साङ्गणिकम् अन्वयचित्रणम्

☛ चैतन्यः सु लक्कुण्डि
Email- chaitanya.lakkundi@gmail.com

पद्यानाम् अर्थावबोधे अन्वयः महते उपकाराय कल्पते । प्रायः अन्वयः द्विधा भवितुम् अर्हति । दण्डान्वयः आकाङ्क्षान्वयः चेति । तृतीयोऽपि कश्चन चित्राधारितः क्रमः भवितुम् अर्हति । कोशात् ज्ञायते यत् अन्वयः इति पदस्य वंशः, कुलं, वंशपरम्परा, परस्परसम्बन्धः इत्यादयः अर्थाः भवन्ति इति । पद्यप्रसङ्गे प्रयुक्तस्य अन्वयशब्दस्य पदानां परस्परसम्बन्धः इत्यर्थः स्वीकार्यः । दण्डान्वये पदानां क्रमः परिवर्त्य उपस्थाप्यते । तथा च आकाङ्क्षान्वये क्रियापदं निश्चित्य तत्क्रियायाः कर्ता कर्म साधनं विशेषणम् इत्यादीनि ऊह्यन्ते । तृतीये च चित्रान्वये पदानां परस्परसम्बन्धः चित्ररूपेण द्योत्यते । आकाङ्क्षान्वयस्य संवर्धितं रूपम् इदम् । अधस्तनश्लोकस्य त्रिप्रकारकम् अन्वयं पश्याम ।

प्रथमम् उदाहरणम् –

महाकविः कालिदासः रघुवंशमहाकाव्ये मङ्गलम् आचरयति अनेन श्लोकेन ।

वागर्थाविव संपृक्तौ वागर्थप्रतिपत्तये ।
जगतः पितरौ वन्दे पार्वतीपरमेश्वरौ ।।

दण्डान्वयः –

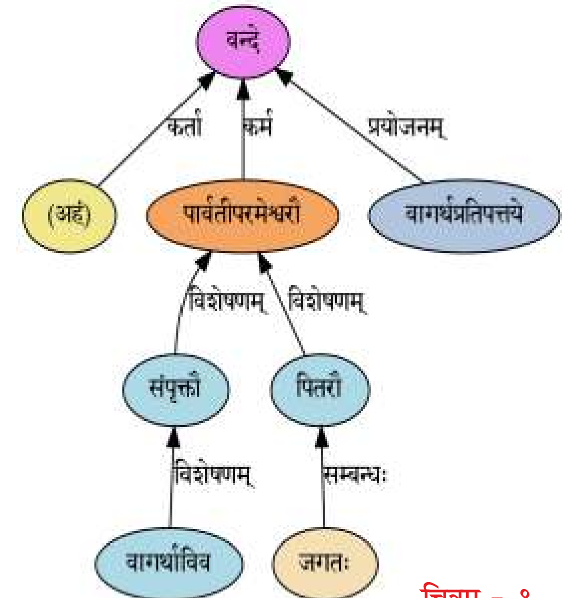
(अहं) वागर्थाविव सम्पृक्तौ जगतः पितरौ पार्वतीपरमेश्वरौ वागर्थप्रतिपत्तये वन्दे ।

आकाङ्क्षान्वयः

वन्दे – अभिवादये
कः ? – अहम्
कौ वन्दे ? – पार्वतीपरमेश्वरौ वन्दे
तौ कीदृशौ ? – पितरौ
कस्य पितरौ ? – जगतः पितरौ
पुनः कीदृशौ ? – संपृक्तौ
कौ इव संपृक्तौ ? – वागर्थाविव संपृक्तौ

चित्रान्वयः -

चित्रेऽस्मिन् बाणाः वर्तुलानि च दृश्यन्ते । वर्तुलेषु



चित्रम् – १

श्लोके विद्यमानानि पदानि बाणचिह्नेन सम्बन्धनामानि च निर्दिष्टानि । एतस्य पठनम् अधस्तात् उपरि करणीयम् । ततः चत्वारि पृथग्वाक्यानि सिद्ध्यन्ति ।

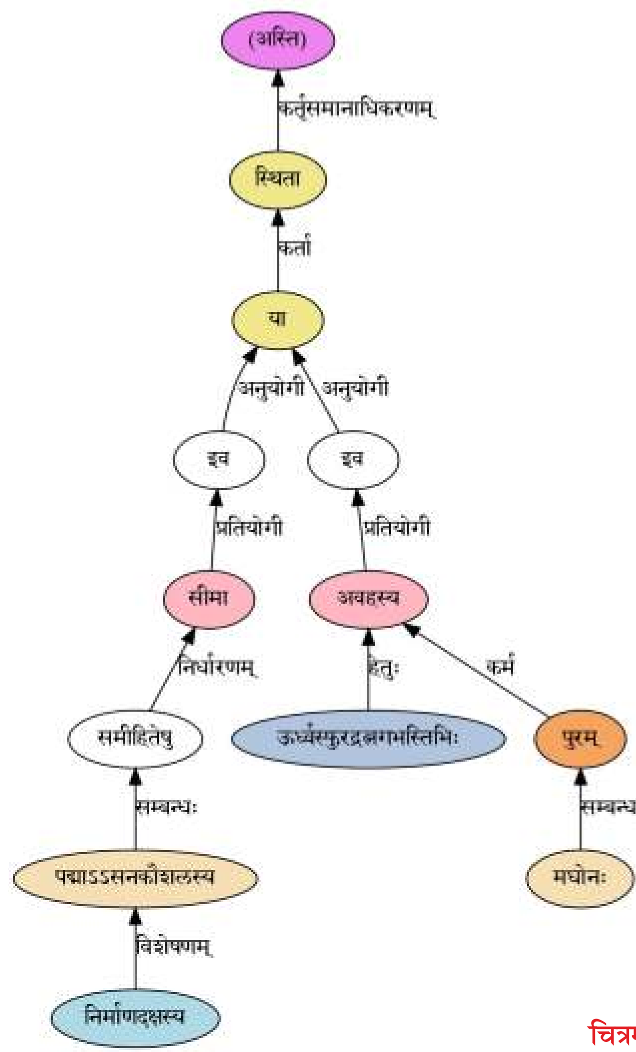
१. अहं वन्दे ।
२. वागर्थाविव सम्पृक्तौ पार्वतीपरमेश्वरौ वन्दे ।
३. जगतः पितरौ पार्वतीपरमेश्वरौ वन्दे ।
४. वागर्थप्रतिपत्तये वन्दे ।

अहं वन्दे इत्येतयोर्मध्ये कर्ता इति लिखितं चित्रे । अर्थात् अहम् इति पदं वन्दनक्रियायां कर्तृत्वसम्बन्धेन

अन्वेति । तथैव पार्वतीपरमेश्वरौ इति पदं क्रियायां कर्मत्वसम्बन्धेन अन्वेति । वागर्थ-प्रतिपत्तये इति प्रयोजनम् । जगतः इति पित्रोः षष्ठीसम्बन्धः । अन्यानि विशेषणानि ।

क्रियापदेन सह येषां पर्वणां साक्षात् सम्बन्धो वर्तते तेषां पङ्क्तिशः अपि पठनं शक्यते । यथा अहं पार्वतीपरमेश्वरौ वागर्थप्रतिपत्तये वन्दे । एवमपि अर्थपूर्णं वाक्यं सिद्ध्यति । अत्र वयं दृष्टवन्तः यत् एकस्यैव वाक्यस्य बहुप्रकारकं पठनं सेत्स्यति । लघुना उपायेन बृहद्वाक्यस्य अर्थः अवगम्यते ।

अस्मिन् उदाहरणे स्पष्टप्रतिपत्त्यर्थं प्रसिद्धश्लोकः स्वीकृतः आसीत् । इदानीं प्रायः अज्ञातं नूतनमेकं पद्यं स्वीकृत्य अन्वयचित्रणं पश्याम ।



चित्रम् - २

द्वितीयम् उदाहरणम् -

महाकविना भट्टिना विरचितम् अद्वितीयं काव्यं भट्टिकाव्यमिति मन्यते । तत् रावणवधम् इति अपरनाम्नापि ख्यातम् । अत्र भट्टिः विरलप्रयोगान् दर्शयति । इतः स्वीकृतम् अग्रिमम् उदाहरणम् -

इतः प्राक् भट्टिः अयोध्यायाः विवरणं करोति । तत्सम्बद्धः अस्ति अयं श्लोकः ।

निर्माणदक्षस्य समीहितेषु
सीमेव पद्मासनकौशलस्य ।
ऊर्ध्वस्फुरद्रत्नगभस्तिभिर्या
स्थितावहस्येव पुरं मघोनः ।। - (१.६)

दण्डान्वयः -

निर्माणदक्षस्य पद्मासनकौशलस्य समीहितेषु सीमा इव या ऊर्ध्वस्फुरद्रत्नगभस्तिभिः मघोनः पुरम् अवहस्य इव स्थिता (अस्ति, ताम् अध्यास्त) ।

अत्र साक्षात् चित्रेण तात्पर्यम् अवगन्तुं प्रयतामहे । तत्रादौ पठनक्रमः परिलक्ष्यते ।

१. या स्थिता अस्ति । (पूर्वतनश्लोकात् यत्शब्दः अयोध्यायाः द्योतकः इति ज्ञेयम्)
२. सीमा इव या स्थिता अस्ति ।
३. अवहस्य इव या स्थिता अस्ति ।
४. निर्माणदक्षस्य पद्मासनकौशलस्य समीहितेषु सीमा इव ।
५. मघोनः पुरम् अवहस्य इव ।
६. ऊर्ध्वस्फुरद्रत्नगभस्तिभिः मघोनः पुरम् अवहस्य इव ।

प्रथमवाक्यात् ज्ञायते यत् अयोध्यायाः एव विवरणम् अत्रापि क्रियते । द्वितीय-तृतीय-वाक्याभ्याम् अवबुद्ध्यते

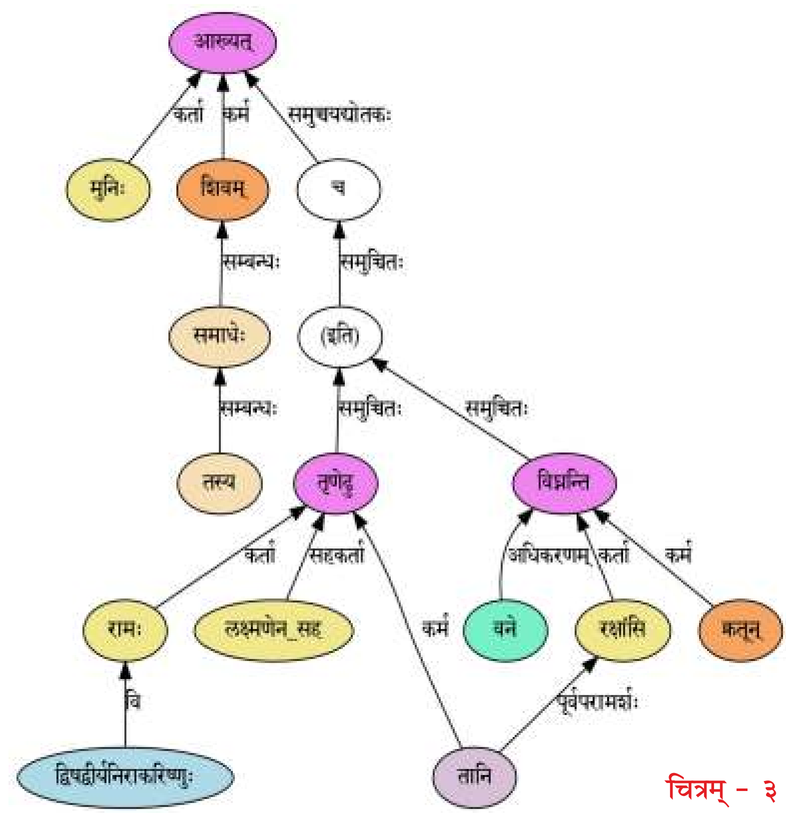
यत् अयोध्यायाः तोलनं क्रियते अत्र इति । इव इति शब्दोऽपि एतस्य सूचकः । वस्तुतः अत्र उत्प्रेक्षालङ्कारः प्रयुक्तः । उत्प्रेक्षालङ्कारस्य विवरणमत्र न क्रियते । परन्तु तस्य प्रसिद्धम् उदाहरणं भवति कुमारसम्भवस्य मङ्गलाचरणम् – 'स्थितः पृथिव्या इव मानदण्डः' इति । हिमालयः पृथिव्याः मानदण्डः इव स्थितः । आङ्ग्ल-भाषया वक्तव्यं चेत् as if इति अर्थः इवपदस्य । The Himalayas are, as if a measuring scale used to measure the earth.

प्रकृतपद्येऽपि सीमा इव स्थिता, अवहस्य इव स्थिता इति वाक्यद्वयेन उत्प्रेक्षा परिलक्षिता ।

(तत्र प्रतियोगी इति दण्डस्योपरि लिखितम् । यस्मिन् सादृश्यं सः अनुयोगी । यस्य सादृश्यम् उच्यते सः प्रतियोगी इति उच्यते । चन्द्रसदृशं मुखमित्यत्र चन्द्रः प्रतियोगी, मुखम् अनुयोगि ।)

अत्र अयोध्या अनुयोगिनी, सीमा प्रतियोगिनी । (चित्रे सामान्येन प्रतियोगी इति पुंलिङ्गशब्दः प्रयुक्तः ।) कस्य सीमा इति पृष्टे चित्रदर्शनात् ज्ञायते यत् निर्माणदक्षस्य पद्मासनकौशलस्य (ब्रह्मणः) समीहितेषु (रचितपदार्थेषु) सीमा । तेषु समीहितेषु सीमा इव इयम् अयोध्यानगरी स्थिता अस्ति । अर्थात् सर्वेभ्योऽपि उत्कृष्टम् अस्तीति भावः । द्वितीयं तोलनं मघोनः (इन्द्रस्य) पुरेण साकं कृतम् । इन्द्रपुरी अमरावती अतिसुन्दरी इति प्रसिद्धमेव । इयम् अयोध्या मघोनः पुरम् अवहस्येव स्थिता अस्ति । तन्नाम अयोध्यायाः पुरतः इन्द्रपुरी अपि उपहासपात्रतां याति इति अवगम्यते ।

इदानीं सर्वेषाम् अंशानाम् अवगमनं सम्पन्नम् ऊर्ध्व-स्फुरद्रत्नगभस्तिभिः इत्येकस्मात् ऋते । रत्नगभस्तयः नाम रत्नकिरणाः उपरि रत्नकिरणाः प्रसृताः एव इन्द्रपुर्याः अवहासस्य कारणम् (हेतुः) इति आशयः ।



चित्रम् - ३

एवं हि श्रूयते यदेकं चित्रं सहस्त्रशः शब्दानां तुल्यम् इति । इत्थं पद्यस्य तात्पर्यं चित्रस्य दर्शनात् आत्म-साद्भवति ।

उपरिदत्तं चित्रम् अवलोक्य स्वयमेव अनुमातुं यत-न्ताम् । सङ्क्षेपेण अत्र विव्रियते । भट्टिकाव्ये पूर्वं विवृते राजा दशरथः मुनेः विश्वामित्रस्य कुशलम् अपृच्छत् । प्रकृते मुनिः दशरथं प्रति वदति । श्लोकस्तु इत्थम् –

आख्यन्मुनिस्तस्य शिवं समाधे-
र्विघ्नन्ति रक्षांसि वने क्रतूंश्च ।
तानि द्विषद्वीर्यनिराकरिष्णु-
स्तृणेढु रामः सह लक्ष्मणेन ।। – (१.१९)

श्लोकस्यास्य दर्शनात् आदौ वर्णसाम्येन ज्ञायते यत् अत्र त्रीणि क्रियापदानि सन्ति इति । तानि च आख्यत् (उक्तवान्), विघ्नन्ति (रोधयन्ति), तृणेढु (मारयतु) इति ।

१. मुनिः शिवम् आख्यत् ।
२. रक्षांसि क्रतून् विघ्नन्ति ।

३ . रामः लक्ष्मणेन सह तृणेढु ।

इत्येतैः त्रिभिः वाक्यैः तात्पर्यमिदं सङ्गच्छते ।

१ . मुनिः कुशलः अस्ति । (तस्य समाधेः शिवम् आख्यत्)

२ . किन्तु राक्षसानां कश्चन अवरोधः वर्तते । (रक्षांसि क्रतून् विघ्नन्ति)

३ . रामलक्ष्मणौ तान् राक्षसान् मारयताम् । (रामः लक्ष्मणेन सह तृणेढु)

द्विषद्वीर्यनिराकरिष्णुः इति रामस्य विशेषणम् । द्विट् नाम शत्रुः, वैरी । वैरिविक्रमनिराकरणशीलः इत्यर्थः तत्पदस्य ।

उपसंहारः

दण्डान्वयस्य आकाङ्क्षान्वयस्य च पूरकरूपेण इमं चित्रान्वयक्रमम् आश्रित्य पठिते सति अर्थावगमः झटिति भवति । पदानां परस्परसम्बन्धः क्रियया सह सम्बन्धः च साक्षात् अवगम्यते । छात्राः शिक्षकाः च समानरीत्या प्रयोक्तुं प्रभवन्ति । विविधशोधैः ज्ञातमस्ति यत् चित्ररूपि ज्ञानं स्मरणोपयोगि इति । अतः पाठ्यक्रमे अस्य संयोजनेन छात्राणाम् उपकारः महान् भविष्यति । भवन्तः दत्तसङ्केतं प्रविष्य नूतनानि चित्राणि अपि रचयितुं शक्नुवन्ति । इति दिक् ।

जालपुटस्य सङ्केतौ

अत्र भट्टिकाव्यस्य प्रथमसर्गस्य अन्वयचित्राणि भवन्तः द्रष्टुम् अर्हन्ति । चित्रान्वयरचनायाः तन्त्रांशः द्वितीयसङ्केते प्राप्यते । तत्र संप्राप्य नूतनानि चित्राणि रच्यन्ताम् ।

* भट्टिकाव्यस्य अन्वयचित्रणम् – https:// sambhasha. ksu.ac.in/Comp Ling/bhattikavya-nandini/
* चित्रं रचयितुं तन्त्रांशः – https://sambhasha.ksu.ac.in /anvaya´ chitranam

PointerViz - Towards Visualizing Pointers for Novice Programmers

Akhila Sri Manasa Venigalla, Chaitanya S. Lakkundi*, Sridhar Chimalakonda
Intelligent Software & Human Analytics (ISHA) Research Lab
Dept. of Computer Science and Engineering
Indian Institute of Technology
Tirupati, India
{cs18m017, cs18s502*, ch}@iittp.ac.in

Abstract

Pointers are considered as one of the key concepts in learning programming and are extensively used for implementing several data structures. They lay the foundation for handling dynamic aspects of a program, increase execution speed and handle data types with more efficiency. This makes it critical for budding programmers to be well versed with using pointers. However, most of the novice programmers find it difficult and tricky to understand concepts such as address allocations, pointers referring pointers and data structures containing pointers. Hence, drawing the physical structure and flow of pointers is considered to be a common learning practice to gain better clarity and avoid confusion when learning pointers. But, it is time consuming and tedious to draw the flow of pointers on paper while programming. To help programmers understand these variations in pointers, we propose ***PointerViz*** *as a Google Chrome extension that displays the pictorial representation of selected code with pointers. We conducted a preliminary survey with 40 students from various universities and 83% of the users reported positive experience with the plugin.*

1. Introduction

Good programming skills require sound knowledge of data structures. Learners of programming languages face various difficulties in terms of understanding various functions, attributes and data structures [1]. Several tools have been developed to help novices learn programming [2,3,4,5,6]. They include games that help students learn computer programming [7], environments that support conventional programming instructions like *Mindstorm* [4], *Scratch* [5], *Blockly* [6], *Snap!* [8] and many other code visualization tools such as *Examplore* [9] and *Python tutor* [3].

*This author has discontinued from the institute.

execution speed, handling complex data structures with more efficiency and ease. Pointers allow sharing without copying through pass by reference, which is advantageous when programmers desire to pass around big arrays. They also enable programmers to resize data structures whenever required, supporting dynamic memory allocation. While security seems to be a concern when pointers are used by novice programmers, pointers provide a greater advantage in terms of performance by speeding up program execution [11]. Performance being a critical aspect of programming, trade-off between security and performance can be considered useful. A study conducted by Lahtinen et al., states that most of the students face difficulties in understanding pointers and references [12]. This reveals that though pointers are the basic concepts of programming, they are still difficult to understand.

Visualization is one of the ways that can help learners to gain a better of data structures. Researchers have proposed various forms of code visualizations to improve learning of novice programmers since a couple of decades [13,3]. Continuous improvements are being made to develop techniques and tools that can better support visualization. *Online Python Tutor* proposed by Philip Guo is one such tool that visualizes code written after compilation [3]. Visualizing code snippets written by programmers helps in better program comprehension. Visualization can also aid programmers who are well aware of data structures and their implementation in helping them with trade-offs of using pointers. Learners of programming are generally tested on the address references and pointers to assess their knowledge of data structures [12], making it necessary for even novice learners to understand these concepts.

A common practice of learners is to draw down the flow of pointers' data and references with respect to memory allocations to better understand the code. But this method demands time, effort and sometimes may also be incorrect [14]. Hence, there is a strong need to

URI: https://hdl.handle.net/10125/63754
978-0-9981331-3-3

provide learners with technology that visualizes pointers based on the program or pseudo code they write, which is the main motivation for our work.

Though there are many existing visualization tools that help in visualizing code snippets, there is not much work done to visualize pointers and their references before compilation, to the best of our knowledge. Hence we propose *PointerViz*[1] to support learners of programming language to comprehend pointers better.

The remainder of this paper is structured as follows. Section 2 discusses the related work followed by Section 3, which focuses on design methodology and development of *PointerViz*. Working of *PointerViz* is described in Section 4. Section 5 presents user scenario in the form 4 cases. We present the evaluation and user survey results in Section 6 and Section 7. Finally, we discuss the limitations in Section 8 and end the paper with conclusions and future directions in the Section 9.

2. Related Work

Researchers have developed various visualization tools to help novice programmers learn programming quicker have been developed. Scratch [5] is a block-based visualization tool that helps users to program easier by supporting drag and drop of blocks than writing the program. *Blockly* is also a block-based tool that provides visualization of programs [6]. Coding concepts are represented as interlocked blocks and *Blockly* generates syntactically correct code in programming language of our choice [6]. Tools like *Examplore* provide visualization of API usage examples to help users understand various correct ways of using APIs in programming languages [9]. *Progranimate*, an e-learning web based tool assists users in programming using flow chart representations. It provides code generation, inspects variables and thus provides syntax and semantic learning of programmers to the users. This helps users to gain an in-depth sound knowledge of the programming language [15]. *NetBlox* is another visualization tool that has been developed to enhance understanding of distributed programming [16]. In this tool, messages that are communicated among systems are represented as blocks with message payloads. Programmers are provisioned to provide Message Type that defines data present in the message.

Games is another direction of research that has been leveraged to make programming interesting and easy [17,18,19,20].One of the games developed by Leutenegger et al. teaches fundamental programming concepts in C++language, with the help of 2D game development [18].Robot ON! is a game developed to improve program Comprehension among learners. It helps players in understanding of control flow, program statements, data types and function calls by allowing players to demonstrate their understanding of the above in a given program [19]. *RoBUG* game has been developed to support and motivate players in learning of effective debugging. It comprises of four levels that require player to do certain tasks in each of them, like code tracing, using print statements to identify bugs, use divide-and-conquer strategy to spot the bugs and using breakpoints to analyze variable values [20].

Extensions to *GCspy* tool have been developed, that track and visualize dynamic memory allocations as nodes and graphs [21]. *BlueJ* is an IDE that provides UML notation of Java code which helps users visualize structure of the application [22]. Users can view source code of classes present in UML diagrams by clicking on them. *CoffeeDregs*, a dynamic analysis tool, has been developed to support and visualize debugging facilities [23]. *VisuAlgo* has been introduced to visualize a set of predefined algorithms. It shows the visual execution of an algorithm for a given input [24]. *Jeliot3* has been developed as a programming tool that enables users program and visualize step by step execution of the program in the form of animations. It is mainly focused on expression evaluation and is depicted by the movement of messages, method calls, values and references in the code [25]. Java Visualizer illustrates dynamic run-time behavior of program by moving back and forth in program execution[2]. JavelinaCode, a web-based IDE, supports synchronized visualization of static and dynamic aspects of Java source code [26]. *Pythontutor*, proposed by PhilipGuo visualizes the code by displaying the data structures used [3]. Another visualization tool called *PlayVisualizerC (PVC)* that dynamically visualizes the code in terms of memory allocations has been proposed by Ryosuke et al. in [27]. Visualization tools have been developed to help students learn programming, debugging and explore various possible ways of writing programs. Games have been developed to support students in program comprehension, various concepts of programming such as function calls, values, movement of messages and so on. To the best of our knowledge, there is no visualization tool or a game that can help programmers comprehend concepts of pointers and their implementation in the program without compilation of. Hence, in this paper, we propose *PointerViz* to address this issue. *PointerViz* aims to visualize the statements insitu IDE, before compilation, rather than visualizing

[1] https://github.com/AkhilaSriManasa/PointerViz

[2] http://www.cs.princeton.edu/~cos126/java_visualize/

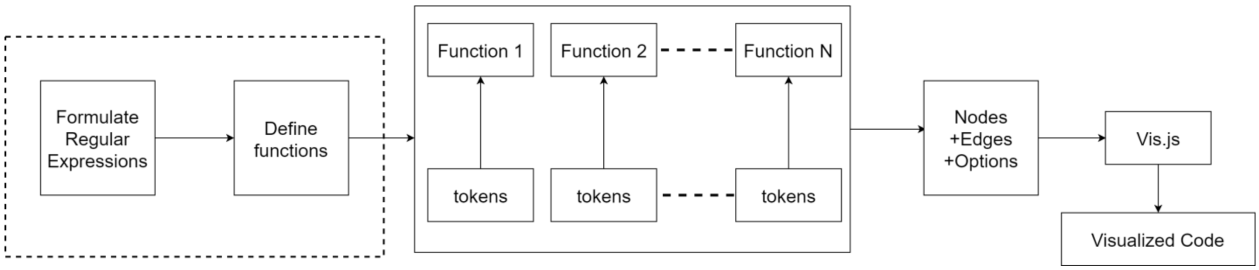


Figure 1. Approach for design of PointerViz

them after compilation of the code unlike existing works such as *pythontutor* [3] that visualizes the code after compilation. Visualizing statements on the go will help novice programmers in identifying and rectifying the mistakes if any, at the early stages of the code, as a result reducing the debugging efforts. Also, *PointerViz* displays primitive visualization of pointers, which is identical to the way budding programmers draw on paper, unlike *PVC* [27] which displays memory locations and internal details. This primitive representation can help novice programmers relate better to their interpretations.

3. Design of *PointerViz*

PointerViz prototype is currently developed as an extension[3] to Google Chrome. As a proof of concept, we developed *PointerViz* as a plugin to support an online compiler and interpreter, ideone.com[4], as shown in Figure 1. However, our plugin can be extended to support any other[5] online coding playgrounds such as *codepad.org* and *compileonline.com*. In its current form, *PointerViz* can also be added as an extension to browsers other than Google Chrome, such as Mozilla Firefox. Visualizations are generated using an open source Javascript framework, *vis.js*[6]. Vis.js enables us to dynamically visualize graphs within the browser by facilitating manipulation of and interaction with dynamic data and also customization of nodes. Vis.js also helps in handling large amounts of dynamic data, making it a suitable choice to generate visualizations in coding environments which involve considerably large amounts of dynamic data.

PointerViz is designed to support novice programmers get a better view of pointer data structures. It provides interactive visualization to the users. The floating representations of links among various nodes might motivate users to view representations and thus comprehend the concepts better. As the first step of design, we formulated regular expressions that portray various ways in which pointers are defined. The regular expressions used to match the statements written in the code by users, are as follows:

```
Regular expression [A]:
/[a-z]+\ \*\ ([a-zA-Z$_][a-zA-Z0-9$_]*)\;/gm;
recognizes examples such as : int *p;

Regular expression [B]:
/[a-z]+\ \*\ ([a-zA-Z$_][a-zA-Z0-9$_]*)\ *=\
*null\;/gmi;
recognizes examples such as : int *p = NULL;

Regular expression [C]:
/[a-z]+\ \*\ ([a-zA-Z$_][a-zA-Z0-9$_]*)\ *=
\ *\&\ ([a-zA-Z$_][a-zA-Z0-9$_]*)\;/gm;
recognizes examples such as : int *p = &a;

Regular expression [D]:
/[a-z]+\ ([a-zA-Z$_][a-zA-Z0-9$_]*)\
*\[\ *([0-9]+)\ *\]\;/g;
recognizes examples such as : int p[10];

Regular expression [E]:
/[a-z]+\ ([a-zA-Z$_][a-zA-Z0-9$_]*)\
*\[\ *([0-9]+)\ *\]\;/g;
recognizes examples such as : int p[9][10];

Regular expression [F]:
/[a-z]+\ *\*\ *([a-zA-Z$_][a-zA-Z0-9$_]*)\
*=\ *\"([a-zA-Z0-9$_]+)\"\ *;/gm;
recognizes examples such as : char *p="test";

Regular expression [G]:
[a-z]+\ *([a-zA-Z$_][a-zA-Z0-9$_]*)\ *\[\
*([0-9]+)\ *\]\ *\=\ *((\{('([[a-z]|[A-Z]|
[0-9]+|[$_]])' *\,*)*\})
|(\{(([0-9]+)\ *\,*)*\}));/gm;
recognizes examples such as : int p[2]={1,2};

Regular expression [H]:
/[a-z]+\ *([a-zA-Z$_][a-zA-Z0-9$_]*)\ *\[\
*([0-9]+)\ *\]\ *\[\ *([0-9]+)\ *\]=\
*((\{('([[a-z]|[A-Z]|[0-9]+|[$_]])'
*\,*)*\})|(\{(([0-9]+)\ *\,*)*\}));/gm;
recognizes examples such as :
char p[2][3]={'a','b','1','d','c','2'}
```

Regular expression [A] points to pointer declarations alone, where in the pointers point to a garbage value. Statements in which pointers are assigned NULL values are recognized by Regular expression [B]. Regular

[3]PointerViz can be installed on Mozilla Firefox as well.
[4]https://ideone.com/
[5]http://codepad.org/
[6]https://visjs.org/

expression [C] recognizes statements in which the pointers are assigned the address of a variable. Arrays of a given size can be obtained from Regular expression [D]. Two dimensional arrays of given row and column sizes are recognized by Regular expression [E]. Pointers to strings are recognized by Regular expression [F]. One dimensional arrays whose elements are defined are recognized by Regular expression [G] and 2D arrays whose elements are defined at the time of declaration are recognized by Regular expression [H].

For every regular expression that has been formulated, a function is defined to generate the corresponding visualization. Every visualization is a set of nodes with connections between them that are represented using edges. Node shapes are defined in functions in a way, such that they convey the semantic meaning of statements written by the user. The difference in address space between nodes is also represented in by a numeric value between the nodes, representing the number of bytes. These nodes and edges are passed as a dataset to previously stated visualization framework, vis.js to render visualizations.

Functions take parsed tokens as inputs and verify if they match any of the defined regular expressions. They return options of shapes, nodes and edges in the form of data. The function in script below deals with statements of the form defined in Regular expression [A].

```
function ptr_type1(code) {
        const regex_1 =/[a-z]+\
        \*([a-zA-Z$_][a-zA-Z0-9$_]*)\;/gm;
        const m = regex_1.exec(code);
        const identifier =m[1];
                .
                .
        const data = {
                nodes: nodes,edges: edges};
                return [data, options];
                }
```

Definitions for Edges, nodes and shape options are shown in the script below considering Regular expression [A] as an example, where nodes are defined to be displayed as circles and edges as arrows. Variables and pointers are defined to be displayed as circles and arrays are represented as rectangular boxes.

```
const identifier = m[1];
const nodes = [
    {id:0, label: identifier,
     group: "0", title: code},
    {id:2, shape: 'dot',
        label: "Garbage"}
        ];
const edges = [
    {from: 0, to: 2, arrows: 'to'}
        ];
const options = {
        nodes: {
        shape: 'circle',
        size: 30,
        font: { size: 30,
                multi: true
            },
        borderWidth: 2,
        shadow:true
        },
        edges: {
                width: 3,
                }
        };
```

Visualization is updated regularly at the end of every statement. Code is processed in the form of tokens, which are fed into each of the pattern recognizing functions. The function which contains matching regular expression is executed. New patterns that are not defined previously can easily be added, by defining new functions the newly defined patterns, making it easy to extend this plugin from a developer's perspective.

4. Working of *PointerViz*

The main aim of *PointerViz* is to display the references of pointer data structures as used in the code written by the user. Workflow of *PointerViz* is a seven step mechanism, as shown in Figure 2.

- *Step 1*: User enters a statement in desired programming language among C or C++ in the text space provided by *ideone.com*, as the current prototype is being tested for*ideone.com*.

- *Step 2*: Tokens are extracted from these statements and are considered as individual statements. Token extraction is done with help of *pointerViz.js script* that filters out statements by semi colons used. Even if users enter various statements in the same line, they can be separated out by considering semi colon symbols.

- *Step 3*: Extracted tokens are passed to *pointerViz.js file*, which forms the basis of plugin. Functions written in the script process these tokens and compare them with regular expressions which are defined apriori.

- *Step 4*: The *pointerViz.js file* generates nodes edges based on the matched regular expressions. It then renders shapes corresponding to these regular expressions, as defined in each function.

- *Step 5*: Resulting nodes and edges are passed to *vis.js framework* to produce results by processing them. It provides overlays for the same as mentioned in the *pointerViz.js* file.

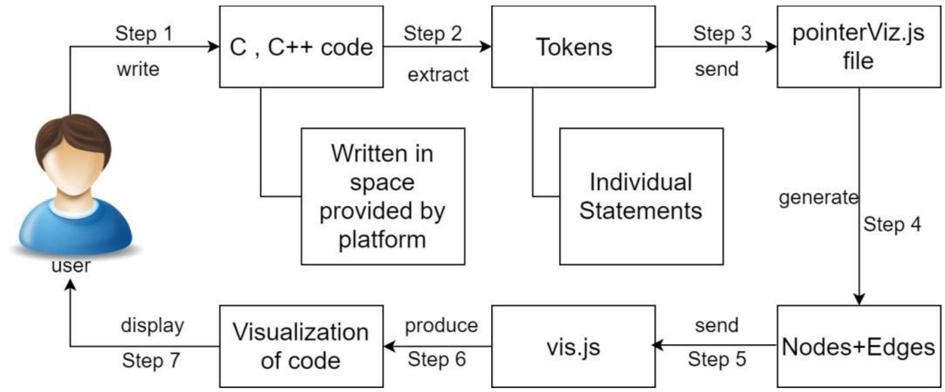


Figure 2. Stepwise Working of PointerViz

- *Step 6*: Results are produced in the form of visualizations based on the code entered by user.

- *Step 7*: The results obtained in the previous step are displayed to the user in the space provided below the existing text space. Users are provided with facilities to change orientations of the figures and also to view code that resulted in creation of the node by hovering on the first node.

For each line entered by the user on the console provided, *PointerViz* compares the statement with predefined regular expressions. For a correct match, the shapes and their relations are displayed. These shapes help novice users to differentiate among various types of pointer declarations. The pointers are visualized using circles with arrows emerging from these circles. These arrows point to the referenced variables as per the identified tokens. The shapes used to represent these data structures contain respective variables that have been used by users in their code for better readability.

Semi colons are used as delimiting symbols to separate tokens in monolithic code. As we have implemented *PointerViz* for programming languages like C and C++, it identifies next line or statement by semi colon. As new statements are entered, new visualizations with respect to those statements are generated and displayed to the user. Visualizations of previously entered statements are maintained to help users revisit those structures instead of re-writing the same statements. Visualizations are done in *First Come First Display pattern* i.e., visualizations of latest statements are displayed at the bottom, similar to push operations in queue. Users can select and drag nodes in the displayed visualizations to alter their orientation.

5. User Scenario

Suppose *Veda* is a novice programmer working on C programming language and she wishes to learn the how pointers refer to various variables in the program. She then visits *ideone.com*, an online coding playground, selects C as the language she wishes to code.

- **Case 1:** She starts typing the first statement:

  ```
  int * p ;
  ```

  Visualization of this statement is displayed. She then hovers on the *node p* displayed in the visualizations. The code that resulted in creation of *node p* is displayed above this node as shown in Figure 3.

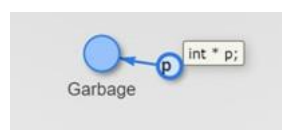


Figure 3. Visualization of code for Case 1

- **Case 2:** She writes another statement in addition to the previous statement resulting in the following code

  ```
  int * p ;
  int * q = NULL;
  int * s = &a ;
  ```

  A visualization of the statement is displayed with **s** as a circular node and an arrow originating from this node to another circular node **a** as shown in Figure 4. Also, the statement where a pointer is assigned a NULL value is represented with a circular node **q** pointing to another circular node having the value as **NULL**. These statements are visualized and displayed below the previous statement as shown in Figure 4.

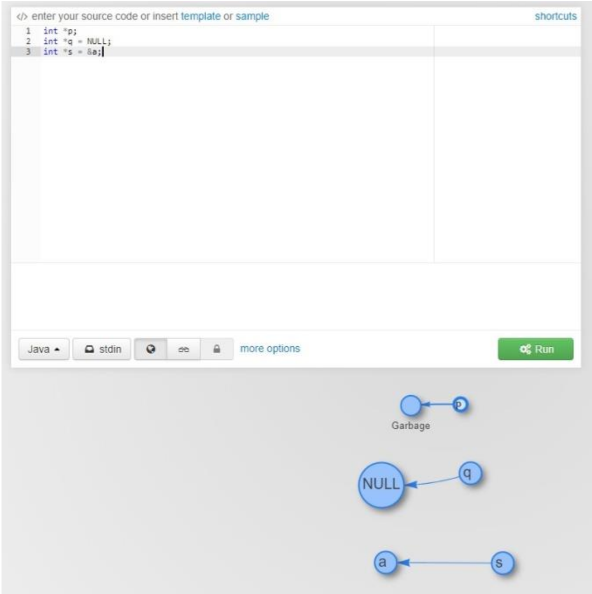


Figure 4. Screenshot of PointerViz showing visualization of code for Case 2

- **Case3:** Veda adds another statement that declares an empty array of size five.

  i n t p [5] ;

  Veda can view visualization of the statement by scrolling down the page as in Figure5. The difference between memory address of one element of the array to the next element is represented with the numeric value of number of bytes that differ, on a line between the elements. Since the entered array is an integer array, each element occupies a memory of 4 bytes and hence, memory locations of consequent elements differ by 4 bytes, as indicated between the nodes in Figure 5.

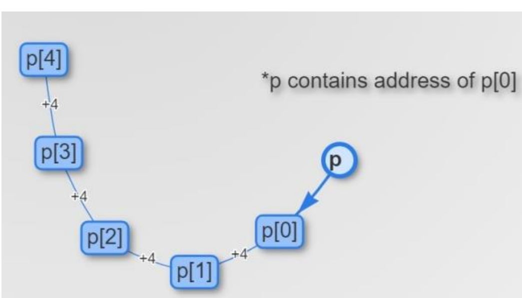


Figure 5. Visualization of code for Case 3

- **Case 4:** When Veda writes a statement with a two dimensional character array of two rows and three columns.

  char p [2] [3] ;

  Veda can view visualization on the page as in Figure 6. Since, the array is a character array, each element of the array occupies 1 byte in the memory and hence, it indicates that one element occupies the memory location that is equal to memory location of the preceding element+1.

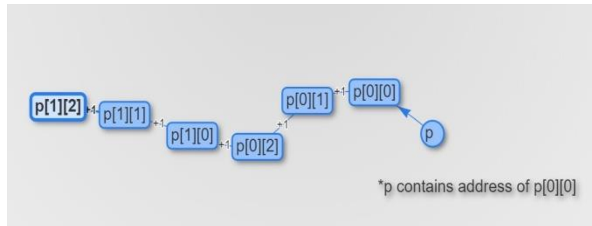


Figure 6. Visualization of code for Case 4

- **Case 5:** Veda, then writes a statement representing a pointer to string:

  ```
  char *p = "moksha";
  ```

  The above statement is then displayed to Veda as shown in Figure 7, where the pointer variable **p** points to the given string and stores the address of the first character in the string (**m** in this example).

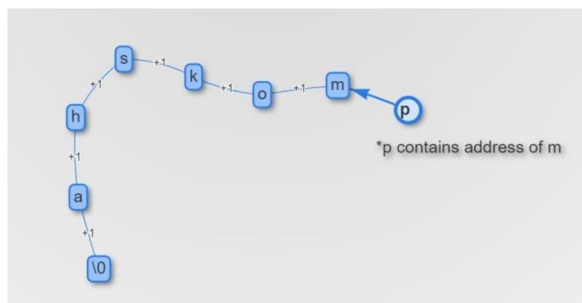


Figure 7. Visualization of code for Case 5

- **Case 6:** Veda writes another statement that defines a character array of size3.

  i n t p [3] = { ' a ' , ' s ' , 'm ' } ;

  Visualization of the above statement is displayed as shown in Figure 8.

- **Case 7:** Visualization of a two dimensional array defined at the time of declaration as given below, is displayed as shown in Figure 9.

  i n t p [2] [3] = { 1 , 3 , 5 , 7 , 9 , 2 } ;

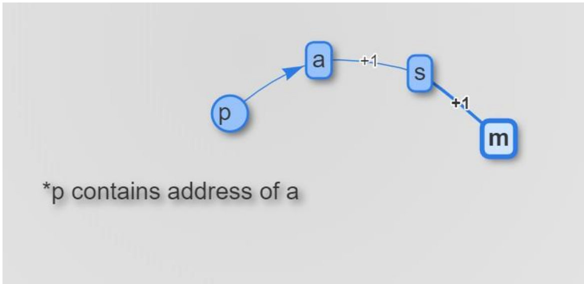


Figure 8. Visualization of code for Case6

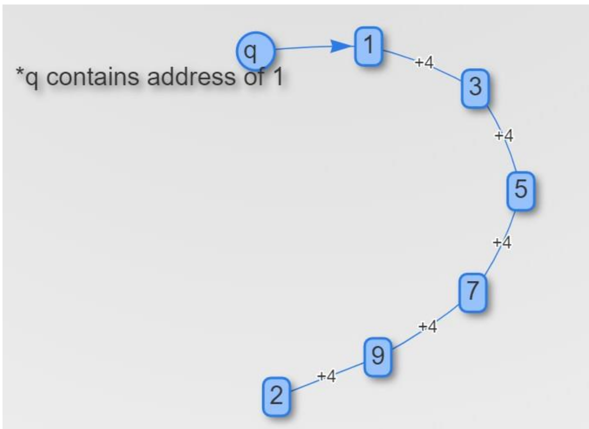


Figure 9. Visualization of code for Case7

6. Evaluation

To evaluate *PointerViz*, we have conducted a user experience study with 40 volunteers, in the age group of 18-20 years, from various universities. The participants were asked to install our *PointerViz* plugin as an extension to Google Chrome browser, on their personal desktops or laptops. They were also provided with a slide-show depicting the procedure to install *PointerViz*, a sample working video of the plug-in and few sample statements that contain pointers, which served as a basic tutorial. They were then asked to write code that involved snippets containing pointers using*ideone.com*. The participants were suggested to view and verify visualizations displayed based on the code that they have written. A user survey has been conducted with the help of a five point Likert Scale. A questionnaire as provided in Table 1, in which each question has to be rated on a scale of 1 to 5, has been sent to volunteers to assess their experience and evaluate *PointerViz*.

7. Results

As reported in Figure 10, *PointerViz* had a good user-friendly interface (83% in Q1). In Q2, about 82% of participants have agreed that *PointerViz* has visualized the statements clearly and correctly. The ratings in Q3 and Q4 indicate that *PointerViz* has helped about 77% of participants learn about various ways of using pointers and that the experiment has been considerably interesting (80% in Q4). However, they have also suggested increasing the scope of visualization to various definitions of pointers. In Q5, most of the participants have agreed that they would recommend *PointerViz* to their peers (83%).

Table 1. Questions in survey using a 5-point Likert Scale.

Q1: How easy was it to use *PointerViz* interface? (1=very easy, 5=very difficult)
Q2: *PointerViz* has visualized pointer data structures clearly and correctly. (1=strongly agree, 5=strongly disagree)
Q3: *PointerViz* has helped me in learning about various ways of usages of pointer data structures. (1=strongly agree, 5=strongly disagree)
Q4: *PointerViz* has kept the whole experiment interesting and informative. (1=strongly agree, 5=strongly disagree)
Q5: I will recommend *PointerViz* to my peers. (1=strongly agree, 5=strongly disagree)

8. Discussion and Limitations

The core idea of this paper is to apply the concept of visualization to aid users in understanding critical concepts in programming languages. One of the critical aspects as identified by researchers is pointers [12]. *PointerViz* prototype is a first step towards supporting critical programming concepts through visualization. In order to do an in-line visualization of code, the current implementation of the tool uses lexical analysis and parsing at statement level instead of block level. We have limited the scope of *PointerViz* to understand individual statements and visualize the same. We shall hence extend *PointerViz* to support analysis of complete code considering relations among the statements in code in the future versions. While we initially planned to map understanding pointers with different levels of *Bloom's taxonomy*, we limited our scope to basic concepts in the current version.

While the idea seems to be simple, we aim to extend this to support visualization of pointers in cases where

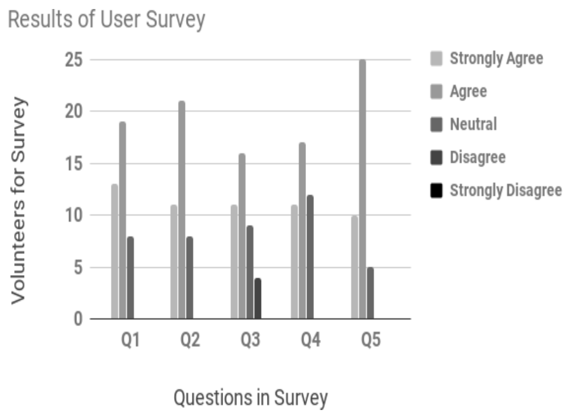


Figure 10. Results of User Survey Questionnaire

pointers deal with various programming concepts such as use of pointers in a function, array of pointers, linked lists and use of pointers in user defined data types such as structures. Though the current prototype focuses only on visualizing code statement wise, based on the feedback we received, *PointerViz* could help novices get a better understanding of the pointers and references.

9. Conclusion and Future Work

In this paper, we have introduced *PointerViz* to visualize pointers, as a prototype extension to Google Chrome web browser that augments *ideone.com*. As pointers are considered to be one of the critical aspects of learning programming, our work aims to support novice programmers learn better [12,1]. *PointerViz* prototype has visualized code written by users upto a decent level of satisfaction, owing to 82% of participants willing to recommend this plugin to their peers. *PointerViz* can easily be extended to support other online coding platforms as well. As reported by survey participants, one most important suggestion is to extend *PointerViz* for other usages of pointers such as linked lists and doubly linked lists. We plan to extend the plugin to support multiple scenarios of pointers pointing to pointers, array of pointers. We shall also extend the plugin to include display of timely visualizations of code blocks. We see this work as a first step towards improving program comprehension through visualization that could help novice programmers.

Acknowledgements

We thank all the volunteers for their valuable time and honest feedback that helped us in evaluating *PointerViz*.

References

[1] Y. Bosse and M. A. Gerosa, "Why is programming so difficult to learn?: Patterns of difficulties related to programming learning mid-stage," *ACM SIGSOFT Software Engineering Notes*, vol. 41, pp. 1–6, 01 2017.

[2] A. Luxton-Reilly, E. McMillan, E. Stevenson, E. Tempero, and P. Denny, "Ladebug: an online tool to help novice programmers improve their debugging skills," in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pp. 159–164, ACM, 2018.

[3] P. J. Guo, "Online python tutor: embeddable web-based program visualization for cs education," in *Proceeding of the 44th ACM technical symposium on Computer science education*, pp. 579–584, ACM, 2013.

[4] S. H. Kim and J. W. Jeon, "Programming lego mindstorms nxt with visual programming," in *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*, pp. 2468–2472, IEEE, 2007.

[5] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk, "Programming by choice: Urban youth learning programming with scratch," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, (New York, NY, USA), pp. 367–371, ACM, 2008.

[6] A. Marron, G. Weiss, and G. Wiener, "A decentralized approach for programming interactive applications with javascript and blockly," in *Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions*, pp. 59–70, ACM, 2012.

[7] M. Muratet, P. Torguet, J.-P. Jessel, and F. Viallet, "Towards a serious game to help students learn computer programming," *International Journal of Computer Games Technology*, vol. 2009, p. 3, 2009.

[8] C. North and B. Shneiderman, "Snap-together visualization: can users construct and operate coordinated visualizations?," *International Journal of Human-Computer Studies*, vol. 53, no. 5, pp. 715–739, 2000.

[9] E. L. Glassman, T. Zhang, B. Hartmann, and M. Kim, "Visualizing api usage examples at scale," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, p. 580, ACM, 2018.

[10] B. W. Kernighan and D. M. Ritchie, *The C programming language*. 2006.

[11] A. D. Robison and P. F. Dubois, "C++ gets faster for scientific computing," *Computers in Physics*, vol. 10, no. 5, pp. 458–462, 1996.

[12] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *Acm Sigcse Bulletin*, vol. 37, no. 3, pp. 14–18, 2005.

[13] S. Bassil, R. K. Keller, *et al.*, "Software visualization tools: Survey and analysis.," in *IWPC*, pp. 7–17, 2001.

[14] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *2004 IEEE Symposium on Visual Languages-Human Centric Computing*, pp. 199–206, IEEE, 2004.

[15] A. Scott, M. Watkins, and D. McPhee, "E-learning for novice programmers; a dynamic visualisation and problem solving tool," in *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pp. 1–6, IEEE, 2008.

[16] B. Broll, A. Le´deczi, P. Volgyesi, J. Sallai, M.Maroti, A. Carrillo, S. L. Weeden-Wright, C. Vanags, J. D. Swartz, and M. Lu, “A visual programming environment for learning distributed programming,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pp. 81–86, ACM, 2017.

[17] A. Vahldick, A. J. Mendes, and M. J. Marcelino, “A review of games designed to improve introductory computer programming competencies,” in Frontiers in Education Conference (FIE), 2014 IEEE, pp. 1–7, IEEE, 2014.

[18] S. Leutenegger and J. Edgington, “A games first approach to teaching introductory programming,” in ACM SIGCSE Bulletin, vol. 39, pp. 115–118, ACM, 2007.

[19] M. A. Miljanovic and J. S. Bradbury, “Robot on!: a serious game for improving programming comprehension,” in Games and Software Engineering (GAS), 2016 IEEE/ACM 5th International Workshop on, pp. 33–36, IEEE, 2016.

[20] M. A. Miljanovic and J. S. Bradbury, “Robobug: A serious game for learning debugging techniques,” in Proceedings of the 2017 ACM Conference on International Computing Education Research, pp. 93–100, ACM, 2017.

[21] A. M. Cheadle, A. Field, J. Ayres, N. Dunn, R. A. Hayden, and J. Nystrom-Persson, “Visualising dynamic memory allocators,” in Proceedings of the 5th international symposium on Memory management, pp. 115–125, ACM, 2006.

[22] M. Ko¨lling, B. Quig, A. Patterson, and J. Rosenberg, “The bluej system and its pedagogy,” Computer Science Education, vol. 13, no. 4, pp. 249–268, 2003.

[23] C. Huizing, R. Kuiper, C. Luijten, V. Vandalon, et al., “Visualization of object-oriented (java) programs.,” in CSEDU (1), pp. 65–72, 2012.

[24] S. Halim, “Visualgo,” Dostupne´ net/en, 2015.
z:¡ https://visualgo.

[25] A. Moreno and M. S. Joy, “Jeliot 3 in a demanding educational setting,” Electronic Notes in Theoretical Computer Science, vol. 178, pp. 51–59, 2007.

[26] J. Yang, Y. Lee, and D. Hicks, “Synchronized static and dynamic visualization in a web-based programming environment,” in Program Comprehension (ICPC), 2016 IEEE 24th International Conference on, pp. 1–4, IEEE, 2016.

[27] R. Ishizue, K. Sakamoto, H. Washizaki, andY. Fukazawa, “Pvc: Visualizing c programs on web browsers for novices,” in Proceedings of the 49th ACM Technical Symposium on Computer Science Education, pp.245–250,ACM,2018.

ऋषिराजपोपटवर्यस्य शोधप्रबन्धस्य वृत्तान्तः

✍ चैतन्यः सु लक्कुण्डी
Email : chaitanya.lakkundi@gmail.com

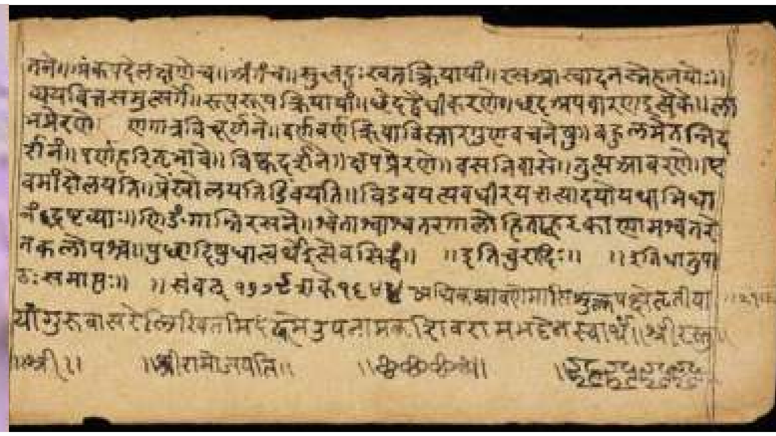
'वयं पाणिनौ विश्वसिमः...' इत्यस्य शोधप्रबन्धस्य विषये बहुधा चर्चा श्रूयते । वार्तापत्रिकासु अपि अनेकविधाः वार्ताः प्रकाशिताः । एतस्य सर्वस्य यत् मूलं तस्य आशयः कः, कश्च सारः इति स्थूलतया विवृणोति अयं लेखः ।

एषु दिनेषु प्रायः अस्माभिः पत्रिकासु पठितं स्यात् – 'केम्ब्रिज्विश्वविद्यालयस्थेन भारतीयशोधच्छात्रेण आ पञ्चशतोत्तरद्विसहस्रात् वर्षेभ्यः अवशिष्टः प्रश्नः समाहितः' इति । 'युरेकाक्षणः' इत्याख्यं लेखम् अधिकृत्य विभिन्नेषु माध्यमेषु प्रचारः चर्चा च जाता बहुधा । जनानां प्रतिस्पन्दनम् अपि अविस्मरणीयम् । अनेन कारणेन बहूनां जनानां मनसि संस्कृतं प्रति, विशिष्य व्याकरणं प्रति कौतुकम् अपि उत्पन्नम् ।

ऋषिराजपोपटनामकः केम्ब्रिज्विश्वविद्यालयस्य शोधच्छात्रः । सः विद्यावारिधेः निमित्तं यं शोधप्रबन्धं लिखितवान् सः तदीयेन विश्वविद्यालयेन अङ्गीकृतः । तस्य एव प्रसारः प्रचारः डा. ऋषिराजपोपटवर्येण कृतः । तस्य शोधप्रबन्धस्य शिरोनाम अस्ति – 'In Panini We Trust: Discovering the Algorithm for Rule Conflict Resolution in the Astadhyayi' (वयं पाणिनौ विश्वसिमः – अष्टाध्याय्यां नियमविरोधस्थलेषु समाधानस्य प्रक्रियायाः अन्वेषणम्) इति ।

अत्र उल्लिखिता 'अष्टाध्यायी' व्याकरणशास्त्रीयः कश्चन प्राचीनः ग्रन्थः, यश्च पाणिनिमहर्षिणा रचितः । ग्रन्थेऽस्मिन् संस्कृतभाषायाः नियमाः सूत्ररूपेण निबद्धाः सन्ति । एतानि सूत्राणि उपयुज्य साधुशब्दानां ज्ञानं प्राप्तुं शक्यम् । साधुशब्दानां निर्माणकार्यमेव 'प्रक्रिया' इति नाम्ना सूच्यते । सूत्रैः बहूनि कार्याणि विहितानि भवन्ति ।

वयं जानीमः यत् सर्वेषां पदानां काचित् प्रकृतिः, कश्चन प्रत्ययः च

ऋषिराजपोपटः, अष्टाध्यायीस्थधातुपाठसम्बद्धं तालपत्रं च

भवति इति । उदाहरणार्थं जानाति इति रूपं पश्याम । जानाति इति रूपं साधनीयं चेत्, ज्ञा इति धातुं स्वीकृत्य तत्र तिप् इति प्रत्ययं विधास्यामः । ज्ञा + तिप् इति स्थिते तत्र 'ज्ञा'धातौ कश्चन विकारः भवति, येन 'जानाति' इति अन्तिमं पदं निष्पद्यते । ईदृश्यां प्रक्रियायां प्राधान्येन कार्यत्रयं सम्भवति । तानि कार्याणि च – १. नूतनवर्णस्य आगमः २. वर्णस्य लोपः ३. वर्णस्य विकारः (परिवर्तनम्) चेति ।

पाणिनीयैः सूत्रैः कुत्र आगमः, कुत्र लोपः, कुत्र च विकारः इति विधानं क्रियते । यदा एकस्मिन् स्थले सूत्रद्वयेन कार्यद्वयं विहितं भवति तदा सहजतया शङ्का उत्पद्यते यत् कतरत् कार्यं करणीयम् ? इति । एतादृशान् प्रसङ्गान् परिहर्तुं पाणिनिमहर्षिणा एकं परिभाषासूत्रं रचितम् । तत् सूत्रमस्ति – 'विप्रतिषेधे परं कार्यम्' इति ।

अष्टाध्याय्याम् अष्टसु अध्यायेषु विभक्तानि सूत्राणि क्रमेण रचितानि । तत्र विप्रतिषेधे (द्वयोः कार्ययोः युगपत् प्राप्तयोः) परसूत्रस्य कार्यं करणीयम् इति नियमः । अर्थात् यस्य सूत्रस्य क्रमाङ्कः अधिकः वर्तते तस्य प्रवृत्तिः भवति । परशब्दस्य 'उत्तरवर्ति सूत्रम्' इत्यर्थः पारम्परिकैः वैयाकरणैः स्वीक्रियते ।

राजपोपटवर्येण एतस्य सूत्रस्य अवगमने महान् प्रयासः विहितः इति तु प्रबन्धपठनात् अवगम्यते । तदीयः प्रयासः श्लाघ्यः एव । परन्तु तत्र परिपूर्णता अस्ति किम् इति कश्चन प्रश्नः व्याकरणज्ञानां पुरतः उपस्थितः भवति ।

राजपोपटः अभिप्रैति – कात्यायनपतञ्जलि-प्रभृतिभिः वैयाकरणैः अस्य सूत्रस्य सर्वथा दोषपूर्णं व्याख्यानं कृतम् इति । अत एव सः उत्तरवर्तिभिः वैयाकरणैः कृतान् ग्रन्थान् सिद्धान्तान् च निराकरोति । अष्टाध्यायीम् अतिरिच्य अन्यत् सर्वं प्रमाणत्वेन न अङ्गीकरोति अयं जनः । अतः स्वस्य स्वतन्त्रं व्याख्यानं कर्तुं प्रयत्यते तेन ।

तन्मते परशब्दस्य अर्थः – 'दक्षिणभागः' इति । अतः दक्षिणभागे यत् कार्यं प्राप्तं तत् प्रवर्तनीयम् इति तस्य प्रतिपादनम् । ज्ञा + ति इत्यस्मिन् स्थले यदि एकं कार्यं 'ज्ञा' इत्यत्र प्राप्तं, द्वितीयं कार्यं 'ति' इत्यत्र प्राप्तं तर्हि द्वितीयं कार्यमेव प्रवर्तते इति तदीयः आशयः । यतो हि 'ति' इति दक्षिणभागे विद्यमानम् ।

वस्तुतः परशब्दस्य बहवः अर्थाः सन्ति । तत्र 'दक्षिणभागः' इत्यपि अर्थः अष्टाध्याय्याम् बहुत्र अस्ति एव । परन्तु प्रकृतसूत्रे कः अर्थः परिग्रहीतव्यः इति तु निर्णेतव्यम् । तत् निर्णेतुं बहवः शब्दाः परिशीलनीयाः । राजपोपटस्य व्याख्यानस्य स्वीकरणात् यदि साधुशब्दाः एव

निष्पन्नाः भवेयुः तर्हि सः अर्थः स्वीकार्यः स्यात् अवश्यमेव, अन्यथा तु तत् व्याख्यानं स्वीकारार्हं न भवेत् । एतादृशे प्रसङ्गे कथं व्यवहरणीयम् इति अंशं भारतीयपरम्परागतं किञ्चन सुभाषितं ज्ञापयति । तत् च इत्थम् –

युक्तियुक्तं वचो ग्राह्यं बालादपि शुकादपि ।
युक्त्ययुक्तं वचस्त्याज्यं साक्षादपि बृहस्पतेः ।।

इति ।

इदानीं सूत्रार्थं पुनः अवगच्छाम । पारम्परिकमते यत्र द्वयोः तुल्यबलवतोः (समानबलयुक्तयोः) सूत्रयोः प्राप्तिः युगपत् भवति तत्रैव विप्रतिषेधः । परन्तु राजपोपटस्य मतं तु – यत्र क्वापि एकस्मिन् सोपाने बहूनां सूत्राणां प्राप्तिः भिन्नस्थलेषु भवति (यत्र स्थानिनः अनेके सन्ति) तत्र विप्रतिषेधः, तत्र च दक्षिणभागे विद्यमानस्य स्थानिनः कार्यं कर्तव्यम् इति ।

राजपोपटेन स्वस्य प्रबन्धे पञ्चाशदधिकानि उदाहरणानि निरूपितानि, यत्र तन्मते साधुशब्दः निष्पन्नो भवति । स्थाने स्थाने तेन अन्येषां सूत्राणाम् अर्थः अपि यथेष्टं स्वबुद्ध्या परिवर्तितः अपि । तद्विषये केषुचित् स्थलेषु तेन युक्तयः प्रस्तुताः, पुनः केषुचित् च नैव प्रस्तुताः ।

अद्यत्वे बहुभिः पारम्परिकपण्डितैः एतस्य प्रबन्धस्य समालोचनम् अपि कृतं वर्तते । तैः दर्शितं यत् बहुषु प्रसङ्गेषु असाधुशब्दानां निष्पत्तिः अपि राजपोपटस्य मतानुसरणेन भवति इति । तेषां समाधानं च प्रबन्धद्वारा नैव प्राप्तम् । संस्कृत-भारत्याः दक्षिणकर्णाटकशाखाद्वारा आयोजितायां युवविद्वद्व्याख्यानमालायां श्रीमता नीलेशबोडस-वर्येण विस्तृतरूपेण अनेकानि उदाहरणानि प्रदर्शितानि, यत्र असाधूनि रूपाणि सिद्ध्यन्ति । लक्षाधिकानि दुष्टरूपाणि नूतनमतस्य स्वीकारेण सम्भवन्ति इति भासते । कीदृशेषु स्थलेषु तस्य क्रमः साधूनि रूपाणि यच्छति, कियती वा तस्य व्याप्तिः इत्यादयः विषयास्तु शोधनीयाः ।

कश्चन अपरः पक्षः राजपोपटमहोदयेन उपस्थापितः । सः वदति – 'सङ्गणकार्थम् इतः प्रयोजनम् अस्ति' इति । यः कोऽपि विधिः क्रमः वा नियमसमन्वितः यदि स्यात् तर्हि तस्य सङ्गणकान्वयः कर्तुं शक्यः एव । परन्तु एतदीयं मतम् अवलम्ब्य असाधूनि रूपाण्येव सिद्ध्यन्ति इति कारणतः स्पष्टं यत् यन्त्रद्वारा अपि असाधून्येव रूपाणि निष्पाद्यन्ते इति ।

अन्ततो गत्वा 'यथास्थितं तन्मतम् अङ्गीकर्तुं नैव शक्यते' इति निर्णेतव्यं भवति अनन्यगतिकतया । एवं सत्यपि ये अंशाः राजपोपटवर्यस्य प्रबन्धात् स्वीकारयोग्याः भवेयुः ते हंसक्षीरन्यायेन अवश्यमेव ग्राह्याः ।

आदौ अस्माभिः अवगन्तव्यं यत् अस्य सूत्रस्य ('विप्रतिषेधे परं कार्यम्' इति सूत्रस्य) अवगमने व्याख्याने च पारम्परिकाणां दोषः नैव आसीत् । इयं वस्तुतः समस्या अपि न । रहस्यमपि न । वैयाकरणाः तु मुनित्रयस्य प्रामाण्यम् अङ्गीकुर्वन्ति । तत्र पाणिनिः, कात्यायनः, पतञ्जलिः चेति त्रयः मुनयः । एतेषां परम्परा एव 'अष्टाध्यायीपरम्परा' इति उच्यते । अनया लौकिकानां वैदिकानां च साधुशब्दानां सिद्धिः क्रियते । अद्य तावत् अस्याः व्याकरणपरम्परायाः सहस्राधिकवर्षाणि व्यतीतानि । तत्र उत्कृष्टैः विद्वद्भिः महर्षिभिः च उद्यमः कृतः । समीचीनाः सङ्गताः च सिद्धान्ताः प्रतिपादिताः । अतः इयं परम्परा सर्वथा प्रामाणिकी । तत्र परिष्कारादिकं कर्तव्यं चेत् पारम्परिकपद्धत्या अधीत्य विदुषः प्रणम्य तेषां सम्मतिं च स्वीकृत्य कर्तव्यम् । तत् अननुष्ठाय क्रियमाणः प्रयासः विद्वल्लोके मान्यतां न प्राप्नुयात् । विद्वल्लोकमान्यतारहितस्य लौकिक-प्रचारस्य न किमपि मूल्यं संस्कृतक्षेत्रे ।

SOTagger - Towards Classifying Stack Overflow Posts through Contextual Tagging

Akhila Sri Manasa Venigalla
Indian Institute of Technology
Tirupati, India
cs18m017@iittp.ac.in

Chaitanya S. Lakkundi
Indian Institute of Technology
Tirupati, India
cs18s502@iittp.ac.in

Sridhar Chimalakonda
Indian Institute of Technology
Tirupati, India
ch@iittp.ac.in

***Abstract*—There is an ever increasing growth in the use of Q&A websites such as Stack Overflow (SO), so are the number of posts on them. These websites serve as knowledge sharing platforms where Subject Matter Experts (SMEs) and developers answer questions posted by other users. It is effort intensive for developers to navigate to right posts because of the large volume of posts on the platform, despite the presence of existing tags, that are based on technologies. Tagging these posts based on their context and purpose might help developers and SMEs in easily identifying questions they wish to answer and also in identifying contextually similar posts. To support this idea, we propose *SOTagger* as a prototype plug-in for Stack Overflow to tag questions contextually. We have considered SO data provided on *SOTorrent* and automated the identification of 6 categories of questions using Latent Dirichlet Allocation. We have also manually verified relevance of these categories. Using these categories and dataset, we have built a classification model to classify a post into one of these six categories using Support Vector Machine. We have evaluated *SOTagger* by conducting a user survey with 32 developers. The preliminary results are promising with about 80% developers recommending the plugin to others.**

***Index Terms*—Stack Overflow, Contextual Tagging, LDA**

I. Introduction

Stack Overflow (SO) is one of the most frequently used websites with about 11M visits every day. With a user base of 10M users, about 7.3K questions are posted per day. It comprises of about 18 million questions, of which 71% are answered[1]. These questions correspond to various technical categories, tools, libraries and are tagged into atmost 5 of 54K tags[2] present on the website. This tagging is done based on their technical relevance with the posted content and is used to organize posts and thus help users to browse for questions and answers concerning to particular topics such as *javascript, jquery, python* and so on [1]. However, these tags don't classify questions based on the context in which they are asked. The context would capture situations pertaining to conceptual understanding, issue resolving and so on.

Recent studies have aimed at classifying questions on SO based on their context and arrived at almost similar taxonomies of categories. They have used various techniques such as K-NN clustering [2], automatic categorization by topic modeling using LDA and MALLET [3] and manual categorizations [1], [4]. Some of these studies have aimed to contextually categorize technology-specific questions such as questions related to Android application development [2] and mobile operating systems like *Android, Apple* and *Microsoft Windows*. However, existing tools do not categorize posts on SO platform based on context. To this end, the contributions of this paper are as follows:

- *SOTagger*[3] - a prototype plug-in that classifies posts on SO into six categories: *Conceptual, Discrepancy, Implementation, Error, Learning and MWE (Minimum Working Example)*.
- Application of NLP techniques - Latent Dirichlet Allocation(LDA) and Machine learning (ML) classifier - Support Vector Classifier (SVC) to classify SO posts.
- Evaluation of *SOTagger* with 32 professional developers and manual cross-verification of 100 posts.

II. Related Work

In the recent years, several studies have been done to analyze posts on SO, which include analyzing developers' area of interest based on questions asked [5], analyzing and suggesting tags of the questions [2] [1] [6] [7], identifying difficulties faced by developers [8], identifying trending technological topics [9], and so on. Researchers have classified posts on SO based on the context by manually interviewing software developers. In a survey conducted by Latoza et al., 179 professional software developers were asked to identify hard-to-answer questions pertaining to code that they solicit wherein 371 questions were reported. They have manually categorized them into 21 categories with 94 distinct questions, of which the 5 most frequently reported categories were - *Rationale, Intent and Implementation, Debugging, Refactoring and History of code* [10].

Studies have been conducted to investigate various question categories based on the context in which they

DOI reference number: 10.18293/SEKE2019-067

[1]https://stackexchange.com/sites?view=list#traffic

[2]http://bit.ly/SONumTags

[3]https://github.com/chaitanya-lakkundi/SOTagger

were asked. Rosen et al. manually categorized 380 posts on SO into 3 question categories based on the three interrogative words- *How, What and Why*, corresponding to three mobile operating system categories - *Android, Apple* and *Microsoft Windows* [4]. Treude et al. have manually classified 385 questions on SO into 10 categories - *How to, Decision Help, Discrepancy, Environment, Error, Conceptual, Review, Non-Functional, Novice, Noise* [1]. Although methods involving manual effort are necessary to capture ground truth, we see a need to find better ways to scale this approach such that automation is possible.

Elucidating further studies, Beyer et al. have proposed 7 question categories - *API Change, API Usage, Conceptual, Discrepancy, Learning, Errors, Review* by manually classifying 500 SO Android posts and performed automatic classification using supervised machine learning algorithms with a precision of 88% [2]. Allamanis et al. found 5 major question categories using LDA and unsupervised machine learning algorithm [3].

Insofar as the development in methods of classification is concerned, the research community has progressed from significant manual studies to automating them using machine learning algorithms and NLP techniques. Contemporary tools such as EnTAGREC++ [6], TagCombine [7] have been developed to provide tag suggestions to users when they post questions on SO. These tools suggest tags based on technologies involved in the post content. The prototype plug-in we propose, *SOTagger*, tags posts on SO based on their purpose or intent rather than considering the technologies involved. Based on the existing work on classifying posts [2] [1] [4] [3], we propose a taxonomy to tag posts contextually.

III. Proposed Taxonomy

Posts can be classified using several NLP techniques such as LDA, LSA, TF-IDF. However, inline with the existing work, we followed LDA technique.

We present six question categories that we have derived from existing studies and results obtained from LDA topic modeling. As a result of LDA topic modeling configured for 6 topics, we obtained 6 topics characterized by keywords for each topic, along with the weightage of keywords in every topic. Omitting the technical terms and considering interrogatives, it has been observed that *Topic 0* comprises of *discrepancy*, *Topic 1* contains *error*, *Topic 2* contains *how-to or implementation*, *Topic 3* contains *learning*, *Topic 4* contains *conceptual* and *Topic 5* contains *MWE* keywords respectively, as shown in Table I. These results obtained by applying LDA on SO posts indicate the presence of contextual categories in SO data. Comparing these results with the existing taxonomy discussed by Beyer et al. in [2] and other taxonomies presented in [1] [4] [3], we reorganize few categories in the existing literature and arrive at labelling five of these six topics as *conceptual, discrepancy, implementation, error and learning* respectively. We

TABLE I
Taxonomy of Question Categories

S.No.	Topics	Keywords
1	Conceptual	What is use/difference, Is there a way, Is it possible[2]
2	Discrepancy	doesn't work, tried to, have/facing problem, before upgrade previous version [2]
3	Implementation	How to implement [4] [3] [1]
4	Error	Exception, error [2]
5	Learning	suggest, tutorial, where can I find [2]
6	MWE	for this code, code tags

observed that many of the posts on SO contained code snippets, which could indicate that users post questions containing code to reproduce the bug they are facing. Such code snippets serve as *Minimum Working Examples (MWE)*[4], which is proposed as another category *MWE*. We observe this naming to be inline with work proposed by Allamanis et al. [3]. Each post can be classified into one or more of these six categories.

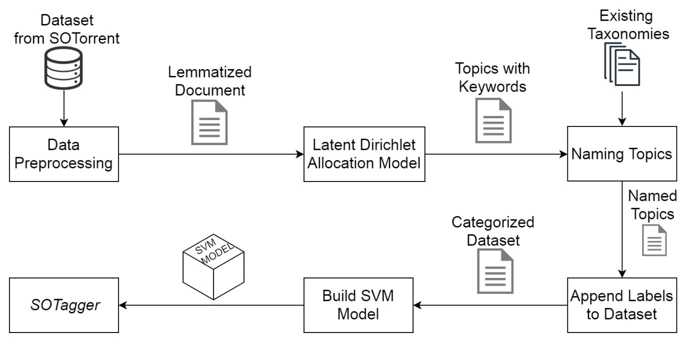


Fig. 1. Overview of Approach for *SOTagger*

IV. Design Methodology

We followed a six step approach in designing a contextual classification model as shown in Fig 1.

Step 1 - Extract DataSet. To perform categorization of SO posts, we downloaded *Posts.xml* file avilable on *SOTorrent*[5]. We considered a subset of this file that constituted 100K Stack Overflow posts under *Body* column and filtered out questions based on *PostTypeId* column that resulted in a dataset of 20K posts.

Step 2 - Data Preprocessing. Data present in *Body* column whose *PostTypeId = 1* was considered for preprocessing. We considered English stop words provided by NLTK library and omitted interrogative words from the list of stop words keeping in view, the taxonomy proposed. We processed the data for stop word, punctuation removal and lemmatization using *spaCy*.

[4]https://stackoverflow.com/help/mcve
[5]https://zenodo.org/record/2273117

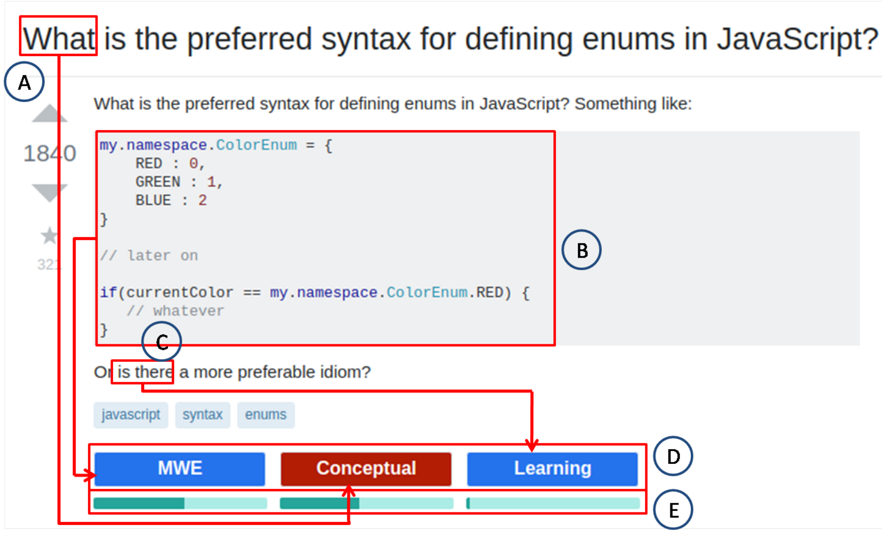


Fig. 2. A Snapshot of *SOTagger*

Step 3 - Latent Dirichlet Allocation Model. We applied LDA to perform topic modeling. We primarily created a dictionary of lemmatized words and then created a corpus of these words with their frequency of occurrence. Considering this corpus, we generated an LDA model that categorizes given data into 6 topics.

Step 4 - Naming Topics. Based on existing taxonomies in the literature [2] [1] [4] [3], we identified contextually useful keywords in each of the 6 topics, and used them to identify and name topics.

Step 5 - Append Labels to Dataset. The LDA model provided us with a topic-document correlation matrix, where document refers to content of one post. This matrix contained probabilities of every identified topic for each document. We then classified posts in the dataset into topics based on the dominant topic from correlation matrix which had the highest probability.

Step 6 - Prepare a Machine Learning model - Build SVM Model. We applied various machine learning classification algorithms such as *Linear SVC, Logistic Regression, Multinomial Naive Bayes, Random Forest Classifier* to arrive at the best classification model on available dataset with 75% train and 25% test data. We observed that SVC was able to classify the given data set with higher accuracy (78.5%) than other models. Based on this, we designed SVC model and pipelined to CalibratedClassifierCV to get prediction probabilities.

V. Development of *SOTagger*

This plug-in has been developed as an extension to *Google Chrome* to support classification of posts on SO. It tags posts on SO based on their context. *SOTagger* reads SO posts on the page and extracts questions from these posts which are fed into previously developed ML classification models using SVM classification. This model outputs the categories of specific posts along with associated probabilities which are presented as tags below the posts on SO platform.

A snapshot of *SOTagger* is shown in Fig 2 for a sample post on SO. Tags corresponding to context of the question are displayed below the post as shown in *[D]* of Fig 2 and are arranged in decreasing order of probability. The probability with which a post is tagged into each of the displayed categories is represented by a bar as depicted in *[E]* of Fig 2. According to *SOTagger*, this post is classified as *MWE* category with highest probability. As pointed in *[B]* of Fig 2, presence of code segment justifies classification of the post into *MWE* category. Presence of *What* keyword as highlighted in *[A]* of Fig 2, contributes to *Conceptual* tag, with a lesser probability than *MWE* tag. *is there* phrase represented by *[C]* of Fig 2 contributes to *Learning* category, with least probability.

However, the keywords or phrases demonstrated in Fig 2, are for the purpose of analyzing the correctness of *SOTagger*, but are not the only basis for classification. Actual classification was based on *NLP* and *ML* techniques that have been used in development of *SOTagger*.

VI. Evaluation and Results

We evaluated *SOTagger* by conducting a user survey with 32 professional developers with a development experience ranging from 2 years to 19 years.

The participants were asked to use *SOTagger*, navigate to SO website and analyze the contextual tags added by *SOTagger*. A user survey was conducted with the help of five point Likert scale, containing a questionnaire as provided in Table II.

Apart from user survey, we manually evaluated[6] contextual tags of about 100 random posts on SO tagged by *SOTagger* and obtained an accuracy of 77%. The

[6] https://git.io/fjC83

results of our survey indicate, *SOTagger* had a good user-friendly interface (82% in Q1). In Q2, about 85% of participants have agreed that *SOTagger* has appropriately tagged the posts. The ratings in Q3 and Q4 indicate that *SOTagger* has helped about 80% of participants in faster browsing of posts on SO and that the experiment has been considerably interesting (81% in Q4). In Q5, most of the participants have agreed that they would recommend *SOTagger* to their peers (83%). .

TABLE II
QUESTIONS IN SURVEY USING A 5-POINT LIKERT SCALE.

Q1: How easy was it to use *SOTagger* interface?

Q2: *SOTagger* has tagged SO posts correctly based on their context.

Q3: *SOTagger* has helped me in quick browsing of posts based on context.

Q4: *SOTagger* has kept the whole experiment interesting and informative.

Q5: I will recommend *SOTagger* to my peers.

VII. Threats to Validity

We have manually examined top 20 posts based on probability values in each of the 6 topics generated by *LDA* technique to assign topic name. This could be inaccurate considering limited number of posts examined.

To understand the accuracy of classification, we randomly browsed 100 posts on SO. We realize that examination of 100 posts in total is not enough to get an overall idea about the accuracy of classification. During the creation of LDA model, we tweaked a few parameters such as chunk size and number of passes which resulted in different statistical distribution of topics. Some of the distributions were imbalanced and biased towards one particular topic. We selected those parameters which resulted in a nearly Gaussian distribution. We assume that LDA model which classifies data in Gaussian distribution performs better than other models. However, initial results show that accuracy of trained LDA model is around 70%, but with scope for experimenting with other distributions. The machine learning model has been trained on a dataset of 20K questions, however we should consider a larger number of posts from SO to improve our approach.

VIII. Conclusion and Future Work

In this paper, we presented *SOTagger*, a prototype plug-in to SO that tags questions on SO based on the purpose for which they are asked. We performed LDA topic modeling on data set available on *SOTorrent* to identify categories. We labelled the resultant LDA topics by harmonizing the existing taxonomies. We presented 6 question categories, independent of technical aspects involved in the questions. We then labelled question posts in the dataset into one or more of the 6 categories. We applied SVC on the labelled dataset to obtain machine learning classification model which was integrated into the plug-in to support tagging of posts on SO.

As a part of future work, we plan to extend *SOTagger* to display contextual tags of posts on SO landing page by training machine learning model only over titles of questions. We plan to work in the direction to improve levels of taxonomy from single level presented in the paper to multiple levels and display the same as a part of detailed contextual tagging. We could conduct an experiment to check whether we get better results by considering the opening and closing statements of SO posts.

Questions tagged with *MWE* could be of greater use for future research. Researchers interested to understand and analyze code provided by users when posing questions can easily find questions with this tag. We envision that future work based on this paper may include clustering posts classified as *MWE* to automatically find bugs, combine co-occurring tags to formulate new tags and so on. Also, several empirical studies on SO posts such as understanding code quality, misuse of code snippets and automatic bug reporting could be conducted.

References

[1] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?: Nier track," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 804–807.

[2] S. Beyer, C. Macho, M. Pinzger, and M. Di Penta, "Automatically classifying posts into question categories on stack overflow," in *Proceedings of the 26th Conference on Program Comprehension*. ACM, 2018, pp. 211–221.

[3] M. Allamanis and C. Sutton, "Why, when, and what: analyzing stack overflow questions by topic, type, and code," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 53–56.

[4] C. Rosen and E. Shihab, "What are mobile developers asking about? a large scale study using stack overflow," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.

[5] R. K.-W. Lee and D. Lo, "Github and stack overflow: Analyzing developer interests across multiple social collaborative platforms," in *International Conference on Social Informatics*. Springer, 2017, pp. 245–256.

[6] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec ++: An enhanced tag recommendation system for software information sites," *Empirical Software Engineering*, vol. 23, no. 2, pp. 800–832, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9533-1

[7] X.-Y. Wang, X. Xia, and D. Lo, "Tagcombine: Recommending tags to contents in software information sites," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 1017–1035, 2015.

[8] A. Joorabchi, M. English, and A. E. Mahdi, "Text mining stackoverflow: An insight into challenges and subject-related difficulties faced by computer science learners," *Journal of Enterprise Information Management*, vol. 29, no. 2, pp. 255–275, 2016.

[9] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[10] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Evaluation and Usability of Programming Languages and Tools*. ACM, 2010, p. 8.

StackDoc - A Stack Overflow Plug-in for Novice Programmers that Integrates Q&A with API Examples

Akhila Sri Manasa Venigalla, Chaitanya S. Lakkundi, Vartika Agrahari, Sridhar Chimalakonda
Department of Computer Science and Engineering
Indian Institute of Technology
Tirupati, India
{cs18m017, cs18s502, cs18m016, ch}@iittp.ac.in

***Abstract*—There is a tremendous increase in the use of online coding platforms, courses and walkthrough tutorials to learn programming today. Stack Overflow, a Q&A website of crowd-sourced knowledge on programming is one of the popular platforms that developers and learners use to ask and answer Q&As related to programming. However, novice programmers often face difficulties in understanding the answers as they may contain new terminologies, function calls and attributes of certain technology or programming language. Researchers have proposed different ways to augment Stack Overflow in the literature, but to the best of our knowledge, there is no work that exists to augment Stack Overflow posts with definitions of API calls and relevant examples. To this end, we propose *StackDoc*, a prototype plug-in that augments Stack Overflow with definitions and examples of API calls in the questions and answers with the goal of helping novice programmers. We did a preliminary survey with 20 students of various universities, novice to Java and 85% of the users reported positive experience with the plugin.**

***Keywords*- Stack Overflow; API call; Novice Programmers; plug-in; Learning**

I. Introduction

Programming is considered as one of the fundamental skills in the 21st century. With the emergence of tremendous web content and novel technologies, one can learn programming through competitive programming or through online courses or with the help of Q&A sites. Developers today extensively rely on using code snippets and answers present Q&A websites like Stack Overflow [1]. It has been observed by researchers, that developers use Q&A sites such as Stack Overflow to clarify their doubts [2]. These questions on Stack Overflow could refer to debugging a code or adding new features to a given code [3] [2]. To raise their level of understanding on the code snippets provided on Stack Overflow, users generally search for definition, usage and importance of certain function calls that are used in the code snippets [3].

It was also observed that code examples are searched in Android API documentation by programmers [4]. However, most of the novice programmers might not be aware of all the functions and attributes used in code snippets, making code reuse difficult for them [5] [6] [7]. If these Q&A sites can serve as knowledge reserves, beginners might be able to understand how to solve a problem and purpose of using different function calls. This analysis of existing literature presents the need to support novice programmers with additional information to effectively use Q&A websites. Also, we observed that tools such as *ExampleCheck* [8] have been developed to augment Stack Overflow to support developers by reporting incorrect usage of APIs. However, to the best of our knowledge, existing work did not focus on providing information about API calls on Stack Overflow for novice programmers, motivating the need for our work. In this paper, we propose to augment Stack Overflow with *definition*, *usage* and *examples* for certain inbuilt API calls of *Java* programming language.

StackDoc[1] is a Stack Overflow prototype plug-in to help beginners for simple and rapid learning. In this preliminary version of our plug-in, a pop-up consisting of Java API calls information is shown on the webpage to the users, instead of searching on other sources for API definitions. We also conducted a survey to evaluate our plug-in and received satisfactory results with feedback of about 85% users recommending the plug-in to their peers. Fig 1 shows a high-level overview of *StackDoc*. We extract API definitions from standard Java documentation, OpenJDK 10[2] and to demonstrate our idea, we use JExamples[3] to extract API examples. This extracted information is displayed to the user on Stack Overflow. The essence of the plug-in is to extract examples and information about a certain API call from multiple sources and present this information to novice programmers.

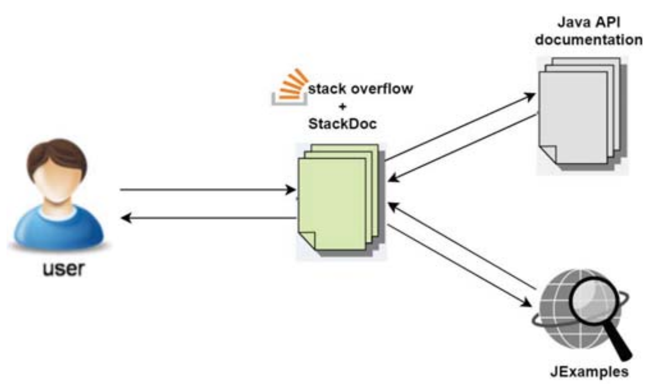


Fig. 1. Overview of *StackDoc*

The remainder of this paper is structured as follows. Section II discusses the related work followed by Section III, which focuses on design methodology and development of *StackDoc*.

[1]https://github.com/AkhilaSriManasa/StackDoc
[2]http://cr.openjdk.java.net/~iris/se/10/latestSpec/api/
[3]http://www.jexamples.com

2161-377X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICALT.2019.00077

We present the evaluation and user survey results in Section IV and Section V. Finally, we discuss the limitations in Section VI and end the paper with conclusions in Section VII and future directions in Section VIII.

II. Related Work

The number of code snippets on websites such as Stack Overflow are increasing dramatically and most of these snippets use many API calls [9]. There are various development, debugging and learning tools to support novice programmers (as shown in Table I). *Scratch* supports beginners to learn programming by designing, creating and remixing code blocks [10]. *Alice* was developed to support novice programmers learn basic concepts of programming through 3D visualization [11]. Treude et al. have augmented API documentation with insights from Stack Overflow [12]. Tools such as *Prompter* have been developed to support users with discussions on Stack Overflow, based on the context of code in an Eclipse IDE [13]. *Exemplar* provides applications relevant to queried APIs entered by users in the tool [14]. The relevant applications are retrieved using information retrieval and program analysis techniques [14]. In *Examplore*, code snippets have been bound with the examples of API usage and definitions of API calls [15]. *Examplore* produces an interactive visualization to view general usage patterns of an API call in a code snippet [15]. *Blueprint* integrates source code examples into Adobe Flex Builder, a development environment, resulting in faster example code search [16]. Zhang et al. augmented Stack Overflow with API misuse warnings through a Google Chrome plug-in, *ExampleCheck* [8]. It displays a pop-up with API misuse alert and a suggestion to fix these misuses by providing curated examples that use the specific API call correctly [8].

TABLE I
Related Work

S.No.	Domains	References
1	Development	Examplore [15]
		Exemplar [14], Prompter [13]
		Blueprint [16]
2	Debugging	[17], [18], ExampleCheck [8]
3	Learning	Scratch [10], Alice [11]

Although *ExampleCheck* provides examples for correct usage of APIs, it does not provide definitions and examples of any other APIs used in the code snippets. *StackDoc* displays definitions and examples of all identified in-built API calls. It can thus serve as a unified solution, as it reduces the effort of searching for API usage and definitions explicitly. *Exemplar* provides a search interface to retrieve examples for queried API calls, whereas, *StackDoc* integrates API definitions and API examples inline with the code snippets in java.

III. Design and Development of *StackDoc*

We have developed *StackDoc* as a browser extension and tested on Google Chrome and Mozilla Firefox browsers. *StackDoc* facilitates novice programmers by parsing the Stack Overflow webpage and retrieving documentation for identified API calls from online sources such as OpenJDK and JExamples. These API calls are highlighted, for which, documentation and relevant example usages are displayed in a popup to the user.

This could help users learn about many new API calls that the user might be unaware of. This design methodology of *StackDoc* makes it distinct from existing approaches which display definitions and examples after an explicit query from the user. Whereas, we rely on standard Java documentation, OpenJDK, to retrieve API definitions and JExamples.

Fig2 shows the development process of *StackDoc* consisting of 5 steps.

In the first Step, we create a regular expression to identify function calls in Java. The regular expression we used to identify API calls is given below.

```
(([a-zA-Z$_]+[a-zA-Z0-9$_]*)\.)?
[a-zA-Z$_]+[a-zA-Z0-9$_]*\(
```

The defined regular expression (regex) accounts for the fact that API calls can be made using a class name directly or using an instantiated object. If the function is called directly without specifying any variable or class name, then the regex will still identify the function call. The defined regex identifies API calls of the form <variable>.<function_call>, <class_name>.<function_call>and <function_call>.

In Step 2, *StackDoc* generates a list of keywords using the regex defined previously. These keywords are matched with the existing corpus of class names and API calls of Java 10 specification. *StackDoc* then stores all the identified API calls into an array for further processing.

In the third Step, for every identified API call, we search its documentation in OpenJDK. Initially, we search the combination of <variable or class name>.<function_call>directly in the documentation. If a matching combination is found, then its documentation comprising of the arguments and definition is fetched. If any matching combination is not found, we search OpenJDK documentation only for the <function_call>. Once a matching API call is found, it is highlighted and made interactive so that the user will be able to click on it.

As a part of Step 4, when any of these highlighted API calls are clicked, corresponding usage examples for the particular API call are fetched from JExamples website.

Finally, in Step 5, the documentation of API calls and their usage examples are concatenated and displayed as an overlay. The overlay popup is hidden by default and becomes visible only when the user clicks on any highlighted API call. For every API usage example, a link to JExamples is provided from where the complete example can be viewed.

IV. Evaluation

We conducted a study to evaluate the usefulness of *StackDoc*. We aimed at assessing the extent to which *StackDoc* could be helpful to novice programmers in understanding and implementing API calls and data types in Java. Hence, we considered 20 university students novice to Java. The study

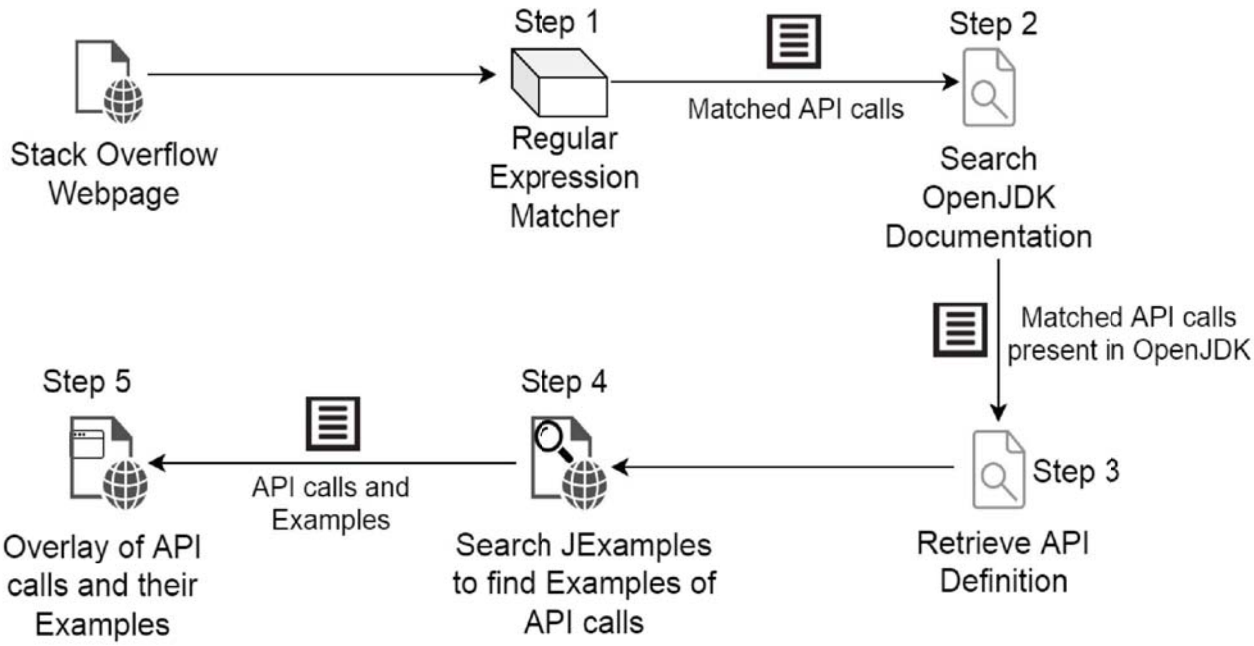


Fig. 2. Design Methodology of StackDoc

was performed with the help of a questionnaire based on Likert scale [19], on personal laptops of students.

A. Procedure

All the participants were requested to add *StackDoc* extension to browsers on their laptops. They were all provided with a slide-show depicting the working of *StackDoc*, that served as a basic tutorial. These 20 participants were then asked to search for questions related to Java programming language and go through at least 20 questions and answers that they have retrieved as a result of their search. They were requested to use *StackDoc* to clarify their doubts about usage and definitions of API calls that might arise in the process of their observation.

After completion of the above exercise, participants answered a questionnaire using a 5-point Likert scale. Questionnaire given to the participants is as shown in Table II.

B. User Scenario

Suppose *Veda* is a novice programmer working on Java and wishes to know why sorted arrays are processed faster than unsorted arrays, for which she queries on Stack Overflow.

She is then displayed with a list of posts related to this query as shown in [A] of Fig 3. She randomly selects one of the displayed posts. Once a post is selected, Java API calls present in code snippets of the post are highlighted ([B] of Fig 3). Among the answers displayed, *Veda* encounters API calls such as *System.println(), System.nanoTime(), Arrays.sort(), Random.nextInt()* and wishes to know their definitions and usage. *Veda* clicks on *System.println()* in the code snippet. She is then displayed with a pop-up containing definition and examples of *System.println()* as shown in [C] of Fig 3.

If *Veda* navigates to another example containing *log* and clicks on the API call, she will be able to view description and usage of *log(Level level, Supplier <String>msgSupplier)* (as represented in part [D] of Fig 3).

V. Results

A. Questionnaire

As reported in Fig 4, *StackDoc* had a good user-friendly interface (84% in Q1). In Q2, participants have agreed that *StackDoc* retrieved moderately sufficient number of examples to understand the API usage (83%, about 10 participants voted for *Agree* and 5 participants voted for *Strongly Agree* out of 20). The ratings in Q3 and Q4 indicate that *StackDoc* has helped participants learn about partly unaware API calls, reducing the search time (73% in Q3 and 79% in Q4). Participants have also suggested to improvise *StackDoc* to support other languages and to provide descriptions to a wider range of API calls. In Q5, most of the participants have agreed that they would recommend *StackDoc* to their peers (83%).

VI. Limitations

We presented a prototype of *StackDoc*, that augments Stack Overflow by helping programmers to learn about Java API calls. However, our plug-in could be improved in multiple ways in future versions. Currently, *StackDoc* shows API definitions only for Java, restricting its application to one programming language. Also, we were not able to find API definitions for few APIs as they do not exist in OpenJDK 10 or might have been deprecated. Similarly, the limited availability of examples on JExamples website limits *StackDoc* to retrieve examples for few of the desired APIs.

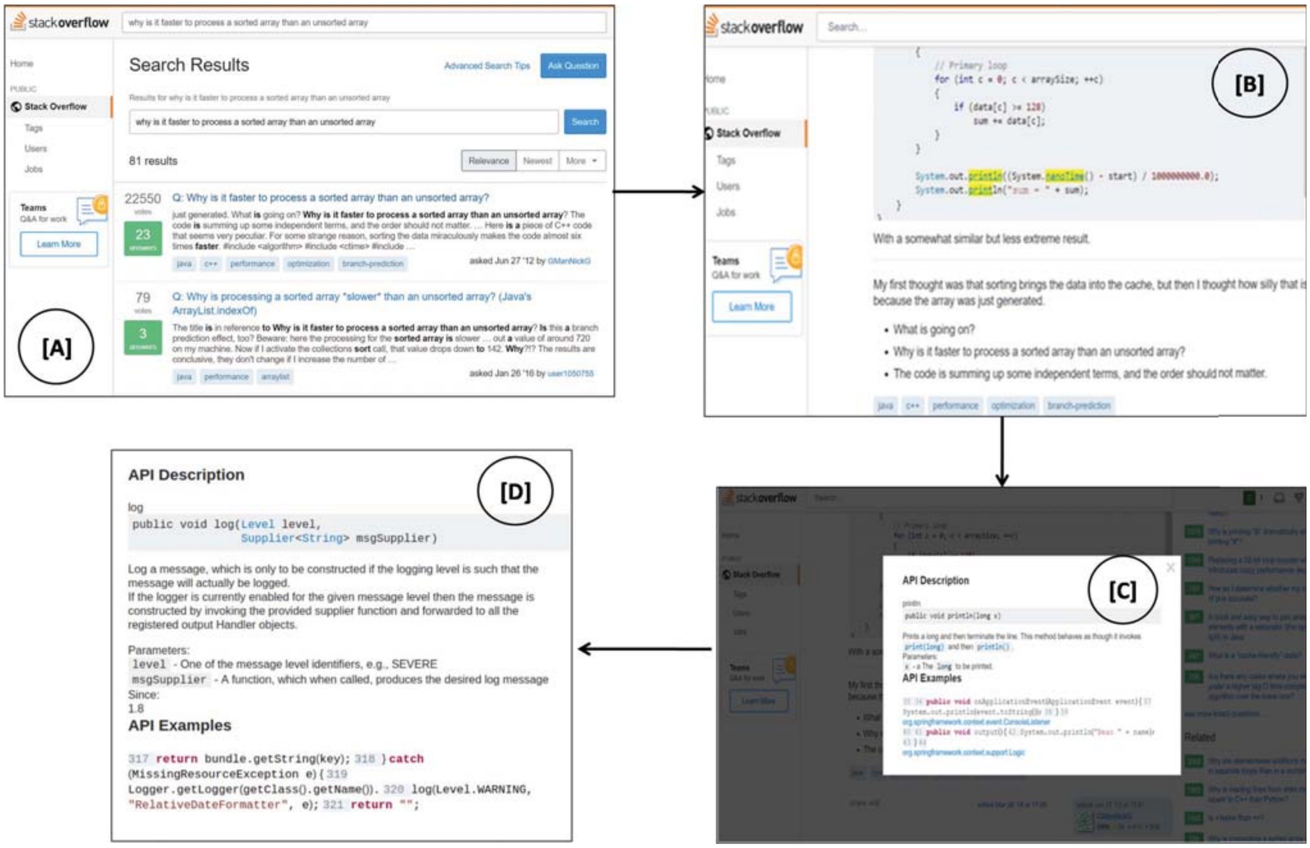


Fig. 3. Example user scenario of *StackDoc* by user *Veda*; A: Search question on Stack Overflow; B: API calls highlighted by *StackDoc*; C: Description of API call with Example as given by *StackDoc*; D: API Description for another example *StackDoc*

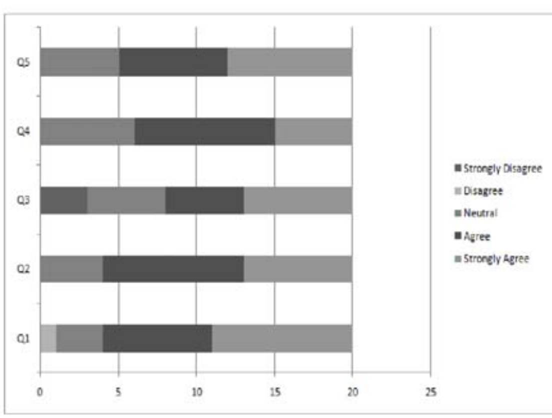


Fig. 4. Results of Questionnaire

In its current version, there is a delay between the user asking for API definitions and *StackDoc* retrieving it from the online sources, OpenJDK 10 and JExamples. The reason of this delay being that *StackDoc* fetches required details only after the user clicks on highlighted API call.

VII. Conclusions

We emphasized the need to support novice programmers when they browse Q&A websites like Stack Overflow. Hence, we have introduced *StackDoc*, a browser plug-in that augments Stack Overflow. Our tool is an initial step to support novice programmers with better mechanisms. We have extracted documentation of java API calls from OpenJDK 10 and examples of these API calls from JExamples website. These API calls are highlighted on Stack Overflow page and the extracted information is displayed to the user as a pop-up when clicked on the highlighted API calls. Our initial user study indicated that *StackDoc* has helped users in finding definitions, but with the need to have more examples. We plan to extend *StackDoc* to support a larger number of programming languages and multiple data sources. We plan to do an extended study with 50 users and incorporate changes to improve our plug-in.

VIII. Future Research Directions

Beyond *StackDoc*, our core idea is to support software engineers by integrating documentation with software development platforms. Here are a few potential future directions:

TABLE II
QUESTIONS IN SURVEY USING A 5-POINT LIKERT SCALE.

Q1: How easy was it to use the plug-in interface?(1=very easy, 5=very difficult)
Q2: *StackDoc* has retrieved enough number of examples to understand a particular API usage. (1=strongly agree, 5=strongly disagree)
Q3: *StackDoc* has helped me in learning about API calls that I wasn't aware prior to this exercise. (1=strongly agree, 5=strongly disagree)
Q4: *StackDoc* has made my learning quicker by reducing my search time. (1=strongly agree, 5=strongly disagree)
Q5: I will recommend *StackDoc* to my peers. (1=strongly agree, 5=strongly disagree)

A. Code hosting platforms

For example, *Github* could be augmented with appropriate documentation such as API information to help developers, especially novice developers to understand source code repositories.

B. Documentation for System Administrators

Documentation is critical in system administration tasks, but is often missed as the focus is only on executing the tasks, making it difficult for novice system administrators. We see that developing a plugin to support novice system administrators based on *StackDoc* is a valuable future direction.

C. Algorithm Documentation

Understanding code snippet might be difficult than understanding an algorithm for a novice programmer. If an algorithm is available or if an abstraction can be created, code snippets could be integrated with the algorithm.

D. Deployment Scenarios

While deploying an application, in case of failures, it is difficult to identify code location of failed modules and make necessary changes. Each module could be integrated with deployment document containing code repository information.

IX. ACKNOWLEDGEMENT

We thank all the participants for their valuable time and feedback that helped us in evaluating *StackDoc*.

REFERENCES

[1] Yang, Di and Martins, Pedro and Saini, Vaibhav and Lopes, Cristina, "Stack overflow in github: any snippets there?" in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 280–290.

[2] R. Garcia, K. Falkner, and R. Vivian, "Systematic literature review: Self-Regulated Learning strategies using e-learning tools for Computer Science," *Computers & Education*, vol. 123, pp. 150 – 163, 2018.

[3] Ko, Andrew J and Myers, Brad A and Aung, Htet Htet, "Six learning barriers in end-user programming systems," in *2004 IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 2004, pp. 199–206.

[4] J. E. Montandon and H. Borges and D. Felix and M. T. Valente, "Documenting APIs with examples: Lessons learned with the APIMiner platform," in *2013 20th Working Conference on Reverse Engineering (WCRE)*, Oct 2013, pp. 401–408.

[5] Lahtinen, Essi and Ala-Mutka, Kirsti and Järvinen, Hannu-Matti, "A Study of the Difficulties of Novice Programmers," in *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '05. New York, NY, USA: ACM, 2005, pp. 14–18.

[6] Bosse, Yorah and Gerosa, Marco Aurelio, "Why is programming so difficult to learn?: Patterns of Difficulties Related to Programming Learning Mid-Stage," *ACM SIGSOFT Software Engineering Notes*, vol. 41, pp. 1–6, 01 2017.

[7] Joorabchi, Arash and English, Michael and Mahdi, A.E., "Text mining stackoverflow: Towards an Insight into Challenges and Subject-Related Difficulties Faced by Computer Science Learners," *Journal of Enterprise Information Management*, vol. 29, pp. 255–275, 03 2016.

[8] Zhang, Tianyi and Upadhyaya, Ganesha and Reinhardt, Anastasia and Rajan, Hridesh and Kim, Miryung, "Are code examples on an online Q&A forum reliable?: a study of API misuse on stack overflow," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 886–896.

[9] Subramanian, Siddharth and Holmes, Reid, "Making sense of online code snippets," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 85–88.

[10] Resnick, Mitchel and Maloney, John and Monroy-Hernández, Andrés and Rusk, Natalie and Eastmond, Evelyn and Brennan, Karen and Millner, Amon and Rosenbaum, Eric and Silver, Jay and Silverman, Brian and others, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[11] Cooper, Stephen and Dann, Wanda and Pausch, Randy, "Alice: a 3-D tool for introductory programming concepts," *Journal of Computing Sciences in Colleges*, vol. 15, no. 5, pp. 107–116, 2000.

[12] Treude, Christoph and Robillard, Martin P., "Augmenting API Documentation with Insights from Stack Overflow," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 392–403.

[13] Ponzanelli, Luca and Bavota, Gabriele and Di Penta, Massimiliano and Oliveto, Rocco and Lanza, Michele, "Mining StackOverflow to turn the IDE into a self-confident programming prompter," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 102–111.

[14] Grechanik, Mark and Fu, Chen and Xie, Qing and McMillan, Collin and Poshyvanyk, Denys and Cumby, Chad, "A search engine for finding highly relevant applications," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 475–484.

[15] E. L. Glassman, T. Zhang, B. Hartmann, and M. Kim, "Visualizing API usage examples at scale," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, 2018, p. 580.

[16] Brandt, Joel and Dontcheva, Mira and Weskamp, Marcos and Klemmer, Scott R, "Example-centric programming: integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 513–522.

[17] Spinellis, Diomidis, "Modern debugging: the art of finding a needle in a haystack," *Communications of the ACM*, vol. 61, no. 11, pp. 124–134, 2018.

[18] Torroja, Yago and López, Alejandro and Portilla, Jorge and Riesgo, Teresa, "A serial port based debugging tool to improve learning with arduino," in *Design of Circuits and Integrated Systems (DCIS), 2015 Conference on*. IEEE, 2015, pp. 1–4.

[19] Likert, Rensis, "A technique for the measurement of attitudes." *Archives of psychology*, 1932.