Towards A Dependency-Driven Taxonomy of Software Types

Andrea Capiluppi
Department of Computer Science
University of Groningen (NL)
a.capiluppi@rug.nl

Nemitari Ajienka
Department of Computer Science
Edge Hill University (UK)
nemitari.ajienka@edgehill.ac.uk

ABSTRACT

Context: The evidence on software health and ecosystems could be improved if there was a systematic way to identify the types of software for which empirical evidence applies. Results and guidelines on software health are unlikely to be globally applicable: the context and the domain where the evidence has been tested are more likely to influence the results on software maintenance and health.

Objective: The objectives of this paper are (i) to discuss the implications of adopting a specific taxonomy of software types, and (ii) to define, where possible, dependencies or similarities between parts of the taxonomy.

Method: We discuss bottom-up and top-down taxonomies, and we show how different taxonomies fare against each other. We also propose two case studies, based on software projects divided in categories and sub-categories.

Results: We show that one taxonomy does not consistently represent another taxonomy's categories. We also show that it is possible to establish directional dependencies (e.g., 'larger than') between attributes of different categories, and sub-categories.

Conclusion: This paper establishes the need of directional-driven dependencies between categories of software types, that have an immediate effect on their maintenance and their relative software health.

KEYWORDS

FOSS, Application Domains, Latent Dirichlet Allocation, Machine Learning, Expert Opinions, OO (object-oriented)

ACM Reference Format:
Andrea Capiluppi and Nemitari Ajienka. 2020. Towards A Dependency-Driven Taxonomy of Software Types. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3387940.3392206

1 INTRODUCTION

According to Hindle *et al.* [15], software projects do not exist in isolation and projects belonging to similar domains share some similarities. That assumption was used as a stepping stone in a previous work [6]: in that paper, we detected that application domains played a role in how projects could be grouped, and how structural characteristics vary across (but stay regular within) groups of software systems.

Moreover, the empirical evidence on software health and ecosystems could be improved if there was a systematic way to identify the types of software for which empirical evidence applies [2, 19, 27]. Results and guidelines on software health are unlikely to be globally applicable: the context and the domain where the evidence has been tested are more likely to influence the results on software maintenance and health [25].

This work stems from the need to put those earlier findings in the context of a *taxonomy* of software types. As a matter of fact, and once it becomes clear that groups of software systems differ from each other, two research questions naturally emerge:

(1) what and how many categories should be considered in a taxonomy of software types? and most importantly,
(2) are there directional dependencies (e.g., 'larger than', 'smaller than') between groups of software types?

The investigation of past research on taxonomies of software types reveals many attempts to address the first research question. Glass and Vessey [11] found some 12 existing taxonomies, already in 1995, and at different level of granularity (whole systems or software components), while Forward [9] discovered 22 complete or partial taxonomies while surveying the field. What remains vastly unanswered is how taxonomies fare against each other: when using different taxonomies, will groups of systems be clustered in the same way?

The second research question has also started to be investigated: as one of the latest examples, [25] has shown that design patterns are more prevalent in one category of systems than others. Needless to say, there should be a systematic approach to detect and summarise these findings, and produce a full spectrum of such directional relationships.

In this paper we offer three contributions: first, we discuss how different taxonomies for software systems have been designed and used in past research, and we show how mapping taxonomy elements has an effect on the sample representation (section 2). Second, we provide two empirical case studies (sections 3.1 and 3.2), that show how the directional relationship between elements contained in different categories. These case studies reinforce the need for *dependencies* and *relationships* between elements of the taxonomy: we propose our third contribution as a dependency-driven taxonomy in section 4. Section 5 gives the conclusion of this work.

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05.
https://doi.org/10.1145/3387940.3392206

2 TYPES OF TAXONOMY

There are in general, two ways of coming up with a new taxonomy: the first is super-imposed by a standards body (professionals and/or practitioners); the second is derived from experience, observations or past data by researchers and practitioners. Below we explore the two types, with motivating examples.

2.1 Top-down Taxonomies

As a top-down taxonomy, and oriented to the practicality of organising thousands (let alone millions) of software systems, the Ubuntu flavour of Linux divides its packages into over 50 sections (or categories), ranging from very specific, task-driven types of software ('Editors': *Software to edit files. Programming environments.*[1]) to generic, attribute-wide systems that might have very different characteristics ('Java': *Everything about Java..*[2]). This categorisation was directly created, and currently used, by the Canonical company to organise the Ubuntu packages into similar systems.

A similar approach is currently in use in the GitHub repository, in the form of 'topics'[3]. As for the Ubuntu case, the categories are pre-set and provided as a filtering mechanism to search within hosted projects.

A slightly different top-down taxonomy was originally created to categorise the projects hosted on the SourceForge (SF) portal[4]. The difference with the Ubuntu or GitHub categorisations is that the (SF) categories are chosen by the developers themselves, who select one or more of those categories when hosting a new project for the first time. Currently, the available categories in SourceForge are shown in the Listing 1 below. A subset of this categorisation is been adopted by Paschali *et al.* [24] including video, games, business and enterprise, home and education, science and engineering, communications, development tools, graphics, security and utilities, and systems administration.

2.2 Bottom-up Taxonomies

Bottom-up categories are based on the practitioners and researchers' state of the art, and formulated in a way to group software systems around domains. An example is given in [9]: the process of obtaining the taxonomy was openly documented, and starting from a subset of the SourceForge taxonomy of categories[5]. As one of the steps used to validate their taxonomy, the authors requested feedback on its latest version from six software engineering practitioners.

The resulting taxonomy is composed of 4 macro-categories:

A. Data-dominant software
B. Systems software
C. Control-dominant software
D. Computation-dominant software

These four categories have been adopted in prior empirical software engineering research. Firstly, data-dominant software are software that rely heavily on the use of data. For example enterprise resource planning (ERP) software, customer-facing software, as well as software used for information display and entry [2, 27]. Secondly, systems software include software focused on the development or creation of computing systems components, for example operating systems, middle-ware, networking and communication software and other device or peripheral drivers [9].

Listing 1: categories used in SourceForge.net

(1) Communications
(2) Database
(3) Desktop Environment
(4) Education
(5) Formats and Protocols
(6) Games/Entertainment
(7) Internet
(8) Mobile
(9) Multimedia
(10) Office/Business
(11) Other/Nonlisted Topic
(12) Printing
(13) Religion and Philosophy
(14) Scientific/Engineering
(15) Security
(16) Social sciences
(17) Software Development
(18) System
(19) Terminals
(20) Text Editors

Thirdly, control-dominant software relates to software developed for the purpose of controlling other appliances and software. Examples include hardware control, embedded software, process control software (such as air traffic control), and others [10]. Lastly, computation-dominant software include software that manage and manipulate information, scientific software and artificial intelligence software [2]. In prior studies, practitioners were found to be more familiar with data-dominant software [10, 26] compared to the other three categories. With regards to cloud usage patterns, data-dominant software have also been found to be the most deployed software in the cloud [22].

Bottom-up taxonomies have also been derived by researchers using newly designed categories: as an example, Borges and Valente [4] collected some 5,000 GitHub systems (comprising of systems developed in different programming languages including JavaScript, Python, Java and others) and created 6 categories. Reading through the documentation of each system, they assigned each to one of these categories. Table 1 shows the results of such categorisation: although three (out of six) categories overall cover for some 4/5 of the sample, the percentage of *Non Web Libraries & Frameworks* systems is clearly dominant when focusing on Java systems.

Silva *et al.* [25] adopted this categorisation[6] in their empirical study on software co-change patterns, investigating if co-change patterns occur across programming languages (i.e., C/C++, Java, PHP, JavaScript, Ruby and Python) and whether different co-change

[1] https://packages.ubuntu.com/bionic/editors/
[2] https://packages.ubuntu.com/bionic/java/
[3] https://github.com/topics
[4] https://sourceforge.net/directory/development/development/os:linux/
[5] Parts of the GoogleCode and ACM classifications were also used as seeds.

[6] Although only 5 of the categories were used to cluster their study sample, with the exclusion of the documentation category.

Category	Projects (ALL)	Projects (Java)
Application Software (AS)	437	30
Documentation (D)	433	48
Non Web Libraries & Frameworks (NW)	1,439	342
Software Tools (ST)	972	49
System Software (SS)	184	26
Web Libraries & Frameworks (WL)	1,535	25

Table 1: Number of projects in the categories extracted in [4].

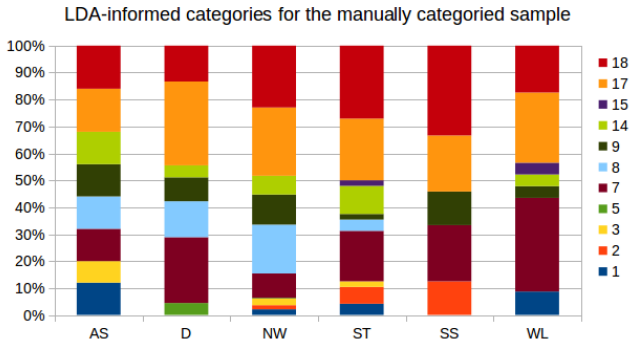

Figure 1: Original manually extracted categories broken down by the LDA-based technique. "AS" stands for ApplicationSoftware, "D" stands for Documentation, "NW" stands for NonWebLibrariesAndFrameworks, "ST" stands for SoftwareTools, "SS" stands for SystemSoftware, "WL" stands for WebLibrariesAndFrameworks. 1 to 20 are the SourceForge categories, available in the list in Section 2.2.

patterns share similarities in terms of team diversity, rippling and other trends.

The authors described the categories as follows [25]: application software (software implemented for end-users, e.g., web browsers); non web libraries and frameworks (software implementing application components); software tools (software that supports development tasks such as IDEs and source code repositories); system software (software providing infrastructure and services such as operating systems); and web libraries and frameworks (software used to implement web application interfaces) [4]. Their empirical results showed that different change patterns are more prevalent depending on the programming language or software category in focus.

Similar categorisation was adopted in a study on identifying and characterising unmaintained software projects hosted on GitHub [7] investigating areas such as reasons motivating developers to abandon projects. The outcome of the study is a model that can identify projects that are not maintained using a set of 13 features about GitHub hosted projects. The libraries and frameworks categories accounted for the highest number of unmaintained projects (e.g., projects without a commit in the last year) followed by the application software category. Within the studied sample of projects, the authors found that the category with the least number of maintained projects is the system software category.

2.3 Comparing Taxonomies

There has not been an effort yet in the literature to see how taxonomies fare against each other. Using an LDA-based approach designed to detect categories of software systems [6], we used the SourceForge categories (i.e., top-down) to classify the subset of 500 Java systems out of the 5,000 systems described in Section 2.2 (i.e., bottom-up). Figure 1 shows how the original six categories defined by Borges and Valente [4] can be broken down into the SourceForge categories outlined in Listing 1.

As visible, each 'manual' category contains more than one of the SourceForge categories, when analysing it via the LDA technique.

2.4 Cross-Cutting vs Leaf Domains

From the gathered results it is possible to notice that some categories (e.g., *Religion*, *Social Sciences*, *Formats and Protocols*) are not detectable via the LDA-based technique, whereas other categories (e.g., *Software Development*, *Mobile*) are most easily found. This means that also the SourceForge taxonomy is too coarse in some parts (e.g., *System*, *Software Development*, etc), and too fine in others (e.g., *Religion*).

Discarding the categories containing less than 3 projects, we plotted the distribution of projects along the SF categories in Figure 2. It becomes therefore possible to discuss whether certain categories could be '*cross-cutting*' for many sub-categories of software types (e.g., 'Mobile' -> 'Software Development', or 'Mobile' -> 'Education'). At the same time, Figure 2 shows that some categories might be considered as sub-categories only, in specific branches (i.e., leaf) of the taxonomy tree (e.g., 'Religion and Philosophy' could be a sub-category of either 'Desktop Environment', 'Internet' or 'Mobile').

2.5 Partial Taxonomies and mapping between taxonomies

Past literature has sometimes provided partial taxonomies, based on experts opinion, or available classifications. The paper in [15], for example, analyses, by category, the software projects contained within the Ubuntu operating system. In Figure 3, we display the categories studied as boxes, while we also apply the labels of the taxonomy derived in [9].

While several research papers have hinted at Ubuntu as an ecosystem, it is possible to observe how the types of software contained in Ubuntu, as an overarching project, are diverse, in the spectrum of a taxonomy.

3 CASE STUDIES

In this section we present two case studies, where we show how groups of systems show different behaviours, when they are gathered under the same category. For the first case study (sect 3.1), we discuss a group of specialised, blockchain-oriented software projects, and how sub-categories can be identified to drive discussion on different software maintenance and varying health. For the

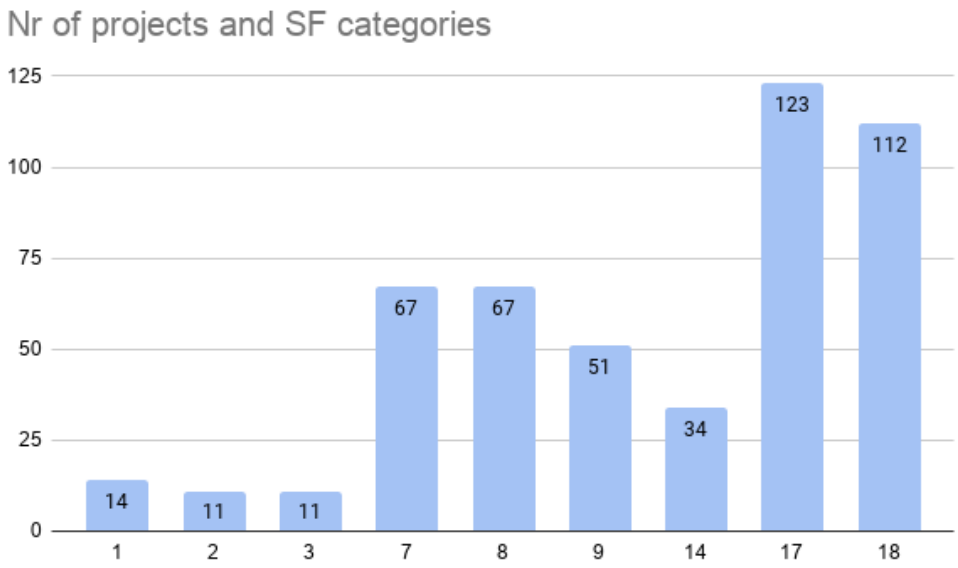

Figure 2: SF categories for the 520 Java systems. 1 represents the "Communications" category, 2 = "Database", 3 = "Desktop Environment", 7 = "Internet", 8 = "Mobile", 9 = "Multimedia", 14 = "Scientific/Engineering", 17 = "Software Development", 18 = "System". Categories with less than 5 projects are excluded.

second case study (sect 3.2), we use the sample of 500 Java systems (and 6 categories) that was discussed in Section 2.2.

3.1 Case study 1: Sub-categories of one software type

As the first case study, we consider one specific category of software types, namely *Smart Contracts* that implement features related to the Ethereum Blockchain. The original objective was to study this specific category of software projects, but the findings went beyond the research questions.

3.1.1 Project selection and sampling. For sampling projects, we used GitHub as the preferred platform: Kalliamvakou *et al.*, investigated the quality of data available from GitHub [16] and found certain issues to take note of when extracting software repository information or data from GitHub. Based on their study, we have made use of the following search criteria when selecting the projects from this category:

- The repository should not be a tutorial or library and should be aimed at the Ethereum blockchain (with Solidity being the main programming language).
- The project should have a minimum of between 5 to 10 commits. A similar filtering requirement has been adopted in previous research [28], [17] to make sure that projects investigated have some development activity.
- It should have at least 2 active contributors to avoid it being a personal project. Similar filtering requirement is adopted in previous research [1].
- To exclude un-maintained or less active projects [7], the projects should feature a minimum of one commit in the 12 months prior to the data collection from GitHub [18].

Given the outlined case study selection requirements, the selected software projects from GitHub are listed in Table 2 showing the number of contributors in each project.

3.1.2 Project clustering and categories. We clustered the projects into two enveloping categories[7]: (i) tokens[8] and (ii) *others*[9]. This is because a large number of the smart contract projects developed for use on the Ethereum blockchain network have the goal of creating a new crypto or digital currency [5, 30].

Out of the eleven projects selected, four projects (*Monerium, Grapevine Token and Crowdsale, Airbloc Token* and *TrueUSD token*) were assigned to the *Token* domain. Based on the project description in their READMe file, they all share the goal of creating a digital currency. The remaining seven projects (*Gnosis, DEXY, Synthetix, Decentralised Microinsurance, Kleros, Token-curated Registry* and *Realitio*) were allocated to the *Others* group.

3.1.3 Internal and external attributes, null hypothesis. All the projects are implemented in Solidity: the (internal) attributes extracted include the number of lines of code (SLOC), coupling between objects (CBO), number of ancestors (NOA), number of outgoing invocations (NOI), number of statements (NOS), depth of

[7] Also, considering the sample size.
[8] The main domain discussed in the blockchain field and the original use case for the blockchain technology [23]
[9] Covering other decentralised applications such as decentralised insurance, gaming, escrows, etc.

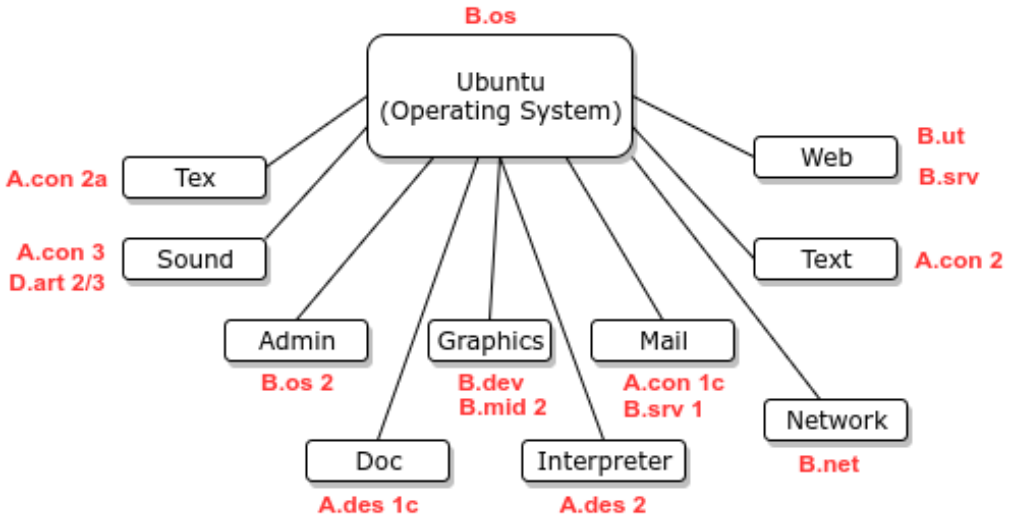

Figure 3: Dependencies between categories of software types (adapted from [15]). The red labels are the corresponding types as proposed in [9].

Table 2: Selected Ethereum Blockchain-Oriented Software Sample

Project	**GitHub Repository** https://github.com/	**# SCs**	**# Contributors**
Airbloc token	airbloc/token	4	3
Decentralized microinsurance	Denton24646/LDelay	2	2
DEXY token exchange	DexyProject/protocol	2	5
Gnosis prediction market	gnosis/pm-contracts	22	10
Grapevine World token and crowdsale	GrapevineWorld/crowdsale-contracts	4	2
Kleros	kleros/kleros	1	14
Monerium	monerium/smart-contracts	15	2
Realitio (crowd-sourced SC verification)	realitio/realitio-contracts	2	2
Synthetix	Synthetixio/synthetix	3	12
Token-curated registry	kangarang/is-tcr	5	11
TrueUSD token	trusttoken/trueUSD	6	4

inheritance tree (DIT), and comment only lines of code (CLOC). Software attributes were extracted using the SolMet tool [14].

The external attribute that was measured, per project, is the *gas* required to deploy the smart contract of those projects to the blockchain network. *Gas* is the resource which users of the decentralised blockchain network pay for computation power or resources. *Gas* is described by Grech *et al.* [12] as the fuel for computation on the Ethereum blockchain network and the amount of *gas* to be consumed for each computation or *transaction* is paid for in the native digital currency in Ethereum (i.e, the Ether).

Using the Spearman's Rank Correlation, we test for the following null hypothesis H_0: 'the application domains of the smart contracts do not play a role in the correlations between OO metrics and *gasUsed*'.

3.1.4 Results. Table 3 displays the summary statistics of the correlated metrics in the two categories. It is evident that while the smart contracts in the *Token* domain heavily rely on inherited attributes (DIT and NOA), the smart contracts in the *Others* domain consist of a higher number of statements (NOS) and outgoing functionality dependencies or invocations (NOI).

In order to increase security, specific audited token projects exist which provide the security of token-oriented projects, most especially as these projects deal with a high volume of funds (equivalent to millions or sometimes billions worth of US dollars [8, 13]). Prior to deployment, developers in these domains tend to inherit features from secure and audited smart contracts instead of developing theirs from scratch. For example, *OpenZeppelin* is a publicly available smart contract security framework on GitHub[10] offering a suite of secure smart contracts that can be extended.

This can provide an explanation for the high correlation between inheritance based metrics and the gas used for deployment in the

[10]https://github.com/OpenZeppelin/openzeppelin-contracts

Table 3: Descriptive statistics of highest correlated metrics (Token and Others categories)

	Descriptive Statistics				
	Token				
OO metrics	Mean	Median	Mode	Min	Max
NOS	16.5	9	12	0	81
DIT	2.5	2	1	0	8
NOA	4.3	2	2	0	12
NOI	6.5	4.5	0	0	28
	Others				
NOS	28.7	17	3	2	183
DIT	0.7	0	0	0	3
NOA	1.3	0	0	0	6
NOI	10.9	6	1	1	46

Token-based domain as shown in Table 4. The results in Table 4 demonstrate (statistically significant) large correlations between the inheritance-based metrics (DIT and NOA) and the *gasUsed* metric when considering only the projects in the *Tokens* domain. On the other hand, the correlation results have shown moderate correlations when looking at the non inheritance-based attributes (NOS and NOI) when evaluating the smart contracts from the 7 projects in the *Others* domain.

Table 4: Spearman's Rank Correlation of highest correlated metrics across domains and p-values (α = 0.01)

	Spearman's Rank Correlation ρ	
OO metrics	Tokens	Others
NOS	0.4 (p = 0.07971)	**0.5 (p = 0.00326)****
DIT	**0.7 (p = 0.0002)****	0.4 (p = 0.00634)
NOA	**0.7 (p = 0.0001)****	0.4 (p = 0.02041)
NOI	0.3 (p = 0.09614)	**0.5 (p = 0.00034)****

Based on these observed results we can reject the second null hypothesis H_0 and fail to reject the second alternative hypothesis H_1: the application domains of the smart contracts do play a role in the correlations between OO metrics and *gasUsed*. Developers who want to implement IDE (integrated development environment) plugins or tools for optimising gas costs for smart contracts prior to deployment can also learn from these empirical results.

3.2 Case study 2: many categories of software types

As the second case study, we consider again the sample of 520 Java systems (and 6 categories) mentioned above, and analysed in [3, 4]. Their overall dataset contains 5,000 project URI's hosted on GitHub. The sampled projects represent the Java subset of that data set[11].

3.2.1 Empirical set-up. This case study is an extension, to a different data set, of the approach that we proposed in [6]. Similarly to that study, in order to detect similarities between the categories, we performed an empirical evaluation of a suite of metrics, and a subsequent hypothesis testing. The metrics that were extracted for all the Java projects are 9, well-known structural OO attributes (NOC, DIT, CBO, RFC, WMC, LCOM, NIM, IFANIN, NIV[12]) [21].

Also similarly to [6], we adopted the Kolgomorov Smirnov (KS) test [20] to investigate the null hypothesis H_0: 'the distribution of software metrics in the compared categories are from the same population'. In addition to the original study, we also evaluated the *direction* of the test, in order to test if the distribution of values had higher values in one of the categories. Due to the multiple tests being carried out at the same time, the Bonferroni correction [29] was applied.

3.2.2 Results. Table 5 shows the results of the statistical analysis: as above, each cell contains the p-value of the Kolgomorov-Smirnov (KS) test between two subsets of the dataset. For example, the '*AS v D*' row contains the results of both the one-sided (e.g., *directional*) and two-sided KS tests between the projects in the *Application Software* domain, and the projects in the *Documentation* domain. For each pair of categories, the table provides four types of possible (colour-coded) results:

- H1, when we could not reject the null hypothesis '*the samples are drawn from the same population*', but only the alternative hypothesis H1;
- ≠, when we could reject the basic H_0 ('the samples are drawn from the same population'), but no further direction of the relationship ('larger than' or 'less than') could be established (i.e., a two-tailed KS test was performed);
- >, when we could reject H_0, and a 'X greater than Y' relationship could not be rejected (i.e., a one-tailed KS test was performed, and *alternative = "g"* as the option for the R test);
- <, when we could reject H_0, and a 'X less than Y' relationship could not be rejected (again as a one-tailed KS test, and *alternative = "l"* as the option for the R test);

All the tests were carried out using the relative p-value, and corrected with the Bonferroni correction, due to the multiple parallel tests performed.

Considering the results reported in Table 5, we observed the following:

- for most of the comparisons (e.g., cells), the KS tests reported a substantial difference between distribution of values. In only three cases, and all observed in the comparison of the AS and SS categories, the KS tests did not detect a difference in the distribution of values (the H1 cells, relatively to the CBO, DIT and LCOM attributes).
- around $\frac{1}{3}$ of the comparisons rejected the null hypothesis only for the two-tailed test (e.g., the ≠ cells), but no directions ('greater than' or 'less than') could be established.
- the projects in the Documentation (D) category tend to have larger OO attributes as compared to any other category.
- Overall, the projects in the WL category have higher values in all the OO attributes, than other categories, apart from the D category.
- The AS projects show higher values than the NW, SS and ST projects, but not against the D and WL projects.

[11] The list of projects is available at https://zenodo.org/record/804474#.XDi1S9_njCK

[12] Differently from the Solidity language of the first case study, there are several tools that can extract the OO attributes that are needed.

	IFANIN	CBO	NOC	NIM	NIV	WMC	RFC	DIT	LCOM
AS v D	≠	<	<	<	≠	<	<	≠	>
AS v NW	≠	>	<	≠	≠	≠	≠	>	>
AS v SS	<	>	H1	>	<	>	>	H1	H1
AS v ST	≠	>	<	>	≠	>	>	>	>
AS v WL	>	≠	<	<	<	<	≠	≠	≠
D v NW	>	>	>	>	>	>	≠	>	>
D v SS	≠	>	>	>	≠	>	>	≠	>
D v ST	>	>	<	>	>	>	>	>	>
D v WL	≠	>	>	≠	<	≠	≠	≠	>
NW v SS	≠	>	>	≠	<	≠	>	<	<
NW v ST	≠	>	<	≠	>	>	≠	>	>
NW v WL	≠	≠	<	≠	<	≠	≠	≠	≠
SS v WL	>	≠	<	<	<	<	<	≠	≠
ST v SS	≠	<	>	≠	<	≠	≠	≠	<
ST v WL	≠	<	>	≠	<	<	<	≠	≠

Table 5: Results of the pair-wise statistical tests of the OO metrics analysed: AS refers to *Application Software*, D to *Documentation*, NW to *Non Web Libraries And Frameworks*, ST to *Software Tools*, SS to *System Software*, and WL to *Web Libraries And Frameworks*. The ≠ sign defines dissimilar distributions; the < shows distribution X "lower than" distribution Y; while > shows X "greater than" Y.

- The NW, SS and ST projects are in a blurred category, with NW projects at higher values than the SS and ST projects.

4 A DEPENDENCY-INFORMED TAXONOMY

Based on the findings of the second case study, it becomes possible to establish directional dependencies between categories of projects, and given a taxonomy. In the taxonomy that was used in 3.2 we found that:

Directional dependencies between categories of the taxonomy
$D > WL > AS > NW > SS \approx ST$

5 CONCLUSION

In this paper we have discussed categories of software types, and taxonomies, as variability drivers for software maintenance and health. Taxonomies have been divided in top-down and bottom-up, and we have shown how categories might be cross-cutting, or end-of-branch leaves. We also compared two existing taxonomies and shown that categories do not match well transitioning from one taxonomy to the other.

In order to show the need for relational dependencies between categories of the same categories, we presented two case studies. In the first we showed that one sub-category (e.g., 'Tokens') of a software type displays a '*larger than*' behaviour with respect to another sub-category (e.g., 'Others'). In the second case study, we used a larger sample, and similarly concluded that directional relationships apply to larger categories.

These case studies show that a holistic version of the taxonomies of software types is needed, and more precise relationships between categories. These results have an immediate impact on software maintenance of software types, as well as their health, that needs to be distinguished from other software types.

REFERENCES

[1] Rabe Abdalkareem, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid, and Emad Shihab. 2017. Why do developers use trivial packages? an empirical case study on npm. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 385–395.

[2] Adewole Adewumi, Sanjay Misra, Nicholas Omoregbe, Broderick Crawford, and Ricardo Soto. 2016. A systematic literature review of open source software quality assessment models. *SpringerPlus* 5, 1 (2016), 1936.

[3] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the factors that impact the popularity of GitHub repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 334–344.

[4] Hudson Borges and Marco Tulio Valente. 2018. What's in a GitHub star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software* 146 (2018), 112–129.

[5] Efe Caglar Cagli. 2019. Explosive behavior in the prices of Bitcoin and altcoins. *Finance Research Letters* 29 (2019), 398–403.

[6] Andrea Capiluppi and Nemitari Ajienka. 2019. The relevance of application domains in empirical findings. In *Proceedings of the 2nd International Workshop on Software Health*. IEEE Press, 17–24.

[7] Jailton Junior de Sousa Coelho et al. 2019. Identifying and characterizing unmaintained projects in GitHub. (2019).

[8] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar. 2018. A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications* (2018).

[9] Andrew Forward and Timothy C Lethbridge. 2008. A taxonomy of software types to facilitate search and evidence-based software engineering. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*. ACM, 14.

[10] Ahmad Nauman Ghazi, Jesper Andersson, Richard Torkar, Kai Petersen, and Jürgen Börstler. 2014. Information sources and their importance to prioritize test cases in the heterogeneous systems context. In *European Conference on Software Process Improvement*. Springer, 86–98.

[11] Robert L Glass and Iris Vessey. 1995. Contemporary application-domain taxonomies. *IEEE Software* 12, 4 (1995), 63–76.

[12] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2018. Madmax: Surviving out-of-gas conditions in ethereum smart contracts. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), 1–27.

[13] John Hargrave, Navroop Sahdev, Olga Feldmeier, et al. 2018. How value is created in tokenized assets. In *Blockchain Economics: Implications Of Distributed Ledgers-Markets, Communications Networks, And Algorithmic Reality*. World Scientific.

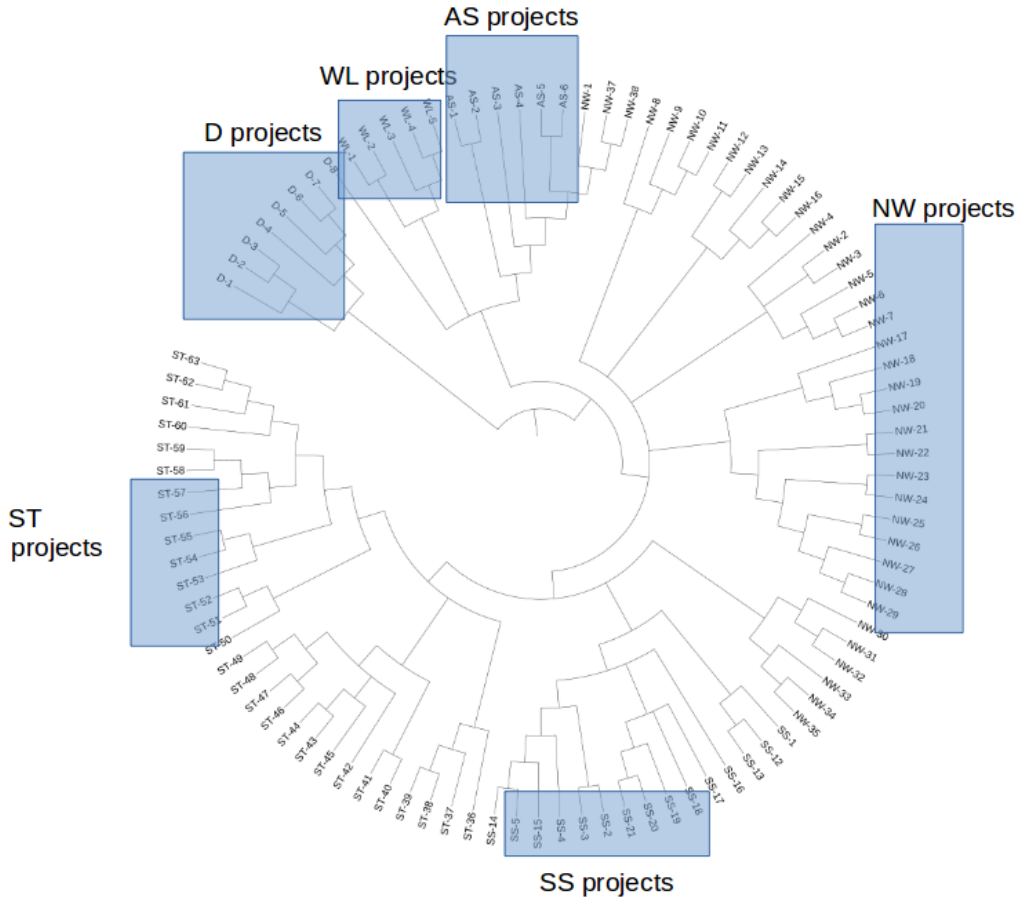

Figure 4: Model of dependencies-informed taxonomy for case study 1

[14] Péter Hegedűs. 2019. Towards analyzing the complexity landscape of solidity based ethereum smart contracts. *Technologies* 7, 1 (2019), 6.

[15] Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 837–847.

[16] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 92–101.

[17] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. 2016. Using dynamic and contextual features to predict issue lifetime in GitHub projects. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 291–302.

[18] Bin Lin, Csaba Nagy, Gabriele Bavota, and Michele Lanza. 2019. On the Impact of Refactoring Operations on Code Naturalness. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 594–598.

[19] Mario Linares-Vásquez, Sam Klock, Collin McMillan, Aminata Sabané, Denys Poshyvanyk, and Yann-Gaël Guéhéneuc. 2014. Domain matters: bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in Java mobile apps. In *Proceedings of the 22nd International Conference on Program Comprehension*. 232–243.

[20] Raul HC Lopes. 2011. Kolmogorov-smirnov test. *International encyclopedia of statistical science* (2011), 718–720.

[21] Mark Lorenz and Jeff Kidd. 1994. *Object-oriented software metrics*. Vol. 131. Prentice Hall Englewood Cliffs.

[22] Aleksandar Milenkoski, Alexandru Iosup, Samuel Kounev, Kai Sachs, Piotr Rygielski, Jason Ding, Walfredo Cirne, and Florian Rosenberg. 2014. Cloud usage patterns: A formalism for description of cloud usage scenarios. *arXiv preprint arXiv:1410.1159* (2014).

[23] Satoshi Nakamoto. 2008. *Bitcoin: A peer-to-peer electronic cash system*. Technical Report. www.bitcoin.org.

[24] Maria-Eleni Paschali, Apostolos Ampatzoglou, Stamatia Bibi, Alexander Chatzigeorgiou, and Ioannis Stamelos. 2017. Reusability of open source software across domains: A case study. *Journal of Systems and Software* 134 (2017), 211–227.

[25] Luciana L Silva, Marco Tulio Valente, and Marcelo A Maia. 2019. Co-change patterns: A large scale empirical study. *Journal of Systems and Software* 152 (2019), 196–214.

[26] Adam Solinski and Kai Petersen. 2016. Prioritizing agile benefits and limitations in relation to practice usage. *Software quality journal* 24, 2 (2016), 447–482.

[27] Nirnaya Tripathi, Eriks Klotins, Rafael Prikladnicki, Markku Oivo, Leandro Bento Pompermaier, Arun Sojan Kudakacheril, Michael Unterkalmsteiner, Kari Liukkunen, and Tony Gorschek. 2018. An anatomy of requirements engineering in software startups using multi-vocal literature and case survey. *Journal of Systems and Software* 146 (2018), 130–151.

[28] Bogdan Vasilescu, Alexander Serebrenik, and Vladimir Filkov. 2015. A data set for social diversity studies of GitHub teams. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 514–517.

[29] Eric W Weisstein. 2004. Bonferroni correction. (2004).

[30] Georgy Yuryev. 2018. *What can explain the performance of Initial Coin Offerings?* Master's thesis. University of Stavanger, Norway.