Decreasing the Barrier to Entry for an Open-Source Full-Stack Web Development

Clark Jason Ngo
clarkngo@cityuniversity.edu
eBay Inc.

Jin Chang
kyongchang@cityu.edu

Sam Chung
chungsam@cityu.edu

School of Technology & Computing
City University of Seattle
Seattle WA

Abstract

The purpose of this paper is to propose how to decrease the barrier to entry for open-source projects and full-stack web development. For this purpose, we conduct documentation and architectural modeling to lessen the steep learning curve for open source and full-stack web development. The research will bring in ten hands-on practices (HOPs) to teach students to build a full-stack web application. The research will conduct the following: 1) Create documentation and architectural model for the hands-on practice. 2) Evaluate the documentation and architectural model with a survey. Then, we evaluate the impact of the documentation and architectural diagram in decreasing the learning curve of full-stack development through a survey. The survey results show that implementing architectural modeling in the documentation of an open-source full-stack development reduces the learning curve for the developers because it gives a visualization that the developer can easily follow and digest the high-level concept.

Keywords: Software Documentation, Software Architecture, Open Source, Full Stack, MEAN

1. PROBLEM AND MOTIVATION

Open-source software invites anyone with the technical capability to contribute to the code base. However, most open-source software has a significant barrier to entry, as there is no support for visual architectural modeling, which involves diagrams to describe components of a system with the aim to provide high level overview of the structure (Kim, Chung, & Endicott-Popovsky, 2014).

Open-source software is very complex. It has a steep learning curve to be able to contribute or use it. The non-existence of documentation leads to further complexity. Full-stack web development is challenging to learn as well (Shah, & Soomro, 2017).

Aghajani et al. (2019) showed issues in software documentation. They surveyed a sample size of 955 document issues, including StackOverflow, GitHub, and mailing lists. 88% (840 issues) were GitHub issues and pull requests. Half (426 issues) of the 88% were information content: "what." Fifty-three percent (227 issues) were about missing or insufficient documents and missing

diagrams. Breakdown of the document issues shows: 50% includes documentation and diagram (What), 27% on Information Content (How), 8% on Process Related, and 14% on Tool Related.

Also, there is a steep learning curve from full-stack development with MEAN (Mongo DB, Express, Angular, & Node.js) Stack. Shah and Soomro (2017) showed difficulties in learning Node.js, server-side component of MEAN stack. They surveyed a sample size of 80 developers to answer survey questions. The following shows the questions and the results: (1) Learning of JavaScript for Node.js was a challenge: 23.9% felt a learning challenge, and 44.8% felt learning a little bit of challenge. (2) Learning JavaScript for NoSQL Databases was a challenge: 31.3% felt a learning challenge, and 20.9% felt learning a little bit of challenge. (3) Event-Driven of Node.js challenging: 34.3% felt a learning challenge, and 25.4% felt learning a bit of challenge. (4) Non-Blocking I/O feature of Node.js was challenging: 31.3% felt a learning challenge, and 26.9% felt learning a little bit of challenge. (5) Asynchronous processing feature of Node.js was challenging: 38.8% felt a learning challenge, and 17.9% felt learning a bit of challenge.

This paper challenges reducing the steep learning curve from both open-source and full-stack development with MEAN Stack.

2. BACKGROUND

Architectural Model

Unified Modeling Language (UML) notation is a language that uses pseudo-code, actual code, pictures, diagrams, and others to help describe systems. However, as UML is an abstraction to provide a high-level overview, the notation will not describe the small details. Still, UML is better than detail overloading from modeling with code and ambiguity from modeling with informal language (Miles & Hamilton, 2006).

UML Documentation improves software maintenance. With UML providing a high-level overview and structure, developers can understand software architecture easily. On the contrary, building UML documentation costs extra time to make and is not useful for easy and small tasks to fix (Arisholm, Briand, Hove & Labiche, 2006).

Full Stack Web Development - MEAN Stack

MEAN stack consists of four technologies used together. These are MongoDB (Data Tier), Express (back-end web framework), Angular (front-end web framework), and Node.js (back-end environment). MEAN stack is a technology that uses JavaScript across all components removing the need for translation and saves build time (Dunka, Emmanuel, & Oyerinde, 2018). See Figure 1 for a diagram on client-side, server-side, and database using JavaScript.

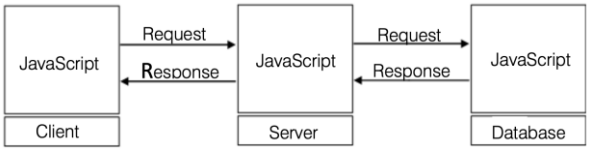

Figure 1. End-to-end JavaScript. Adopted from "Node.js Challenges in Implementation" by Shah, H., & Soomro, T. (2017).

RESTful APIs Design

Following the Representational State Transfer (REST) Application Programming Interface (API) in an application allows us to conform to the web standards to easily establish a connection between the information provider and user. By applying REST principles, the constraints applied on the API make communication interaction simple and system modular. Criteria in RESTful API are client-server architecture, stateless client-server communication, cacheable data, uniform interface, layered design, and code-on-demand (Kulkarni & Takalikar, 2018).

Back-end with Node.js

Node.js is built for handling asynchronous I/O while JavaScript has an event loop built-in for the client-side. See Figure 2 for Node.js processing model.

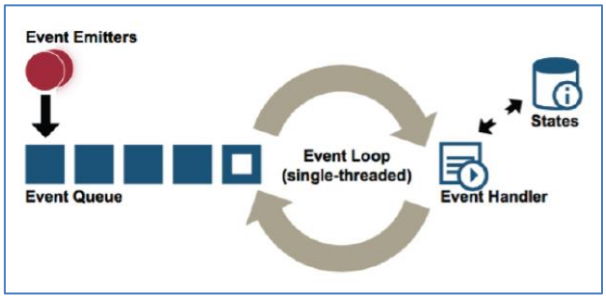

Figure 2. Node.js Processing Model. Adopted from "After much research and photoshop, I am proud to give you the finalized #nodejs System diagram" Esostalgic. (31, July, 2014).

The event-driven/callback approach makes Node.js fast in performance compared to other environments. However, this approach makes Node.js challenging to debug and learn as well. See Figure 3 for Node.js System.

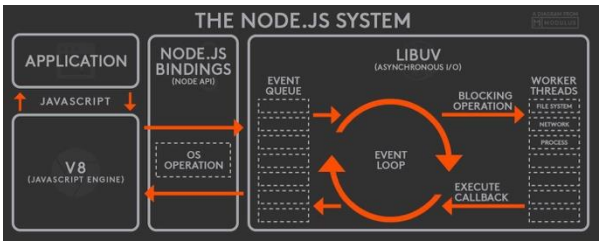

Figure 3. Node.js System. Adopted from "After much research and photoshop, I am proud to give you the finalized #nodejs System diagram" Esostalgic. (31, July, 2014).

Node.js includes modules such as mongoose, which is a MongoDB object modeling, and express web application framework. Through node modules, abstraction can be achieved, which reduces the overall complexity of the MEAN stack.

Back-end with Express Framework

Express is a minimalist and unopinionated application framework for Node.js. It is a layer on top of Node.js that is feature-rich for web and mobile development without hiding any Node.js functionalities (Adhikari, 2016).

Front-end with Angular

Angular is a web development platform built-in TypeScript that provides developers with robust tools for creating the client-side of web applications. It allows the development of single-page web applications where content changes dynamically based on user behavior and preferences. It features dependency injections to ensure whenever a component is changed, other components related to it will be changed automatically. Figure 4 shows the MVC (Model View Controller) architecture (Adhikari, 2016).

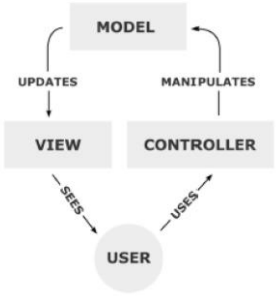

Figure 4. Model View Controller. Adopted from "Node.js Challenges in Implementation" by Shah, H., & Soomro, T. (2017).

Database with MongoDB

MongoDB is a NoSQL database that stores data in Binary JavaScript Object Notation (BJSON). MongoDB became the de facto standard database for Node.js applications to fulfill the JavaScript everywhere using JavaScript Object Notation (JSON) to transmit data across different tiers (front-end, back-end, and database) (Adhikari, 2016). NoSQL database relies on the BASE (Basically Available, Soft state, Eventual consistency). The Basically Available concept guarantees the availability of data, The Soft state concept allows a state of a system to change over time, and the eventually consistency concept means the system will be consistent after it stops receiving input. In Brewer's theorem (Figure 5), MongoDB satisfies CP in CAP (Consistency, Availability, and Partition-tolerance). If the data requirements are changing over time, big data, and cannot lock in on a schema, a NoSQL database would be the preferred choice (Pore & Pawar, 2015).

Learning Curve

When we learn, we acquire new knowledge and skill. The rate of knowledge or skill acquisition is called the learning curve (Kang & Hahn, 2009). Learning is defined as obtaining knowledge through studying and experiencing a subject matter. The three core foundations of learning are knowledge, skill, and competence (Britto, Šmite, & Damm, 2016). Knowledge is information processed and absorbed through structured or unstructured learning. Skills are acquired through practice. Competence is the effectiveness of an individual in a specific field. Reducing a steep learning curve increases the return-on-investment (ROI) for software development as it will lead to the faster building of new features and fixing or bugs. It also increases the productivity of developers who mentor (Tüzün & Tekinerdogan, 2015).

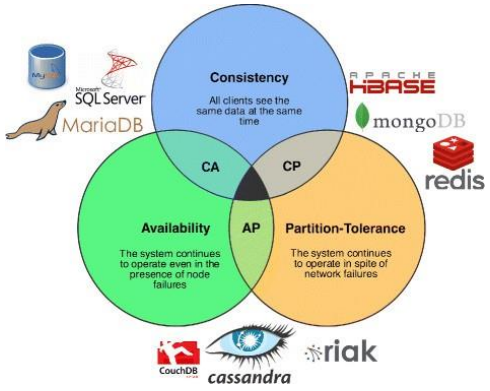

Figure 5. Brewer's CAP Theorem. Adopted from "CAP Theorem. Medium" by Singh, V. K. (2019).

The learning curve becomes steep when the topic to be learned is complex, such as software architecture that has challenges on understanding the theory behind it (design principles, tradeoffs, architectural patterns, etc.),

visibility at scale only, and requiring communication between different stakeholders (Van Deursen et al., 2017). The following principles were introduced to address the challenges: (1) embrace open source, (2) embrace collaboration, (3) embrace open learning, (4) interact with architects, and (5) combine breadth and depth. The approach for tackling challenges was to (1) apply theory to practice, (2) contribute to the system, (3) integrate architectural views, and (4) providing feedback to other students. Additionally, reducing the steep learning curve can be done through mentoring. The use of markdown was implemented to facilitate sharing, versioning, and reviewing various systems.

Another study found another way to reduce the steep learning curve was by pairing developers with mentors. The study showed that an Open Source Project (OS) activity with mentored developers was significantly higher than non-mentored developers (Fagerholm & Sanchez, 2014).

Documentation

Documentation helps define what a project does, why the project is valuable, how users can start using the project, where users can get help, and defines who maintains and contributes to the project (Prana et al., 2018). Why is documentation necessary? There are dangers of internet search instead of looking at software documentation. Users might use the wrong information. Software developers are likely to build poor software quality built due to a lack of good documentation (Loggem, 2014).

Documentation on Open Source

In open-source software, documentation is mainly created using a markdown file called README.md. README is the custom standard platform used for software distribution. This markdown file is analogous to a website's home page. If the README.md is poorly written and maintained, developers would not be attracted to try out the technology. See Figure 6 for a comparison of the README.md file of Node.js in GitHub (Node.js, 2020) and Amazon Web Services (AWS) home page (Amazon Web Services, 2020).

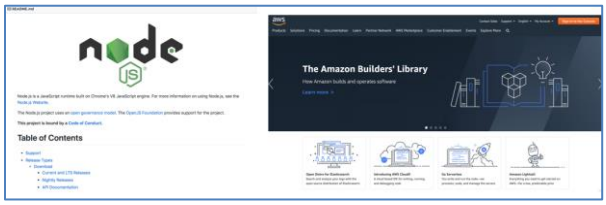

Figure 6. Open Source README.md versus Website Homepage

Amateur developers write documentation last. Most software has no set end date that it will be fully built and functional. Developers might forget to write documentation or have written it late because it is not as valuable anymore. However, professional developers write documentation first using Test-Driven Development or Behavior-Driven Development (Mastropasqua, 2016). Early and timely documentation attracts new developers to collaborate on open-source projects and eases the onboarding process.

According to Lee (2018), best practices for creating and maintaining software documentation are the following: (1) Include a README or text file as it is more human-readable than HyperText Markup Language (HTML). (2) Use a style guide to tell users how to use it. (3) Include a help command for developers without a Graphical User Interface (GUI). (4) Provide a link to the complete documentation to avoid cluttering the README file. (5) Apply version control to support versions of the project.

API Documentation

Documentation is not limited to system architecture, adding a documentation for API endpoints and parameters allows developers to quickly comprehend the resources available in a service. The OpenAPI specification defines the standard for documenting RESTful web services. Swagger UI is an open-source software that follows the specification and self-generates interactive interface of API endpoints. (Koren & Klamma, 2018). See Figure 7 for API endpoint user interface.

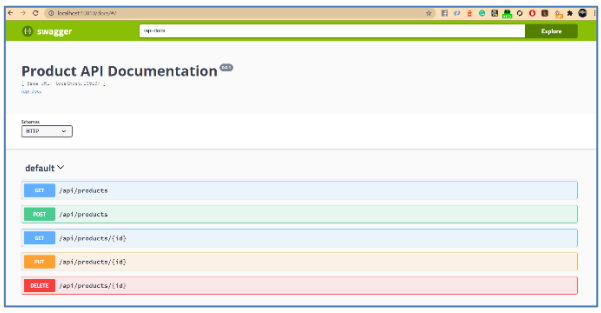

Figure 7. Swagger User Interface

3. RELATED WORK

The key papers are compared with the availability of the topics: (1) documentation, (2) architecture, (3) open-source, and (4) full-stack. See Table 1 for a summary of key papers.

Topic	[KRUC 95]	[KIM 14]	[ADMI 17]	[STAF 15]
Documentation	No	Yes	No	No
Architecture [4+1 View]	Yes	Yes	Yes	Yes
Open Source	Yes	Yes	Yes	No
Full Stack[MEAN]	No	No	No	No

Table 1. Key Topics for Previous Work

The 4 + 1 View Model of Software Architecture

Krutchen (1995) proposed an architectural model using the 4 + 1 views, consisting of the Logical View, Development View, Process View, Physical View, and Scenarios. The Logical View supports functional requirements. The Development View shows the static organization of software in the development environment. The Process View shows the flow of synchronization in the design. The Physical View shows the connection of software to hardware. Scenarios put the 4 views together and validate them through the perspective of the end-user. See Figure 8 for 4 + 1 View Model.

Software Architecture Model Driven Reverse Engineering Approach to Open Source Software Development

Kim, Chung, and Endicott-Popovsky (2014) designed architectural models for deployment view for MITREid Connect, an open-source authentication protocol. See Figure 9 for Deployment View. They also developed a design view to show an overview of classes, object instances, and message exchanges. See Figure 10 for Design View.

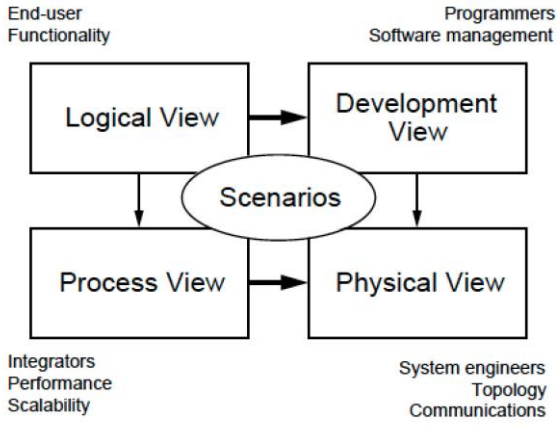

Figure 8. The 4 + 1 View Model. Adopted from "The 4+1 View Model of architecture" by Kruchten, P.B. (1995). IEEE Software. 12(6). 42 –50.

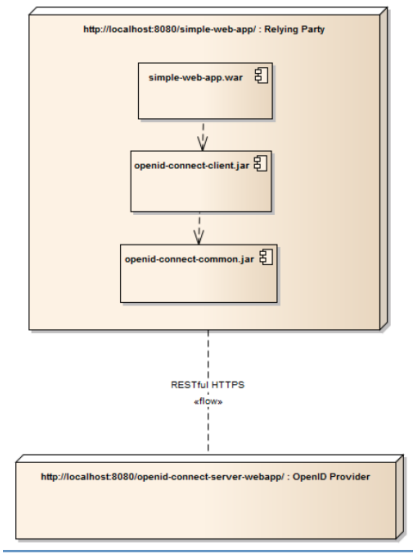

Figure 9. Deployment View. Adopted from "Software Architecture Model Driven Reverse Engineering Approach to Open Source Software Development" by Kim, W., Chung, S., & Endicott-Popovsky, B. (2014). Proceedings of the 3rd annual conference on Research in Information Technology.

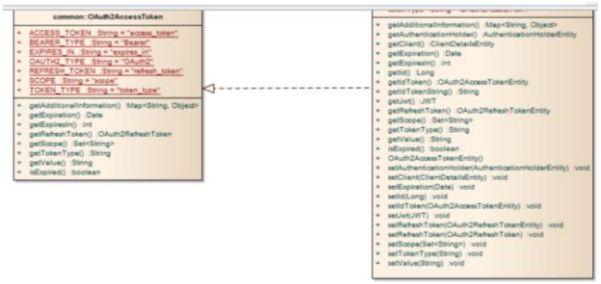

Figure 10. Design View. Figure 9. Deployment View. Adopted from "Software Architecture Model Driven Reverse Engineering Approach to Open Source Software Development" by Kim, W., Chung, S., & Endicott-Popovsky, B. (2014). Proceedings of the 3rd annual conference on Research in Information Technology.

Technical Design for Angular Apps

Admiraal (2017) designed an architectural diagram for Angular applications with UML notation and color coding for the package diagram to provide an overview of Angular modules and their dependencies to each other. See Figure 11 for the package diagram.

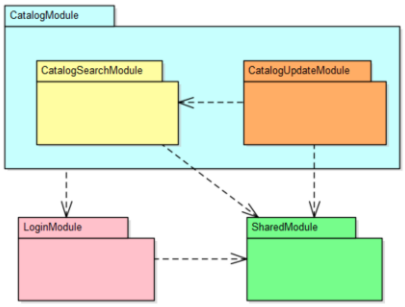

Figure 11. Package Diagram of Angular Modules. Adopted from "Technical Design in UML for Angular Applications" by Admiraal, H. (2017).

He also designed a scenario-based sequence diagram to show the activity flow and color-coded based on the Angular modules. See Figure 12 for the Use Case View example.

Stafford (2015) showed a request and request data flow sequence diagram. The diagram showed both the request and response across the MEAN Stack that includes the front-end, back-end, and database.

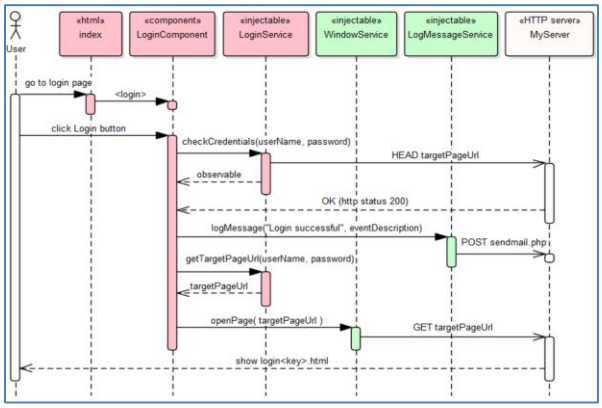

Figure 12. Scenario of User Login. Adopted from "Technical Design in UML for Angular Applications" by Admiraal, H. (2017).

4. APPROACH AND UNIQUENESS

This author's approach was the combined implementation of Kruchten's 4+1 View, Kim's Design View and experiment validation, and Admiraal's color-coded diagrams. The approach brought in in ten hands-on practices (HOPs)to teach students to build a full-stack web application. Check See Table 2 for the author's approach in the [TAA] column.

Topic	[KRUC 95]	[KIM 14]	[ADMI 17]	[STAF1 5]	**[TAA]**
Documen tation	No	Yes	No	No	Yes
Architecture [4+1 View]	Yes	Yes	Yes	Yes	Yes
Open Source	Yes	Yes	Yes	No	Yes
Full Stack [MEAN]	No	No	No	No	Yes

Table 2. Author's Approach in the column [TAA]

4.1 Experimental Design and Methods

We choose a series of examples from "Getting MEAN with Mongo, Express, Angular, and Node" by Holmes, S. and Harber, C. (2019), Manning Publications. This book teaches us how to develop full-stack web applications using the MEAN stack through many examples. Also, we used the online diagram tool, Lucidchart, to build the sequence, package, and class diagrams from scratch. You can see source codes, and diagrams from https://github.com/clarkngo/cityu_capstone.

Physical View with Deployment Diagram

The deployment diagram shows 3 servers: front-end, back-end, and database. In the front-end, we require the browser as angular applications are browser-based web applications. The back-end server hosts our Node.js with Express on top of Node.js. In Express, we have the application and mongoose on top of it. Express will handle the communication on both the front-end and database. The database server only includes MongoDB. JSON is utilized for communicating across servers.

Process View with Sequence Diagram

When an HTTP request is made, the front-end will be triggered, and Angular will pick up the request. The request will be passed internally in Angular with Route sending a request for the view to View/Template. View/Template will request the Controller. The Controller will then create an HTTP request to a Representational state transfer (RESTful) endpoint to the Server Side, Express/Node.js. The request will then be passed internally with Express/Node.js from its Route to the Controller/Model. The Controller/Model will request the Mongoose Object-Document Mapping (ODM) to interact with the Database Server MongoDB. MongoDB will process the request and respond to the callback to Express/Node.js. Express/Node.js sends a JSON response to the Angular Controller. Angular Controller would respond with a view.

Development View with Package Diagram

In a book store application example, the package diagram shows the dependency of each Angular module to other Angular modules. These modules are CatalogModule, which includes CatalogSearchModule and CatalogUpdateModule, LoginModule, and SharedModule in Figure 13.

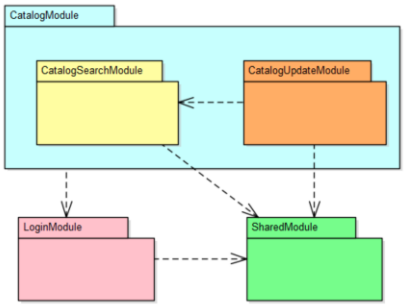

Figure 13. Package Diagram of Angular Modules. Adopted from "Technical Design in UML for Angular Applications" by Admiraal, H. (2017).

Logical View with Class Diagram

On the server-side, the book store application has a Book class the has the following attributes: title, ISBN, author, picture, and price. It also has the following methods: addBook, fetchBooks, fetchBook, updateBook, and deleteBook. See Figure 14 for class diagram for the Book class.

As our database is using a NoSQL database, the diagram will be shown with the JSON format. See Figure 15 for NoSQL diagram.

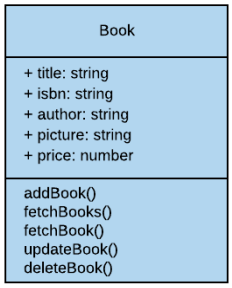

Figure 14. Class Diagram for Book Class

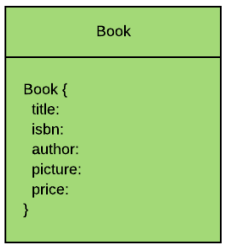

Figure 15. Diagram for NoSQL

Scenarios with Sequence Diagram

The scenario described is a user accessing a book store application. When the user enters the URL, JavaScript will be run and will hit the router of the front-end server, which is AppRoutingModule. AppRoutingModule will call the BooksComponent, which will load fetchBooks as its dependency injection. The fetchBooks will then create an HTTP request to the back-end server with a router, controller, and model to process the request and request to the database server. The database server processes the request and, with the back-end server waiting, grabs the data and sent it back to the front-end server as a JSON response. The front-end will now have the data and the template view to show to the user.

5. EVALUATION

The method to evaluate the impact of the documentation and architectural diagram in decreasing the learning curve of full-stack development is through a survey. The author used Google Forms to create two similar survey forms. They both ask for information about the participants, assessment before reading the documentation, and assessment after reading the documentation. The only difference between the two survey forms is the documentation they will assess. One is assessing the official documentation for the MEAN stack, and the other is the author's documentation for the MEAN stack.

The evaluation method was through student research group participation, which might have a potential bias due to the testers and test subjects being acquainted. The survey had a small sample size that might not be considered a good representation.

6. RESULTS AND FINDINGS

We recruited the students of a research group that the author had collaborated. For the survey on the author's documentation, we surveyed 9 students who are or had a degree in computer science. Among them, 3 students have 0-1 years of experience building software, and 4 students have 2 or more years of experience building software. On average, 3 students have used and have work experience with UML, Full-Stack Web Development, and MEAN Stack.

After reading the official documentation, the 6 students saying it increased their understanding of the MEAN stack, and the description and diagrams in the documentation were helpful. The official documentation got an overall rating of 4 out 5 for 9 students in Figure 16. Most respondents suggested showing more diagrams to explain the technology. Especially, the sequence diagram in Figure 17 was the most helpful in increasing their understanding.

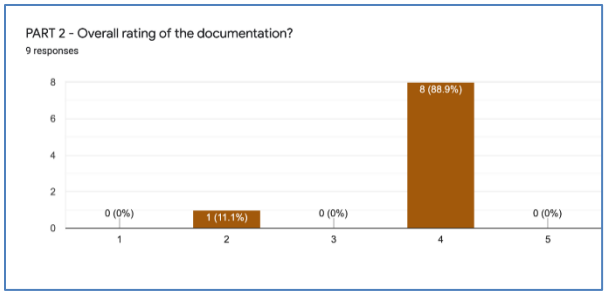

Figure 16. An overall rating of 4 out 5 for 9 students

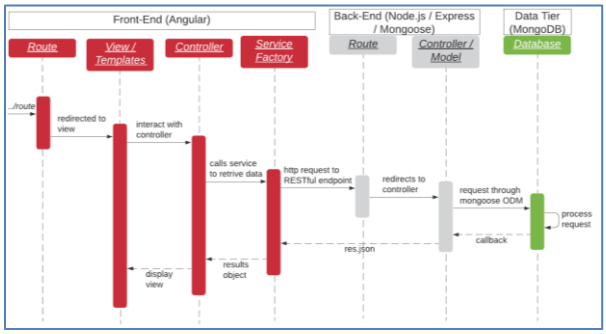

Figure 17. A sequence diagram for MEAN Stack Request/Response

The author's documentation got 57% response of 4 out of 5 and 43% response of 5 out 5 for their overall rating of the author's in Figure 18. The restaurant analogy and process view sequence diagram were the most helpful in increasing their understanding. Most respondents suggested adding more descriptions and more examples for the documentation.

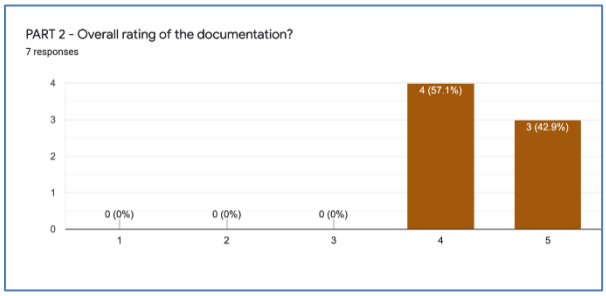

Figure 18. An overall rating of the author's documentation

To compare the two results, we used the average return of high-level understanding to 15-minute time spent. It is calculated by using each result's average score of all responses divided by the max score of 5. No need to divide by 15-minute as the same amount of time was used for the documentation comprehension portion of the survey. The average return of high-level understanding to 15-minute time spent was 67% for the official documentation and 80% for the author's documentation. With this small sample size, it shows that the author's documentation is 13% better than the official documentation for return on high-level understanding.

7. CONCLUSION

The learning curve from open-source full-stack web development is decreased with the help of high-level overview diagrams. The author's documentation is 13% more effective than the official documentation to decrease the learning curve. Documentation with architectural modeling adds additional time and resources to build. High-level overview diagrams should be constructed instead of low-level ones to decrease the time and resources to construct these diagrams. Most open-source full-stack development only has text descriptions and codebase in their documentation, making it too verbose. Implementing architectural modeling in the documentation of an open-source full-stack development gives a visualization that the developer can easily follow and digest the high-level concept. Thus, decreasing the learning curve for the developers.

8. FUTURE WORK

First, we need to increase the sample size for future work. Also, we need to analyze the impact of software documentation according to software development experience.

Second, with UML Notation, we can easily decouple the technology stack and integrate new technologies. For example, we can extend the MEAN stack scenario view using a sequence diagram to add offline capabilities, one of the features of a Progressive Web App (PWA). PWA is an enhancement strategy to create a cross-platform web application. We can also create a physical view using a deployment diagram for PWA. It can show how adding IndexedDB wrapper in the Angular application to communicate with IndexedDB, client-side storage. It can also display a service worker and cache storage to store data and be retrieved when the application's internet connectivity is offline.

9. REFERENCES

Adhikari, A. (2016). Full Stack JavaScript: Web Application Development with MEAN. Retrieved from: https://pdfs.semanticscholar.org/547a/f8ea205a8cc7c02506f58c9599a447da07a7.pdf

Admiraal, H. (2017) Technical Design in UML for Angular Applications.

Aghajani, E., Nagy, C., Vega-Márquez, L., Linares-Vásquez, M., Moreno, L., Bavota, G., & Lanza, M. (2019). Software documentation issues unveiled. *In Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press, 1199–1210. Retrieved from https://dl-acm-org.proxy.cityu.edu/doi/abs/10.1109/ICSE.2019.00122

Amazon Web Services. (2020). Amazon Web Services. Retrieved from: https://aws.amazon.com/

Arisholm, E., Briand, L., Hove, S., & Labiche, Y. (2006). The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation. *IEEE Transactions on Software Engineering*. 32(6), 365-381. Retrieved from: https://ieeexplore.ieee.org/document/1650213

Britto, R., Šmite, D., & Damm, L.-O. (2016). Group Learning and Performance in a Large-scale Software Project: Results and Lessons Learned. Retrieved from https://www.diva-portal.org/smash/get/diva2:1143810/FULLTEXT01.pdf.

Dunka, B., Emmanuel, E., & Oyerinde Y. (2018). Simplifying Web Application Development Using-Mean Stack Technologies. *International Journal of Latest Research in Engineering and Technology (IJLRET)*.

Esostalgic. (31, July, 2014). After much research and photoshop, I am proud to give you the finalized #nodejs System diagram. Retrieved from https://mobile.twitter.com/Esostalgic/status/494959181871316992

Fagerholm, F., Sanchez Guinea, A., Borenstein, J., & Munch, J. (2014). Onboarding in Open Source Projects. IEEE Software, 31(6), 54–61. Retrieved from: https://doi.org/10.1109/MS.2014.107

Kang, K. & Hahn, J. (2009). Learning and Forgetting Curves in Software Development: Does Type of Knowledge Matter? Retrieved from: https://pdfs.semanticscholar.org/05d1/a1ff8b17168bae748dca886ee4b3299f0dea.pdf

Kim, W., Chung, S., & Endicott-Popovsky, B. (2014). Software Architecture Model Driven Reverse Engineering Approach to Open Source Software Development. Proceedings of the 3rd annual conference on Research in information technology. Retrieved from https://www.tacoma.uw.edu/sites/default/files/sections/InstituteTechnology/W_Kim.pdf

Koren, I., & Klamma, R. (2018). The Exploitation of OpenAPI Documentation for the Generation of Web Frontends. *International World Wide Web Conference Committee*, 781–787. Retrieved from https://doi.org/https://dl.acm.org/doi/10.1145/3184558.3188740

Kruchten, P.B. (1995). The 4+1 View Model of architecture, IEEE Software. 12(6). 42 –50. Retrieved from: https://pdfs.semanticscholar.org/c5f4/9f3fe7624bd8ecfc5321d8e7b64a505b8f67.pdf

Kulkarni, C. M., & Takalikar, M. S. (2018). Analysis of REST API Implementation. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 3(5), 108–113. https://doi.org/https://www.researchgate.net/publication/338336716_CSEIT183535_Analysis_of_REST_API_Implementation

Lee, B. (2018). Ten simple rules for documenting scientific software. PLOS Computational Biology. 14(12). Retrieved from https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006561.

Mastropasqua, F. (2016). You Might Be an Amateur Programmer and Not Even Know It. Retrieved from https://www.clearlyagileinc.com/agile-blog/2016/5/21/you-might-be-an-amateur-programmer-and-not-even-know-it.

Miles, R., Hamilton, K. (2006). Learning UML 2.0. Switzerland: O'Reilly Media.

Node.js. (2020). Node.js. Retrieved from https://github.com/nodejs/node

Pore, S. S., & Pawar, S. B. (2015). Comparative Study of SQL & NoSQL Databases. *International Journal of Advanced Research in Computer Engineering & Technology*, 4(5), 1747–1753. Retrieved from http://ijarcet.org/wp-content/uploads/IJARCET-VOL-4-ISSUE-5-1747-1753.pdf

Prana, G. A. A., Treude, C., Thung, F., Atapattu, T., & Lo, D. (2018). Categorizing the Content of GitHub README Files. Retrieved from https://link.springer.com/article/10.1007/s10664-018-9660-3.

Shah, H., & Soomro, T. (2017) Node.js Challenges in Implementation. Retrieved from

https://www.researchgate.net/publication/318310544_Nodejs_Challenges_in_Implementation

Singh, V. K. (2019). CAP Theorem. Medium. https://medium.com/system-design-blog/cap-theorem-1455ce5fc0a0

Stafford, Gary. (2015) Calling Third-Party HTTP-based RESTful APIs from the MEAN Stack. Retrieved from: https://programmaticponderings.com/tag/mean-stack/

Tüzün, E. & Tekinerdogan, B. (2015). Impact of Experience Curve on ROI in Software Product Line Engineering. Inf. Softw. Technol., vol. 59, no. C, pp. 136–148. Retrieved from https://doi.org/10.1016/j.infsof.2014.09.008

Van Deursen, A., Aniche, M., Aué, J., Slag, R., De Jong, M., Nederlof, A., & Bouwers, E. (2017). A Collaborative Approach to Teaching Software Architecture. *In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17).* Association for Computing Machinery, New York, NY, USA, 591–596. Retrieved from: https://doi-org.proxy.cityu.edu/10.1145/3017680.3017737