCS data extractors and operators. Similarly, repository specific grammar rules are defined to ensure the smooth composition of operators on multiple repositories.

In the Table 1 GET refers to a data extractor from a repository. FOREACH refers to an operator that can loop through any information extracted from a version control repository, bug repository, etc. FILTERDATE and FILTERCONDITION refer to two different filter operators. FILTERDATE is used to filter the data based on a date range, where FILTERCONDITION is used to filter the data based on any other specific condition such as committer name, status of the issue, or status of the build. Likewise the operator sequences are defined manually and be extended based on the introduction of new operators and new data sources.

The architecture mentioned above together with the data extraction and integration logic of SA-FLY distinguish our approach with the existing approaches by providing several notable advantages. For example, the queries are not pre-defined, unlike the previous approaches such as [15]. Also, the correctness of the queries are ensured with the help of static grammar rules. In contrast to the approach presented by [12], SA-FLY allows mining data from various repositories. Manual inspection of state-of-the-art tools such as Github, Bitbucket, Jenkins, Jira, and SonarQube is tedious. Moreover, programmatically querying the data in such tools are remarkably effort incentive process. The concept of data extractors and operators presented in SA-FLY addresses this issue. The experimental results presented in Sect. 5 showcase the above claim. The extractors and operators allow SA-FLY users to create queries without disturbing about the extraction or integration logic.

This approach ventures the RESTful nature of software repositories to extract the data. This allows to overcome the problem of interlinking heterogeneous data originated from multiple repositories. With the help of service clients defined in SA-FLY, all the data are extracted and processed in JSON format. In summary, the interactive query interface of the SA-FLY framework facilitates extracting data from heterogeneous software repositories through data extractors and aggregate the results by applying operators to perform real-time software analysis tasks. Static grammar rules define the logic for data integration. Therefore, with all the conceptual building blocks and the proof-of-concept SA-FLY implementation we can answer the first research question, RQ1 yes. As per the elements illustrated in this section, it is possible to provide an interactive query interface to carry out software analytics tasks by creating query workflows visually.

5. Evaluation

This paper intends to present a novel mechanism and a proof-of-concept framework for extracting information from various software artifacts and performing real-time in-depth analyses. First, the evaluation is carried out to measure the overall correctness of SA-FLY. Second, it was evaluated whether the results produced by SA-FLY is time efficient when compared with state-of-the-art continuous integration tools such as Jenkins, SonarQube as well as web-based coding platforms like Bitbucket and Github. Finally, the usability of SA-FLY compared with the same set of tools. Three hypotheses (see Table 3) have been formulated to assess the above claims, and in that way, we plan to answer RQ2: When software practitioners use such a framework to perform software analysis tasks, can we witness an improvement over the state-of-the-art in terms of accuracy, time efficiency, and usability?

5.1 User Study

The user study has been conducted based on the Between Subjects Design, which is one of the well-known experimental design approaches in Software Engineering [30]. The purpose of user study was to carry out a multi-fold evaluation of SA-FLY in terms of correctness when performing the tasks, the time to complete tasks and the overall usability of SA-FLY when compared with other stand-alone tools. The research population of this study consists of 31 subjects where all of them are part-time masters students in computer science, working in software industry. The inclusion criteria were mainly based on working experience in the software industry together with the familiarity of continuous integration tools such as GitHub, Bitbucket, Jira, SonarQube, and Jenkins. SA-FLY’s interactive and programming-free interface allows us to select the population without restricting on factors such as job profile and the experience in a particular programming language as long as they are familiar with above mentioned CI tools.

From the 31 participants, 16 were assigned to the experimental group, and the rest 15 were assigned to the control group. A purely random approach has used to assign the subjects to both experimental and control groups. The experimental group was provided with SA-FLY, whereas the

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Hypotheses</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Null Hypotheses</strong></td>
<td><strong>Alternative Hypotheses</strong></td>
</tr>
<tr>
<td>H1o The total correctness score for all tasks is the same across the experimental and control group.</td>
<td>H1 The total correctness score for all tasks is different across the experimental and control group.</td>
</tr>
<tr>
<td>H2o The System Usability Scores is the same across the experimental and control group.</td>
<td>H2 The System Usability Scores is different from the experimental and control group.</td>
</tr>
<tr>
<td>H3o The time spent on solving all tasks is the same across the experimental and control group.</td>
<td>H3 The time spent on solving all tasks are different across the experimental and control group.</td>
</tr>
</tbody>
</table>
control group was supposed to carry out the tasks with aforementioned state-of-the-art tools. Figure 5 depicts the industry experience and the familiarity with CI tools in both experimental and control groups. Mean values for the industry experience and the experience with the tools of the both experimental and the control groups are statistically not significantly different.

User study started with a brief introduction to outline the objectives of the user study. Then we demonstrated how SA-FLY works by selecting a software analysis task (outside the seven tasks chosen for the user study) and the same task has demonstrated by using stand-alone tools. By doing that, we were able to communicate the main objective of the user study to the participants in both experimental and control groups. The task list has distributed in the form of a questionnaire, where participants were expected to write down the answer upon completion of each task. Knowing the nature of some tasks, the duration to solve a task has limited to 15 minutes.

To evaluate the usability of SA-FLY, it was decided to use System Usability Scale (SUS), which was introduced by Brooke [31]. SUS perfectly fits our requirements as it is technology independent and has tested on hardware, consumer software, and websites. Therefore, in addition to performing the tasks, we asked our subjects to record their immediate response. SUS is based on a ten-item questionnaire, and it was instructed to answer based on the experience with SA-FLY instead of the state-of-the-art tools. By doing that, we were able to communicate the main objective of the user study to the participants in both experimental and control groups. The task list has distributed in the form of a questionnaire, where participants were expected to write down the answer upon completion of each task.

The experimental group was instructed to provide answers to the questions as per the experience with SA-FLY while performing the tasks. The control group was instructed to answer based on the experience with the state-of-the-art toolset that they used to solve the same tasks. Then a measurement of the usability has been calculated for each participant for SA-FLY and the state-of-the-art tools separately.

5.2 Selection of Questions

The question list and the scenarios presented in the literature is quite comprehensive and highlights several critical aspects in the field of software engineering. Therefore, in this research, we used such questions and scenarios as a frame of reference. Our query interface is capable of overcom-

5.3 Selection of Study Objects

The rationale behind the selection of study objects is that the study subjects should be able to perform the tasks by using industry-scale projects to strengthen the external validity of our study. Three open-source Apache projects available in Github is used for the experimentation. Apache projects have been used in many case studies by several researchers over the last decade or so. The selection of projects is based on the availability of such projects in the public instances of Sonarqube and Jenkins to be able to perform all the tasks. On the other hand, we ensured that the selected projects are of a reasonable size due to the time limitation in conducting experiments. General description of the selected projects and their sizes are described in the next section.

Scenario 1: To observe the continuous code quality throughout a project [11]

More than a few studies confirmed that source code metrics are helpful to steer the software development lifecycle. For instance, metrics could be used to review its overall quality or predict defects. The current practice is that the VCSs only keep track of the source code. It means that to calculate metrics, the entire source code needs to be fetched from the repository and parsed or even partially compiled [11]. For example, features like test coverage, test success, code clones, and complexity are critical in observ-
ing the continuous code quality of a project. Therefore, obtaining answers to questions Q1, Q2 and Q3 would provide insights related to software code quality for software practitioners.

Q1: What is the percentage of unit test coverage?
Q2: What is the percentage of unit test success?
Q3: What is the percentage of code duplications?

Tools like SonarQube can be used to observe the continuous code quality of a software system. However, to find answers to the questions mentioned above, a particular user needs to create a SonarQube project and point out the location of the project. The location could be either a Github link or a local path. Then the answers can be found on the SonarQube dashboard. In contrast, the visual interface of SA-FLY permits to generate the above questions in the form of workflows and obtain the answers on-the-fly. However, our approach does not aim at developing tools to calculate the metrics, rather it extracts the results produced by other tools to answer these basic questions. In answering the questions in Scenario 1, SA-FLY also fetches data from SonarQube. Therefore, the experimental results obtained from both experimental group and control group are not significantly different in terms of correctness and the time. However, the critical advantage of SA-FLY is that it can extend to access various quality assurance platforms to obtain numerous other valuable information on software projects rather than depending only on SonarQube. In that way, users can witness a holistic view of software systems. The experiments were conducted on apache commons lang project†, which is a package of Java utility classes for the classes that are in java.lang’s hierarchy, consisting of approximately 28K lines of code.

Scenario 2: To comprehend the contribution styles of a software project[32]

Assessment of the contribution of individuals that work on a software project has identified as an important aspect of MSR research[33], [34]. Appraisals on contribution are carried out to observe the project development rate as well as to realize the bottlenecks of implementation. Such observations can facilitate project planning and future estimations[32]. To effectively analyze this scenario, three key questions have been identified (Q4 to Q6).

Q4: What is the number of commits of a developer in a project?
Q5: What is the number of commits between date A and date B?
Q6: What are the successful builds of a project?

Experiments were conducted on Apache Gora project†††, which is available as a GitHub project as well††††.

Apache Gora is an open source framework consisting of approximately 21K lines of code that presents an in-memory data model and persistence for big data. Answering Q4 without SA-FLY is cumbersome, requiring accessing Git log programmatically (or manually) and filter commits of each developer. Similarly, Q5 required extracting commit log and distill the data according to a date range. Without using SA-FLY, Q6 requires programmatically accessing build information from a continuous integration tool and filtering based on the successful build status. A build is considered to be successful when the compilation reported no errors.

Scenario 3: To infer fine-grained relationships across multiple software artifacts[3], [35]

Benefits of using MSR techniques span for several noteworthy dimensions such as understanding software systems, propagating changes, predicting bugs, understanding team dynamics, improving user experience, reusing code and automating empirical studies. To perform a better analysis on such dimensions, discovering fine-grained relationships among various software artifacts is required. Q7 is a typical example of that nature.

Q7: What is the number of commits between two successful builds?

Experiments were conducted on Apache Falcon†††† project, which is a Java based feed processing and feed management system consisting of approximately 70K lines of code. A notable advantage of SA-FLY can be witnessed when answering Q7 with and without SA-FLY. It was clearly showcased that accessing multiple repositories, filtering data and finding relationships is awkward when done manually. Neither it is a straightforward task to do it programmatically. In that case, one can effectively utilize SA-FLY to access multiple repositories and filter data based on the requirements. Another noteworthy advantage of SA-FLY is that the ability to change the parameters (i.e., data sources, monthly builds, weekly builds, commits, issues) without worrying about the implementation logic. Hence, various types of information can be gathered on several dimensions in less time.

5.4 Data Collection and Analysis

In this section, we overview the data collected to evaluate our three hypotheses and the empirical results obtained through the statistical analysis.

5.4.1 Overall Correctness of the Tasks

A simple rating mechanism has been used to obtain the correctness values for each task. If the answer to a particular

†https://commons.apache.org/proper/commons-lang/
††http://gora.apache.org/
†††https://github.com/apache/gora
††††https://falcon.apache.org/
task is correct (i.e., the perfect match of the obtained values) the participant was given two points. Likewise, for the mentioned seven tasks a maximum score of 14 points could obtain if all of them were answered correctly. Similarly, 0 marks allocated for the wrong answers. However, incomplete solutions were handled differently. In the experimental group, the answers will be obtained based on the analysis workflow designed by the subject. If the workflow is correct, it gives the correct results, and in case of incorrect workflows, SA-FLY gives wrong answers. There can be situations where the subjects may find it challenging to design the workflow completely. In such circumstances, we decided to allocate one mark if there are at least two correct workflow items in the workflow. As an example, if a particular task requires five workflow items to be combined to obtain the correct answer, the subject will get one mark if two or more workflow items are correctly combined in the workflow. In the control group, there are cases where the manual calculation of commits and issues are error-prone. For example, manual calculation of “What is the number of commits between date A and date B?” using Github could be error-prone. In such cases, if the answers are in an acceptable range, we allocate one mark considering it as a partially correct answer.

SA-FLY’s mean correctness score is 10.8 (standard deviation of 1.42) compared with the mean correctness score of 6.7 in control group (standard deviation of 1.53). Box plot is shown in Fig. 6 for the overall correctness for the both experimental and control groups. As per the box plot, the 25th percentile of the experimental group is above the 75th percentile of the control group, denoting considerable overall correctness of SA-FLY over state-of-the-art tools.

We further used the non-parametric Independent Samples Mann-Whitney U (MWU) test with a significance level 0.01 to test H1o. The MWU test rejects H1o at the 99 percent confidence level (with p < 0.0001). We, therefore, accept H1: total scores are different for the experimental and control groups.

5.4.2 Overall Usability Score of the Tasks

We utilized SUS [31] 10 questions to evaluate the usability of SA-FLY compared with the state-of-the-art tools in performing the selected seven tasks as described in Sect. 4.2. As mentioned above, the subjects were requested to give a score from 1 to 5 for each of the ten questions, based on their degree of satisfaction. As per the standard SUS scoring mechanism, for each odd-numbered questions, 1 point has been subtracted from the score. Then for each even-numbered questions, the score has been subtracted from 5. The new values have been added up to obtain the total score. Finally, the total score is multiplied by 2.5 to obtain the final SUS score. Though the final SUS score is not a percentage value, it gives the score out of 100 and hence, it is an unambiguous way to compare the results.

SA-FLY’s mean SUS score is 72.9 (standard deviation of 12.2), and it is 22.7 higher than the that of control group’s score of 50.2 (standard deviation of 13.2). Given that the 25th percentile of the experimental group is well above the 75th percentile of the control group according to the box plots in Fig. 6, it evidently showed the acceptance of SA-FLY over baseline tools used for the evaluation. Furthermore, The MWU test rejects H3o at the 95 percent confidence level (with p < 0.0001), allowing us to accept H3: the SUS scores are different for the experimental and control groups.

5.4.3 Overall Completion Time of the Tasks

Knowing the nature of some questions, we specifically mentioned study subjects to not to spend more than 15 minutes on a single task. At the end of each task, they were requested to write down the time spent on each task. Average completion times for all the questions across the experimental and control groups presents in Table 4. It was observed that overall completion time of the experimental group is less than that of the control group. A notable advantage of SA-FLY was not witnessed in answering Q2 and Q3, in particular. However, SA-FLY significantly surpasses the state-of-the-art tools in answering Q5, Q6, and Q7.

The box plots in Fig. 7(a) shows the distribution of time across the experimental group and the control group, denoting that SA-FLY was capable of obtaining the results much faster than that of baseline tools. The MWU test rejects the null-hypothesis H3o at the 99 percent confidence level. The acceptance of the alternative hypothesis confirms that the distribution of the total completion times among the two groups is different.

Research question (RQ2) can be answered by accepting hypotheses H1, H2, and H3. It was observed that SA-

![Fig. 6](image_url)  Total correctness scores (left) and total usability scores (right) for experimental and control groups
5.5 On the Threats to Validity

**External validity:** Threats to external validity are concerned with the generalizability of the results. First, the selected scenarios and tasks in our study would favor SA-FLY, and therefore the results might not be generalized. However, we have minimized the threat by selecting scenarios from the literature, which have been proven important. Second, the selection of study subjects may not be representing software practitioners in the industry. As per the inclusion criteria of the subject selection in our research, it was mainly based on working experience in the software industry together with the familiarity of continuous integration tools. However, we have not specifically looked for the experience in a particular programming language as long as the subjects are familiar with above mentioned state-of-the-art tools.

**Internal validity** Uneven distribution of experience level across experimental and control group might bias the results of the user study. However, to minimize the risk, the assignment to the groups have been done in a purely random manner. Both groups have used stopwatch application on their mobile phones to record the completion time of each tasks. Therefore, there is a possibility of recording incorrect times by the participants. This can be considered as a minor threat to validity. Furthermore, finding answers to questions 4, 5, 6 and seven are subjective for the selected study objects. Therefore, knowing the nature of study objects, the time to perform the tasks have been limited to a maximum of fifteen minutes. This issue must be resolved by repeating the same tasks for two different study objects selected based on the size of the project.

**Conclusion validity** Being the total number of participants used in the study is relatively low, it might influence the results and hence, a threat to conclusion validity. However, to minimize the error, we employed non-parametric MWU test, which is considered to be accurate even with small sample sizes.

5.6 Discussion and Synthesis

Empirical results of the user study evidently demonstrate that the experimental group was managed to solve the tasks with a higher correctness level in a lower time duration by using SA-FLY. One of the key benefits of this approach is that it is capable of dealing with multiple repositories to answer complex questions. A significant difference regarding accuracy and the time efficiency is observed particularly for analysis questions such as Q6 and Q7 when compared with state-of-the-art tools. However, we do not claim that SA-FLY can fully replace the state-of-the-art tools in every analysis task. For example, finding answers for Q2 and Q3 with and without SA-FLY has not shown a significant difference regarding accuracy and the time efficiency. Figure 7 (b) shows the time difference between Experiment (mean = 1.4 and standard deviation = 0.49) and Control (mean = 2.5 and standard deviation = 0.62) groups when answering Q2 and Q3. Besides, this approach has several noteworthy benefits compared to the state-of-the-art research in the same direction.

**Multiple stakeholder support:** With the evolution of software systems, different stakeholders such as software developers, testers, and project managers continuously seek various information to perform their daily activities. Most of the existing research mainly focused only on facilitating software developers with their information needs. The main reason is that with the technical and programming knowledge the developers have, they can effectively utilize any framework or tool to get the information they need. In contrast, the programming-free interface of SA-FLY allows any non-technical stakeholder to find various information about the software system. For example, software developers can find information about code quality, whereas project managers can find information about the performances of software developers. Therefore, SA-FLY is capable of fulfilling the information needs of multiple stakeholders such as developers, testers, project managers and so on. Further, it would enhance and speed up the work of software practitioners by giving them access to a significant amount of information without having to install several tools and to cope with many output formats.

**Dynamic query generation:** Existing approaches are typically static and only capable of fulfilling the information needs for a set of pre-defined queries. In contrast, our approach does not restrict software practitioners to engage only with a set of predefined queries. Instead, it allows users to create queries and execute them automatically dynamically.

**Data integration across multiple artifacts:** Integration of data originated from numerous software artifacts would thrust software analysis research into a higher level. The evaluation scenarios of this research showcased that the integration can reveal some useful insights of software projects, which are not readily apparent.

**Programming-free composition:** Any stakeholder
having a conceptual knowledge in the software analysis domain can use our mashup platform to create queries without having prior knowledge of the data integration logic. The integration logic is hidden behind the graphical query interface, where the stakeholders only need to drag and drop the components onto the canvas and combine them to generate the queries.

Replication support: Our query interface provides means to replicate previous software analysis experiments based on the availability of the ground data. Further, both expert and novice researchers can perform various analysis experiments on the SA-FLY framework.

6. Conclusion

This research presents a novel mechanism and a proof-of-concept visual query interface to dynamically integrate the data originated from heterogeneous software artifacts to fulfill the complex and dynamic information needs of software practitioners and researchers. Such information is vitally important for them to infer various aspects of software quality. This approach has rationalized by employing GQM method, which allows quantifying the goals by exploiting the fine-grained information needs meaningfully. A user study was conducted with 31 participants to evaluate SA-FLY in terms of accuracy, time efficiency and usability compared with state-of-the-art tools. Evaluation results showcased the capability of SA-FLY in fulfilling the information needs. We believe that many significant challenges and issues are yet to be addressed and resolved. However, we hope our study serves as a stepping-stone for better software analysis that guides software practitioners towards developing high-quality software.

For results interpretation, we plan to integrate a visualization plug-in to SA-FLY, which is currently not investigated in this research. That will drive this research to its full potential by motivating other researchers to use our platform in performing various software analysis experiments. Providing continuous feedback during the query workflow generation is another exciting future direction, where it minimizes the integration burden as well as the possible human errors when creating workflows.

Acknowledgements

The authors of this paper gratefully acknowledge the financial support provided by the National Research Council of Sri Lanka (Grant No: NRC 15-74). Further, we are thankful to the colleagues at SEAL Lab at University of Zurich for providing valuable insights for this research.

References


Chaman Wijesiriwardana received his B.Sc. (Hons) Special degree in Computer Science from the University of Peradeniya, Sri Lanka and obtained his M.Sc in Information and Communications Technology degree from Asian Institute of Technology, Thailand. He worked as a research assistant in Software Evolution and Architecture Lab at University of Zurich for 3 years. His research interests include software evolution analysis, mining software repositories and software security.

Prasad Wimalaratne obtained his B.Sc special degree in Computer Science from the University of Colombo and his PhD in Virtual Environments from the University of Salford, United Kingdom in 2002. He is a senior member of IEEE and a member of Computer Society of Sri Lanka (CSSL). His research interests include Virtual/Augmented reality Secure Software Systems and Assistive Technology. He joined the academic staff of the University of Colombo in 1995 and is currently the head of the department of Communication and Media Technologies at University of Colombo School of Computing. He served as a council member of the Sri Lanka Association for the Software Industry (SLASI), vice chair of Institute of Electrical and Electronic Engineers (IEEE) SL Section and as an executive council member of the Computer Society of Sri Lanka (CSSL).