Hamilton, J.R. Holdsworth, J. Tee, S. & Alshomali, M.A. (2018). A four (4) stage approach towards speeding GitHub OSS development. In *Proceedings of The 18th International Conference on Electronic Business* (pp. 28-35). ICEB, Guilin, China, December 2-6.



# A four stage approach towards speeding GitHub OSS development

*(Full Paper)*

John R. Hamilton*, James Cook University, Australia, John.Hamilton@jcu.edu.au
Jason Holdsworth, James Cook University, Australia, Jason.Holdsworth@jcu.edu.au
SingWhat Tee, James Cook University, Australia, SingWhat.Tee@jcu.edu.au
Mohammad Azeez Alshomali, James Cook University, Australia, Mohammada.Abdulhassan@my.jcu.edu.au

**ABSTRACT**

Many open source software (OSS) project creators adopt GitHub as their chosen online repository. They seek out others within the global OSS community of developers. Such community developers are then encouraged to add their capabilities, ideas and coding into a creator's developing OSS project. A structural equation modelling study of three top OSS programming languages deploys GitHub's operational elements as a four stage directional suite of (1) dependent, (2) intermediaries, and (3) independent elements. It shows a project's activity levels can be enhanced when additional project contributions are effectively stage-wise pursued. A staged development approach helps creators understand the process of attracting OSS developers into a creator's GitHub project.

*Keywords*: GitHub, open source software; OSS, developer repo, big data

*Corresponding author

## INTRODUCTION AND MOTIVATION

GitHub is the most popular hosting site for open source software (OSS) development and repositories (Cosentino, Izquierdo & Cabot, 2017). It houses over 57 million project repositories, with contributions from over 28 million developers (GitHub, 2018). GitHub's active big data projects are places where its software developers, and/or its responders or contributors alter and generally improve their developing OSS project (Gousios, *et al.*, 2014). Occasionally, apparent GitHub projects (each housed in its own repository or repo) can just be an individual programmer's additional code storage site, or they can be a cache that stores a personal code such as a 'code cracking tool' allowing unauthorized access into an existing commercial software package such as illegal pathway into the Chinese internet..

Over time, GitHub's active OSS developer repositories grow in size. Here, new knowledge and additional software capabilities are included as: (1) project corrections and improvements are made, (2) issues are discovered, (3) code is tested, stabilized and solved, (4) observers and raters offer the project external recognition, (5) coders, external developers, and contributors enhance the project's capabilities, and (6) new code, ideas, and documentation are accepted into a specific GitHub OSS project's master branch. Thus, a GitHib repository is dynamic ecosystem (Alshomali *et al.*, 2017) made up of ongoing contributing elements built by a global suite of uniquely-skilled, code-related contributors (Tsay, Dabbish & Herbsleb, 2014).

As little is known of how a GitHub ecosystem of contributors interact, this pilot study models the three most frequently engaged GitHub programming languages. It tests whether GitHub's operational elements do actually combine and align into a directional suite of (1) dependent, (2) intermediary, and (3) independent elements. It also seeks answers as to whether these combined elements exert an effect (or effects) that can provide other repo developers with a potential way to speed their individual project's 24/7/365 net activity level developments.

## BUILDING THE GITHUB ECOSYSTEM ELEMENT MEASURES

### Theoretical Basis

Within GitHub's OSS platform, teams of software developers act as communities and associates. They combine their interests into solving problems and they build operational or release versions of software (Wu *et al.*, 2014). This behavior fits across 'Information Integration Theory' which also builds on the common beliefs and behaviors of the 'Theory of Planned Behavior' (Ajzen, 1991). It links each community of GitHub software developers' attitudes with their subjective norms (Ajzen, 1991) in working collectively, and intentionally, on a project. Thus, a behavioral strength-of-belief is generated, and together they theoretically combine to jointly create an increase in a project's activity level. This also likely reduces this project's overall OSS development time.

Further theoretical support emerges through the 'Theory of Social Translucence' which draws a broad, visual, community-wide awareness around a project, its design strategies, and its deliverable(s) targets (Erickson & Kellogg, 2000). Here, each member of the project's community is recognized for their contributions, and ongoing activities – again ensuring collaboration into and

throughout each shared project. (Dabbish *et al.*, 2012) also support that such community behaviors do occur within GitHub projects.

Natural Ecosystem theory infuses a living community with its non-living organisms, whilst social network theory understands the interactions of participants inside a network. These theories jointly underpin that each element interacts across biological, physical and chemical processes. Similarly, in a GitHub repo ecosystem, the living community of OSS developers, observers, and interested parties interact within the GitHub project, and together they build an operational software solution that runs on a non-living hardware platform. A GitHub ecosystem also generates participant gratification which may be further behaviorally-understood through user gratification theory and motivation-consumption-gratification theory. (Katz, 1959; Severin & Tankard, 2000; Oliver & Raney, 2011; Hamilton & Tee, 2016).

**Research Agenda**

Hence, this study considers both such theory, and the behavior of OSS developers contributing into 600 top GitHub language specific projects. It study's 200 projects for the three most popular GitHub programing languages, and engages their OSS element contributions to poses answers to the research questions:

- *RQ1: do GitHub's operational elements actually combine and align into a directional suite of (1) dependent, (2) intermediary, and (3) independent elements for a programming language?*
- *RQ2: do these combined elements exert an effect (or effects) that can provide other repo developers with a potential way to speed their individual project's net activity levels?*

## GITHUB ECOSYSTEM CONTRIBUTING ELEMENTS

A GitHub project commences when a project creator seeks OSS development assistance to create a repo as generically shown in Figure 1. Those on social media, hacker sites, or on GitHub observation watch the project. There are both passive OSS watchers who merely observe a GitHub repo, and active OSS developer watchers who monitor and occasionally comment on this GitHub repo. These constitute the 'watchers' element. Others note the quality or direction of the project and rate it. These represent the 'stars' element. Still others take a sample of the project creator's code and either use it themselves for other purposes, or develop it. These skilled OSSS developers make-up the 'forks' element. These three elements collectively engage and commence to assist the initial creator's project. Hence these independent variables constitute the initial engagement (or interest) group.


Figure 1: Aspects of a GitHub Repo

Table 1 summarizes the GitHub open source developer contributions to the above repo.

Table 1 Repo contributions

| Repo Encounters | Description |
|---|---|
| Watchers | developers who observe rrepo content changes |
| Stars | developers who indicte a liking for the repo |
| Forks | developers who take a copy of the repos to work on |
| Issues | developers who solve a software issue for the repo |
| Pulls | developers who feed-back a potential solution to improve the repo |
| Contributors | developers who build a direct contribution to the repo |
| Releases | milesstone in the development life cycle of the repo |
| Commits | new addition/update to the current repo |

When a developer clones a repo, this gives them a complete copy of the repo content without necessarily being an active part of that repo. Unfortunately, GitHub statistics do not track information about cloning. When a fork element is then feedback (with new developments) into the creator's original project, and these points are accepted, then this action contributes a 'pull' element. An OSS developer can also recognize a problem that need to be solved (arising from several possible sources). This constitutes an 'issues' element, which once solved is included in the creator's OSS improvements. As time progresses a 'release' element or new, improved version of the creator's OSS code may emerge. As OSS developers come to the creator's project, some skilled ones are invited to join as new additions to the 'contributors' element. Thus the intermediate variables may split into two groups – one raising expectations, and the other adding values to the project.

All these community elements assist the creator in the build of a successful and useful OSS outcome. Each addition that is committed to the project is termed a 'commit' element. Hence the more accepted commits, the higher the project's activity level. This dependent variable (commits) delivers levels of gratification to its OSS community.

Thus, the GitHub project repo has at least three main community blocks of OSS elements: (1) a starting, dependent, engagement/interest group, (2) an intermediary, participant and solvers group, and (3) an intermediary values capturing group, and (4) an independent or OSS activities outcome group. We conceptually model these elements as Figure 2.

GitHub offers select approaches when analyzing its data. However, to build a popular GitHub project, a forks approach is typically selected. Aggarwal, Hindle & Stroulia, (2014) suggest extracting top repos against number of forks generally offers clearer, more-consistent documentation advice, and this helps to draw-in other coding contributors (Hata *et al.*, 2015). Documentation is also related to issues, solutions, and testing (Tsay, Dabbish & Herbsleb, 2014) and to social media and websites (Jiang *et al.*, 2017), again suggesting forks, watchers and stars are also likely related.

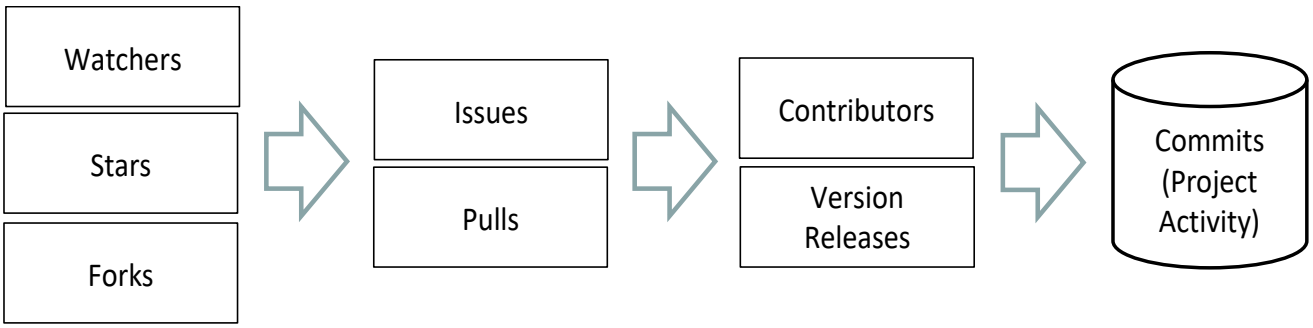



Figure 2: GitHub project ecosystem model

.

## METHODOLOGY

This study extends and refines Alshomali *et al.*'s (2017) and Hamilton *et al.*'s (2017) papers. It engages three programming languages, and follows a similar procedure for data capture. It requires many weeks of continuous downloading to collect the programming language repo data sets. Hence, the number of top repos for each of the top three GitHub programming languages engaged in this study is limited at 200 repos each. This number of repos per language allows for a more definitive structural equation model finding, as with over 20 cases per model construct a valid model can be established for each programming language studied (Hair *et al.* 2010).

**Data Capture**
This study again uses GitHub's web-based API. It again extracts GitHub data using this team's developed code (available at https://github.com/ozyjay/GithubQuery). This process of data extraction takes weeks of continual download time. Captured data is then collated into eight constructs with pulls open and closed combined into a pulls construct, and issues open and closed combined into an issues construct. Eight GitHub constructs and their measures are then imported to SPSS and AMOS 23.0 and SEM path modelled.

**Data Analysis**
The data is visually assessed to remove any outlier cases such as individual code storage cases, or any anti-social/'crack code tool' cases, or any non-release sites from each programming language's data set. This discriminant validity check enhances the accuracy of the targeted data and its modelling.

Table 2. GitHub construct elements (top 200 projects per language)

| GitHub Elements | Watchers | Stars | Forks | Releases | Contributors | Issues | Pulls | Commits | Total |
|---|---|---|---|---|---|---|---|---|---|
| JavaScript | 165153 | 3553983 | 809328 | 15324 | 46707 | 434580 | 241377 | 708605 | 5,975,057 |
| Python | 100346 | 1551498 | 426286 | 10897 | 44978 | 296256 | 306725 | 1178386 | 3,915,372 |
| Java | 119342 | 1495595 | 527948 | 9230 | 14563 | 185634 | 130475 | 941579 | 3,424,366 |
| | | | | | | | | | 13,314,795 |

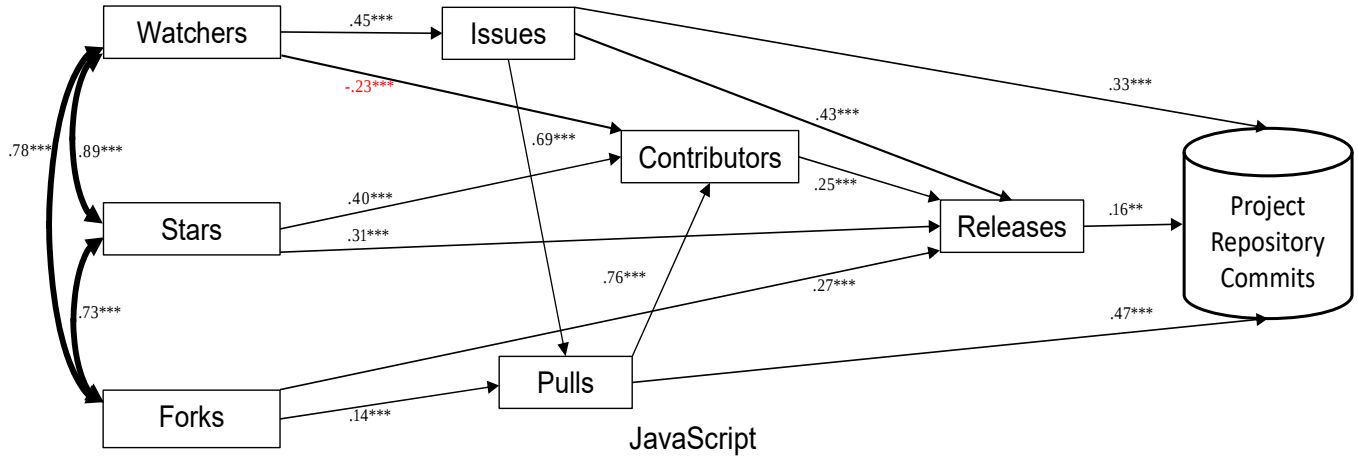
For these top 200 active projects in JavaScript, Python and Java the total element measures total 13.3 million, and all relate to project code or information contributions. Each project is current, active, and has existed in GitHub for at least one year. The dependent outcome variable commits is measured as the number of accepted changes into the repo of the project.

In this study we expect path differences, as OSS developers working in one of these three languages likely bring different individual talents and code building capabilities to their chosen OSS project. Also, JavaScript and Python are more a client side scripting language and Java is more a server side language activates web/file targets (and runs fast across platforms). Thus this study deploys different languages and it looks for commonalities across Figure 2's elements.

Each GitHub OSS project activity level structural path model, for each of the three GitHub OSS programming languages, is shown below across Figures 2 to 4. These three Figures each engage their respective Table 2 programming language data sets.

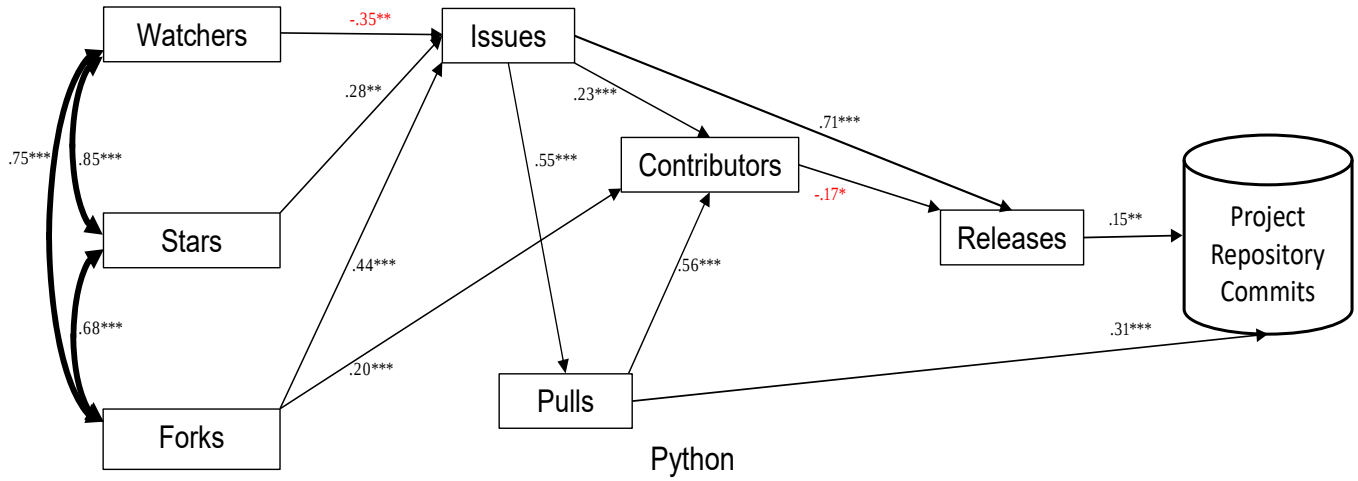
After outlier removal (typically those projects that: (1) are not actually OSS developer community projects, or (2) involve anti-social activities, or (3) have just one developer, or (4) show no releases) all three programming language SEM path models show excellent fit.

The $\chi 2$/df ratios (between 1 and 3) supported by p values (>0.05) indicate excellent fit. All other measures displayed also support excellence of fit. (Cunningham, 2008; Hair *et al.*, 2010). All models have data sets in excess of 160 cases (20 per element) (Hair *et al.*, 2010; Muthén & Muthén, 2002). The maximum number of additive paths towards the dependent outcome variable is five. This is an acceptable maximum structural path length (Hair *et al.*, 2010). Validation by bootstrapping (200 times) further confirms model convergence and validity (Cunningham, 2008; Hair *et al.*, 2010). Thus each structural path model is representative of its top OSS GitHub programming language, and each offers suitable representative insights into the stage-wise behavior of their constituent elements.



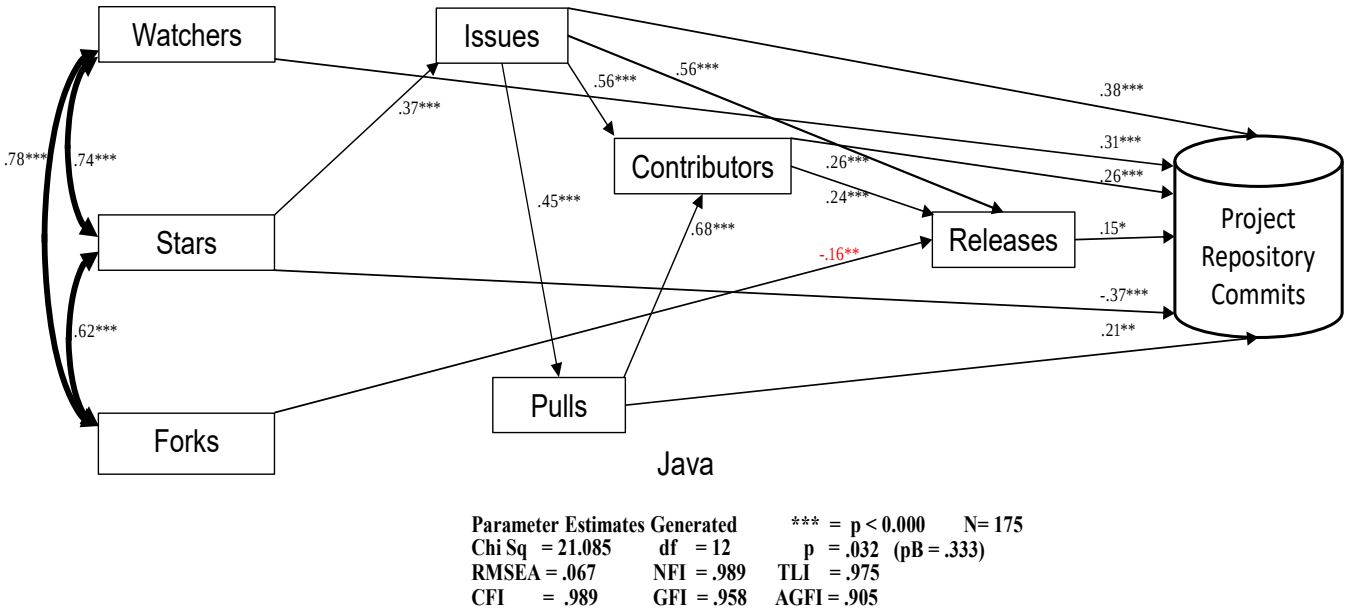

JavaScript

**Parameter Estimates Generated** *** = p < 0.000 N = 195
**Chi Sq = 20.143** df = 12 p = .064
**RMSEA = .059** NFI = .989 TLI = .985
**CFI = .994** GFI = .976 AGFI = .927




Python

**Parameter Estimates Generated** *** = p < 0.000 N = 195
**Chi Sq = 21.774** df = 14 p = .083
**RMSEA = .054** NFI = .989 TLI = .985
**CFI = .993** GFI = .974 AGFI = .932




Figures 3 to 5: GitHub JavaScript SEM path model for project activity level.

## DISCUSSION

Figures 3 to 5 show the concept model's 4 stage transition to delivering project activity. In each case there is an internal OSS development transition from issues to pulls to contributors to releases. In each case these paths are strong and significant. Hence, based on the similarities across our three programming languages, we propose Figure 2's conceptual model as a general project activity model for GitHub. We extend this into Figure 6, indicating the likely progression of successive element influence triggers.

Considering the independent external factors (watchers stars and forks), we note watchers just track the project's activities, and generally, they do not contribute. Hence, they generally exert an indirect, and negative effect, on the overall project activity levels, Thus, although watchers and fairly highly correlated with stars and forks, a strategy to move watchers into being stars providers, or into taking forks, or even into being contributors, or into generating issues contributions is desirable when pursuing accelerated GitHub OSS project developments.

This study notes there are four negative, but significant, paths (one or two in each programming language). Each negative path is both logical for the programming language's normal use applications, and readily explained. Hence when combined with other path strength measures, each programming language SEM model is accepted as valid.

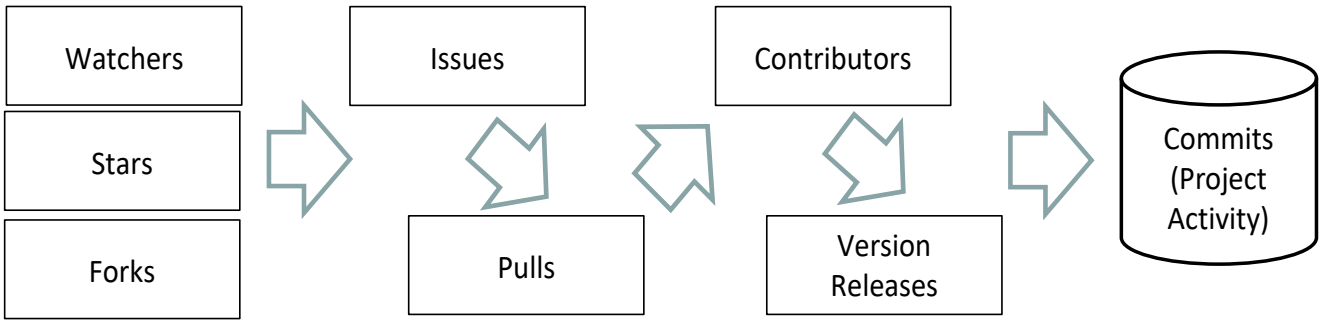



Figure 6: Progression of successive element influence triggers across the GitHub project ecosystem model

## CONCLUSION

This SEM study shows that GitHub's operational elements do actually combine and align into a directional suite of (1) dependent, (2) intermediaries, and (3) independent elements for each programming language studied in GitHub. Figures 3 to 5 support the GitHub project OSS contribution elements of a project's activity level into its repository storage as a GitHub ecosystem.

The path models (Figures 2 to 4) indicate strong interrelationships amongst the elements. Thus, any improvement to the elements likely positively accelerates the GitHub projects OSS development. The SEM study also shows a consistency of combined elements exerting positive directional pathways effects towards the delivery of a project's activity levels. This is summarized in Figure 5's progression of successive element influence triggers across the GitHub project ecosystem model. This suggests a project's activity levels can be enhanced when additional project contributions are effectively stage-wise pursued. This staged development approach helps creators understand the process of attracting further OSS developers into a GitHub project.

Beyond the three languages studied, GitHub contains an additional 150+ project programming languages. Hence, this study's GitHub project ecosystem model likely has application to many of these additional languages – especially when their creators wish to (1) identify the interrelationships within each GitHub project's repo ecosystem, and/or (2) model the elements of these interrelationships as a four stage structural model, and/or (3) stage-wise attempt to accelerate the element uptakes within each of these projects, and/or accelerate their projects towards completion.

The knowledge flows embedded in this study's GitHub project ecosystem model (Figures 1 and 5) likely extends and hold ideas application to other OSS hosting platforms beyond GitHub. Those individual OSS platform creators, working on such hosting platforms, can use this study's approach when considering pathways towards specifically minimizing the OSS development timeframes involved whilst pursuing their next OSS project release.

## REFERENCES


[1] Aggarwal, K., Hindle, A., & Stroulia, E. (2014). Co-evolution of project documentation and popularity within github, In Proceedings of the 11th Working Conference on Mining Software Repositories (pp. 360–363). MSR, Hyderabad, India, May 31-June 1.

[2] Ajzen, I. (1991). The theory of planned behavior. Organizational Behavior and Human Decision Processes, 50(2), 179–211.

[3] Alshomali M-A, Hamilton, J.R., Holdsworth, J., & Tee, S. (2017). GitHub: Factors Influencing Project Activity Levels, In Proceedings 17th International Conference on Electronic Business (pp. 295-303). Dubai, United Arab Emirates, Dec 4-8.

[4] Cosentino, V., Izquierdo, J.L.C., & Cabot, J. (2017). A Systematic Mapping Study of Software Development With GitHub, IEEE Access, 5, 7173–7192.

[5] Cunningham, E. (2008). A Practical Guide to Structure Equation Modeling Using AMOS, Melbourne, Australia: Streams Statsline.

[6] Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: transparency and collaboration in an open software repository, In Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (pp. 1277–1286). ACM, Seattle, Washington, Feb 11-15.

[7] Erickson, T., & Kellogg, W.A. (2000). Social translucence: an approach to designing systems that support social processes. ACM Transactions on Computer-Human Interaction - Special issue on human-computer interaction in the new millennium, Part 1, 17 (1), 59–83.

[8] GitHub (2018). Retrieved from: https://en.wikipedia.org/wiki/GitHub (28 Aug 2018).

[9] Gousios, G., Vasilescu, B., Serebrenik, A., & Zaidman, A. (2014). Lean GHTorrent: GitHub data on demand. In Proceedings MSR2014: the 11th working conference on mining software repositories (pp. 384-387). ACM, Hyderabad, India, May 31-June 1.

[10] Hair, J.F., Anderson, R.E., Tatham, R.L., & Black, W.C. (2010). Multivariate Data Analysis (7th Edition), UpperSaddle River, New Jersey: Pearson Education International.

[11] Hamilton, J.R., & Tee, S. (2016). Consumers and their internet of things items: A consumptive measurement approach using social media. In proceedings 1st World Congress and 21st Asia Pacific Decision Sciences Institute Conference (pp. 198-203), Beijing, China, July 24-28.

[12] Hamilton, J.R., Tee, S., Holdsworth, J., & Alshomali M-A. (2017). Analysing Big Data Projects using Github and JavaScript Repositories. In Proceedings 17th International Conference on Electronic Business (pp. 290-294). ICEB, Dubai, United Arab Emirates, Dec 4-8.

[13] Hata, H., Todo, T., Onoue, S., & Matsumoto, K. (2015). Characteristics of sustainable OSS projects: A theoretical and empirical study. In Proceedings 8th International Workshop on Cooperative and Human Aspects of Software Engineering (pp. 15–21). CHASE 2015, Florence, Italy, May 16-24.

[14] Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P.S., & Zhang, L. (2017). Why and how developers fork what from whom in GitHub. Empirical Software Engineering, 22(1), 547–578.

[15] Katz, E. (1959). Mass Communications Research and the Study of Popular Culture: An Editorial Note on a Possible Future for this Journal. Departmental Papers (ASC), Journal. Studies in Public Communication, 2, 1-6.

[16] Muthén, L.K., & Muthén, B.O. (2002). How to Use a Monte Carlo Study to Decide on Sample Size and Determine Power. Structural Equation Modeling: A Multidisciplinary Journal, 9(4), 599-620.

[17] Oliver, M.B., & Raney, A.A. (2011). Entertainment as pleasurable and meaningful: Identifying hedonic and eudaimonic

motivations for entertainment consumption. Journal of Communication, 61(5), 984-1004.
[18] Severin, W.J.; & Tankard Jr., J.W. (2000). New Media Theory. Communication Theories: Origins, Methods and Uses in the Mass Media. Addison Wesley Longman.
[19] Tsay, J., Dabbish, L., & Herbsleb, J. (2014). Let's talk about it: evaluating contributions through discussion in GitHub. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 144–154). FSE, Hong Kong, China, Nov 16-22.
[20] Wu, Y., Kropczynski, J., Shih, P.C., & Carroll, J.M. (2014). Exploring the ecosystem of software developers on GitHub and other platforms. In Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing - CSCW Companion '14 (pp. 265–268). CSCW, Baltimore, MD, USA, Feb 15-19.