Scientific Research Publishing

Journal of Software Engineering and Applications, 2020, 13, 206-217
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Current Issues in Software Re-Usability: A Critical Review of the Methodological & Legal Issues

Tariq Saeed

Department of Information Science, College of Computer Science and Engineering, Taibah University, Madinah Almunwarah, Kingdom of Saudi Arabia

Email: tmian@taibahu.edu.sa

How to cite this paper: Saeed, T. (2020) Current Issues in Software Re-Usability: A Critical Review of the Methodological & Legal Issues. *Journal of Software Engineering and Applications*, **13**, 206-217.
https://doi.org/10.4236/jsea.2020.139014

Received: January 4, 2020
Accepted: June 7, 2020
Published: September 30, 2020

Abstract

The main objective of this research is to discuss the current legal and methodological issues in the field of software Re-Usability. Though there are enormous online forums discussing such issues via Q&A but this paper is an attempt to raise the awareness about the legal issues, which a software engineer may trap into. The paper discussed the current issues with software reusability within the legal and methodological context. This paper applied an extensive literature review to critically appraise the past studies to come to a collective conclusion. Prior to discussing the issues, the benefits of reuse were mentioned, including the saving of time and cost for users. But legally the reuse of software assets creates complexities for the user in relation to meeting all the licensing requirements and dealing with the liability in case of a breach. Methodologically, there are major barriers to reused software when it comes to technical competence and managerial issues such as a lack of resources. Even when reusing software to save time, and leverage off the specialization of other authors, the end-user must also have the technical expertise to search, adapt and merge these reusable assets into the larger software infrastructure. The review ultimately shows the high barriers still remain to software reuse which could mean that smaller developers and businesses will still be reluctant to fully utilize open-source components to the best advantage.

Keywords

Software Engineering, Software Re-Usability, Legal Issues, Methodological Issues

1. Introduction

The paper considers the reuse of software components as a primary criterion for successful future software development. As a result, previous studies have been directed towards evaluating the extent to which a software component is reusable, taking into account both the benefits and issues related to reuse. These studies can be broken down into two main categories, firstly the studies which follow an expert-based approach as their methodology, and second where researchers rely on the data driven methodologies which focus mainly on open-source software systems [1] [2]. The discussion of software reuse started with Douglas Mcllroy in 1968, who proposed the mass production of software using reusable components [3]. Though, Mcllroy at the time was not able to forecast that the software market would become dominated by several technology bemouths such as HP, Microsoft and Apple. Recently the expansion of open-source software (OSS) forums has presented developers with an opportunity to share their code, and assets among each other to collaborate and improve their software [4].

Reusability of software has grown in importance as technological development increases the complexity of B2B and B2C technology products. Customers are demanding more functionality, pushing businesses to invest heavily into the digitalization of their business models [5]. To allow such a process to be viable for smaller businesses with little technical knowledge, software reusability has allowed systems to be created by piecing together already developed components from other authors. It shows that reusability has major advantages when it comes to speed and reduced cost, and ultimately expands the potential for non-technical businesses to leverage the expertise of others to offer new, digital solutions to users. But this comes with several issues which can be combined into two groups in this study; namely (1) legal, and (2) methodological which will be evaluated.

2. Software Re-Use

Software components are parts of a system or application. Components are a means of breaking the complexity of software into manageable parts. Each component hides the complexity of its implementation behind an interface. Components can be swapped in and out like the interchangeable parts of a machine. Software reuse is regularly discussed in the form of assets, and throughout this piece the term assets will be used frequently to refer to software reuse. These reusable assets can also be referred to as building blocks [4]. Assets can be work products of any kind, and do not just encompass the source code, but also includes interfaces, test plans, architectures. Businesses are attracted to software reuse as it allows for collaboration which ultimately can be used to create value-added potential [6]. Assets can be split into several categories, from technical or management nature, long grained or fine-grained, simple to composite. Leverage is a term which is also referred to in several past studies [7] [8]. Leverage is said to occur when the reuse of one asset allows for the reuse of a chain, or

other assets which are downstream in the development process [4] and [8] showcased how reuse within the APP market was prevalent between smaller developers, studying APP's on the Android store. The authors noted that "thousands of mobile apps across five different categories in the Android Market" [8] were considered, showing that almost 23% of the classes inherit from a base class in the Android API, and 27% of the classes inherit from a domain specific base class. Most importantly, it was concluded that 61% of all mobile APP's had similarities with another, while 217 APP's are reused completely by another mobile APP in the same category.

In another paper the authors noted how the number of APP's had increased exponentially between 2007 and 2012, reaching over 1 Million at the time. However, the main reason behind such was the widespread reuse of underlying APP infrastructure which allowed non-experts to release their own APP's. This environment was supported by the increasingly open-access nature of APP software, allowing all to access and use. But this made developers more dependent on the quality of the apps and libraries that they reuse [9].

Prior to discussing the issues of software reuse a quick summary of the main benefits should be mentioned to showcase why software reuse continues to grow in popularity and is worthwhile the study. The main benefits are related to cost and time savings by reusing specific components rather than building them from the beginning [10]. To summarize reuse can increase productivity, shorten time-to-market for developments, improve software quality by reusing components developed by specialists, reduce maintenance cost, leverage technical skills and knowledge, and improve system functionality [11]. Apart from productivity gains, component reuse allows a business to reduce the critical path in the delivery information systems applications, reducing the time to market, moving into a stage of revenue generation sooner. Pitney Bowes became an early adopted of reused components in their applications and in 1996 documented tremendous labor savings from reusing software as opposed to developing all in-house [12].

3. Legal Issues

Reuse is a common and mostly advocated practice for software development. Significant efforts have been invested into facilitating it, leading to the widespread integration of open source components into proprietary software systems. But there are vulnerabilities which can increase as reusability increases especially into systems with insecure coding practices, or when an asset is reused more and more, creating a larger surface area for a cyber-attack [13]. The counterargument to this is that reused software can become more secure through its maturity and extensive vetting by other users. [14] investigated 301 open-source projects through a holistic multiple case-study methodology. What was concluded was that security vulnerabilities increase as the project size of the reusable software increases, especially when the user doesn't have a deep technical understanding over what they are necessarily using; *i.e.* code, infrastructure. Many may use reused assets for convenience and with such fail to properly understand

the code which has created them, or the process involved within their construction. Thus, the new user opens vulnerabilities in their system, especially if multiple reusable components are used and adapted for their final version. Security vulnerabilities ultimately link to legal issues as any cyber-attack, and subsequent loss of customer data, or business continuity may lead to legal challenges from customers, and thus a financial cost [15]. [14] summarized their findings with the comment "code reuse is neither a frightening werewolf introducing an excessive number of vulnerabilities nor a silver bullet for avoiding them". Legal issues surrounding software reuse can be summarized into three types of protection, namely patents, trademarks and author law. All three of these protections can be used in different forms to protect the intellectual property of the creator.

Ownership has always remained an issue to greater software re-usability. Major technology businesses are keen to keep their ecosystems under their own control, because their future financial performance could be hindered by their components becoming open source. To protect their assets businesses such as Apple place patents over the technology, over their software components. It places legal protection over their software making it impossible to be re-used within other businesses. Author law is discussed given the protection that it provides over creative elements of someone's work. While patents may be used to protect intellectual property and innovation, author law can be used to protect creative processes such as a poem, a song, or even source code for software. Though, globally, there is no set guidelines which define all components of software and thus there is confusion over issues such as how long author law lasts after a product is released; can it be adapted; who is liable among other [16].

Open-source software (hereafter OSS) also has complexities when it comes to the licenses which are attached to a specific component. There are several licenses which could be attached to OSS; from the Apache License 2.0 to the MIT License among many others. Many of these licensing regimes share the same elements (see Thompson & Jena, 2005), such as no royalties to be paid; no restrictions on the application of the software and creation of modifications or derivatives work permitted. The entire purpose of open source licensing is to limit restrictions and promote public availability; accordingly, there is no restriction on OSS being used for commercial purposes [17] [18]. But in using the OSS within their software, the developer becomes obliged under certain licensing terms which were attached to the utilized OSS components. This study has presented several theoretical examples, the main being that the user of the reused component may be required to display certain copyright notices related to the components reused in their version. To add, the user may also be required to make available certain code which could show their modifications and allow others to then use this.

These issues are a serious barrier for many developers seeking to build-upon reused components to create their own modified software. While they are willing to use reused software to build the base for their own version, many developers may then be reluctant to share their own intellectual property with others, essen-

tially make that OSS in the process. In some cases, it means the developers that use OSS may require legal assistance to ensure they comply with all the licensing agreements. Failure to do this can result in legal action from the original author, and so OSS assets should be caveated with the note that they are still under the legal ownership of the author who has the right to take legal action if deemed necessary [19]. This can be a time-consuming process when a developer uses several reused components in their final version from several different authors. Many may steer away from OSS, instead choosing to develop the components in-house as this is then their own intellectual property under their own control and subject only to their regulation. As well as copyright rules, [20] also noted copyleft. Copyleft, also cited in [21] and [22] is the process of granting the right to freely use the intellectual property under the assumption that the user will do the same with their modifications. It supports the argument from [19] that a legal issue with greater adoption of reusing is licensing agreement which may force the user to share their modifications, something which they look to protect for business success.

Legal issues also incorporate liability [23]. Software does not also perform as expected and given the rising issue of cyber security and theft, businesses are increasingly looking at components which can be added to their technology to improve security, especially around the holding of consumer-related data given GDPR regulation. OSS has become a way for businesses to improve their own security quickly and in a cost-effective manner, noted by several authors including [24]. But if the security assets put in place fail the main issue is who will be liable, the business which is using the asset in their software, or the original author of that asset [25]. Trust is also an important concept to note. A user needs to have trust that the assets they are reusing are compliant with all licensing conditions. Given that these assets can be taken and modified and modified further means that open-source forums can have multiple versions available to use all tied up within a web of modifications and multiple developers being involved [24]. It becomes hard to understand whether all these modifications have been done legally and in compliance with all licensing requirements. Ultimately for a user to reuse this they need to have significant trust that the asset is legal or have a significant team of people able to check the origin of the code, and its modifications over time. It is having the resources available to ensure that there is no unlawful plagiarism within the software being reused which could lead to legal challenges, costs and implications within the future (**Table 1**).

4. Methodological Issues

While the main benefit of software reuse is the reduction in time needed to successfully bring software to fruition a methodological issue to such may be that the software being reused may not be designed in a manner conductive to the task it is required for, requiring that ad hoc modification is applied [26]. Many of the studies cited here used a methodology in which they study the code in

Table 1. Legal issues of software re-usability.

Legal Issue	Literature Summary
Ownership	Making software available for open-source usage has benefitted businesses in reducing their workload, costs, and need for knowledge, expertise and resources during the development process [24]. Though without tracking past ownership, and enhancements to OSS a user may be reluctant to use themselves. There is also an issue related to whether the end user, after enhancing the OSS to meet their own specifications would need to then make this openly available, potentially impacting on their business competitive advantage [17] [18].
Licensing Regimes of OSS	The main issues around licensing are 1) Copyleft [21] [22], 2) Caveats attached to OSS and the potential for the owner, or author to take legal action if any are violated [25] [26].
Liability	While reused software can help businesses expand their capabilities beyond their own knowledge by leveraging of others developments, there are significant gaps over who would be liable if something went wrong within the business due to a software asset which has been reused [23]. Businesses may be reluctant to use reused components without full assurance then are protected, though studies from [8] [9] show strong growth in the usage of reused components suggesting that while liability is an issue to overcome, it is being overcome in most cases, mainly by the risk being accepted by the business as the benefits are greater than the costs.

software to find similarities which prove reuse. [27] choose to survey 128 developers over their experience with software reuse. It was found that 72% of the developers questioned cited complexity as a key barrier to software reuse. The complexity of old code essentially made it unusable.

An emergent pattern within software reuse has been that developers are only writing the visible application code which forms the tip of the APP, which differentiates the design and the way in which the APP or software interacts within the user [28]. The bulk of the back-office code and components are now reused in multiple APP's, in a mix-and-match fashion [28]. The process of software reuse is not simple. There is high technical need for any developer seeking to reuse software within their application. The developer would need to undergo a process of component searching, understanding and adaption to their own needs. For an entry-level developer with little understanding over programming this can be a daunting experience and the logistical challenges involved with searching and adapting an asset to reuse could be a significant issue [29] [30].

The availability of open-source software has increased exponentially which had led to the creation of large repositories where software could be stored and viewed by others [31]. However, while [32] cites this as a major benefit to software reuse in general, [33] discusses how the expansion of different product variants is a challenge for developers, crowding out the market with multiple options confusing a would-be developer on which makes the best option (**Figure 1**). Reuse might also hinder new ideas and the making of innovations. Balance between innovation and reuse should be determined by the company's strategy. Earlier work from [34] had created a list of obstacles to software reuse, summa

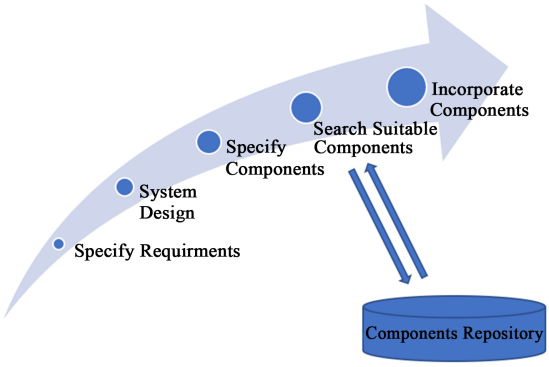

Figure 1. Illustration of the software re-use process. Created for this study.

rized into nine key problems; namely technical (structure mismatch, steep learning curves), managerial problems (infrastructure clash, turf battles, inadequate resources) and cultural or psychological issues (apathy, fear, "not invented here syndrome", ivory tower). It must be remembered that even if software is developed through reusing specific components, they still need to be maintained and updated accordingly. Reuse can add complexity by creating dependencies between developers and businesses [35]. Some of the mentioned problems with the dependencies identified by one of the respondents are web of dependencies, coordination cost, process and tool divergence and integration cost.

[36] summarized the difficulties in fully utilizing OSS as meeting the necessary pre-conditions. These included proper training of all software developers and testers to deal with OSS, including documentation, thorough programming guidelines, testing and programming and software design. The user of the asset needed to ensure they had the resources able to take these reusable components and merge them into their own infrastructure successfully. It has been noted that most reusable assets need to be changed once then are with another user to meet their individual specifications, meaning that technical knowledge is vital [37]. [38] also supported the findings of Kim (2005), focusing their study into the use of software reuse within SME's. The key findings were while increase reuse rates among SME's increased productivity and reduce total costs, it had little impact on the time to market given that SME's needed significant time to ensure that they were: 1) compliant with all legal aspects of using the reusable assets, and 2) they had developed the technical knowledge internally to adapt and maintain this software. Though the current literature is positive over OSS and shows that many of the methodological issues are overcome due to the developer community which is created by sharing software. The main factors that make OSS more reliable are: 1) that developers are usually also users of the software, as well as members of a community of developers, 2) public availability of the source code and fast bug removal practices since thousands of independent programmers testing and fixing bugs of the software [39].

The development environment is also a potential issue restricting access to reusing software [40]. Well documented software/system architectures are essential to support decision-making about reuse, required to correctly integrate

the different assets, thus reducing testing time [40]. Reuse is still possible without documentation as long as domain expertise within the team remains adequate, though without documentation there are legacy issues if key members of the development team leave the business, taking with them their knowledge and expertise. This is an important unsolved issue with legacy systems where experts have left and business and the documentation is out of sync, leaving a significant gap within the team over how to maintain and evolve the system [40] [41] [42]. The stability of the business and its software must be considered here, with stability vital to ensure business continuity [43] (**Table 2**).

Table 2. Methodological issues of software re-usability.

Methodological Issue	Literature Summary
In-house Expertise	While reusing software allows a business to leverage off the expertise of others, there is still the need for in-house expertise to understand, adapt and implement these reused components cited in [35] [37] [38] [39] showing agreement among researchers.
Meeting pre-conditions for utilizing OSS	Proper training of all software developers and testers to deal with OSS, including documentation, thorough programming guidelines, testing and programming and software design [36], with elements of this noted by [29] [30]. Again there is agreement within the literature that these issues will impact all users of reused software components.
Crowded OSS Market	Open-source software has increased exponentially which had led to the creation of large repositories where software could be stored and viewed by others. Makes it harder to search through all options and make a choice [31] [33].
Not-invented-here-syndrome	Stance adopted by some that avoids using or buying already existing products, research, standards, or knowledge because of their external origins [5] [43].

5. Conclusions

This piece has provided a systematic review of past literature related to software reuse; specifically summarizing legal and methodological issues. Many studies agree that the expansion of software reuse has supported the expansion of software options, as well as the number of APP's available to consumers. Reusing software reduces the resources needed to create an APP; *i.e.* resources such as time and cost.

Legally the main challenge for software reuse is ensuring compliance with global regulation related to patents, trademarks and author law. While patents and trademarks largely follow an accepted practice of registered intellectual property, author's law is more complex and there is confusion over what it applies to, for how long, and how this would impact on legal issues such as liability. Methodologically there are several issues to wider adoption of software reuse such as have the internal expertise to fully understand the code, or asset being used; however, many of the studies considered in this piece are largely positive

over using OSS to spur innovation and improve the productivity and security of assets, benefitting all who use OSS (Table 3).

Table 3. Summary of literature into software reuse issues.

Study	Methodology	Key Findings
[43]	Surveys	Effective reuse depends not only on finding and reusing components, but also on the ways those components are combined. The researcher refers to this process as "packaging", noting how it can be technical difficult to match up the packaging and ensure a system using reused components works as intended. Unfortunately, these styles and packaging distinctions are often implicit; as a consequence, components with appropriate functionality may fail to work together.
[44]	Surveys (51 Questions)	Well documented software/system architectures are essential to support decision-making about reuse, required to correctly integrate the different assets, thus reducing testing time. A business needs to be well prepared to identify, handle and adapt reused software for their own business needs. There needs to be that technical expertise within the business, or there is the risk that a business is using reused components which they do not fully understand; leading to issues related to reliability, legal protection, security and performance.
[44]	Testing reliability of reused software components	Highlighted that some software components used in software are not fully functional, and used in a way which was not intended. Issue is that those using these reused components or assets may not fully understand them.
[45]	Analysis of 24 software reuse projects between 1994-1997 in European countries	A third of the projects tracked ultimately failed. Three main causes of failure were 1) not introducing reuse-specific processes, 2) not modifying non-reuse processes, and 3) not considering human factors. The root cause of these issues was a lack of commitment from top management along with the belief that the business did not need to further adapt or engineer these reused components to fit their business needs; instead believing they could simply be taken from a repository and used.
[21]	Extensive literature review	Legal issues surrounding software reuse can be summarize into three types of protection, namely patents, trademarks and author law. Businesses may shy away from using reused components because of the legal risks, and the financial repercussions there could be.

Acknowledgements

The author would like to express his cordial thanks to Prof. M. Z. Khanand Dr. Wadi Boulila for their valuable comments on the initial drafts.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Frakes, W.B. and Fox, C.J. (1995) Sixteen Questions about Software Reuse. *Communications of the ACM*, **38**, 75-67. https://doi.org/10.1145/203241.203260

[2] Coulange, B. (2012) Software Reuse. Springer Science & Business Media, London.

[3] Mcllroy, M.D. (1968) Mass Produced Software Components. *Proceedings of NATO Software Engineering Conference*, Garmisch, Germany, October 1968, 138-155.

[4] Ezran, M., Morisio, M. and Tully, C. (2012) Practical Software Reuse. Springer Science & Business Media, London.

[5] Walton, P. and Maiden, N., Eds. (2019) Integrated Software Reuse: Management and Techniques. Routledge, London. https://doi.org/10.4324/9780429455520

[6] Keswani, R., Joshi, S. and Jatain, A. (2014) Software Reuse in Practice. *Proceedings of the* 2014 *Fourth International Conference on Advanced Computing & Communication Technologies*, Rohtak, India, 8-9 February 2014, 159-162. https://doi.org/10.1109/ACCT.2014.57

[7] Land, R., Sundmark, D., Lüders, F., Krasteva, I. and Causevic, A. (2009) Reuse with Software Components—A Survey of Industrial State of Practice. In: Edwards, S.H. and Kulczycki, G., Eds., *Formal Foundations of Reuse and Domain Engineering. ICSR* 2009. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 150-159. https://doi.org/10.1007/978-3-642-04211-9_15

[8] Ruiz, I.J.M., Nagappan, M., Adams, B. and Hassan, A.E. (2012) Understanding Reuse in the Android Market. 2012 20*th IEEE International Conference on Program Comprehension* (*ICPC*), Passau, Germany, 11-13 June 2012, 113-122. https://doi.org/10.1109/ICPC.2012.6240477

[9] Mojica, I.J., Adams, B., Nagappan, M., Dienst, S., Berger, T. and Hassan, A.E. (2013) A Large-Scale Empirical Study on Software Reuse in Mobile Apps. *IEEE Software*, **31**, 78-86. https://doi.org/10.1109/MS.2013.142

[10] Hauge, Ø., Ayala, C. and Conradi, R. (2010) Adoption of Open Source Software in Software-Intensive Organizations—A Systematic Literature Review. *Information and Software Technology*, **52**, 1133-1154. https://doi.org/10.1016/j.infsof.2010.05.008

[11] Morisio, M., Ezran, M. and Tully, C. (2002) Success and Failure Factors in Software Reuse. *IEEE Transactions on Software Engineering*, **28**, 340-357. https://doi.org/10.1109/TSE.2002.995420

[12] Mandal, A. and Pal, S.C. (2012) Emergence of Component Based Software Engineering. *International Journal of Software Engineering and Knowledge Engineering*, **2**, 311-315.

[13] Ayala, C.P., Cruzes, D., Hauge, O. and Conradi, R. (2011) Five Facts on the Adoption of Open Source Software. *IEEE Software*, **28**, 95-99. https://doi.org/10.1109/MS.2011.32

[14] Gkortzis, A., Feitosa, D. and Spinellis, D. (2019) A Double-Edged Sword? Software Reuse and Potential Security Vulnerabilities. In: Peng, X., Ampatzoglou, A. and Bhowmik, T., Eds., *Reuse in the Big Data Era. ICSR* 2019. *Lecture Notes in Computer Science*, Springer, Cham, 187-203.

https://doi.org/10.1007/978-3-030-22888-0_13

[15] Von Krogh, G., Haefliger, S., Spaeth, S. and Wallin, M.W. (2012) Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development. *Management Information Systems Quarterly*, **36**, 649-676. https://doi.org/10.2307/41703471

[16] Succi, G., Succi, G. and Ronchetti, M. (1996) Legal Issues Regarding Software Use and Reuse within the European Union Legislation. *Journal of Computing and Information Technology*, **4**, 179-186.

[17] Lim, W.C. (1996) Legal and Contractual Issues in Software Reuse. *Proceedings of Fourth IEEE International Conference on Software Reuse*, Orlando, FL, 23-26 April 1996, 156-164.

[18] German, D. and Di Penta, M. (2012) A Method for Open Source License Compliance of Java Applications. *IEEE Software*, **29**, 58-63. https://doi.org/10.1109/MS.2012.50

[19] Kashima, Y., Hayase, Y., Yoshida, N., Manabe, Y. and Inoue, K. (2011) An Investigation into the Impact of Software Licenses on Copy-and-Paste Reuse among OSS Projects. 2011 18*th Working Conference on Reverse Engineering*, Limerick, Ireland, 17-20 October, 2011, 28-32. https://doi.org/10.1109/WCRE.2011.14

[20] Kennedy, D.M. (2001) A Primer on Open Source Licensing Legal Issues: Copyright, Copyleft and Copyfuture. *Saint Louis University Public Law Review*, **20**, Article 7.

[21] Fitzgerald, B. and Bassett, G., Eds. (2003) Legal Issues Relating to Free and Open Source Software. Essays in Technology Policy and Law, 11-36.

[22] Rosen, L. (2005) Open Source Licensing. Vol. 692, Prentice Hall, London.

[23] McGowan, D. (2001) Legal Implications of Open-Source Software. *University of Illinois Law Review*, 241. https://doi.org/10.2139/ssrn.243237

[24] Wu, Y. (2019) Large-Scale Analysis of Software Reuse for Code and License Changes. Osaka University, Osaka.

[25] Singh, S., Singh, S. and Singh, G. (2010) Reusability of the Software. *International journal of computer applications*, **7**, 38-41. https://doi.org/10.5120/1338-1703

[26] Holmes, R., and Walker, R.J. (2013). Systematizing Pragmatic Software Reuse. *ACM Transactions on Software Engineering and Methodology* (*TOSEM*), **21**, 1-44. https://doi.org/10.1145/2377656.2377657

[27] Agresti, W.W. (2011) Software Reuse: Developers' Experiences and Perceptions. *Journal of Software Engineering and Applications*, **4**, 48-58. https://doi.org/10.4236/jsea.2011.41006

[28] Frakes, W.B. and Tortorella, M. (2008) Foundational Issues in Software Reuse and Reliability. Department of Industrial and Systems Engineering, Rutgers University, Working Paper 04-002.

[29] Mikkonen, T. and Taivalsaari, A. (2019) Software Reuse in the Era of Opportunistic Design. *IEEE Software*, **36**, 105-111. https://doi.org/10.1109/MS.2018.2884883

[30] Hooper, J.W. and Chester, R.O. (1991) Software Reuse: Guidelines and Methods, Springer Science and Business Media, London. https://doi.org/10.1007/978-1-4615-3764-9

[31] Deshpande, A. and Riehle, D. (2008) The Total Growth of Open Source. In: Russo, B., Damiani, E., Hissam, S., Lundell, B. and Succi, G., Eds., *Open Source Development, Communities and Quality. OSS* 2008. *IFIP—The International Federation for Information Processing*, Springer, Boston, MA, 197-209.

https://doi.org/10.1007/978-0-387-09684-1_16

[32] AlMarzouq, M., Zheng, L., Rong, G. and Grover, V. (2005) Open Source: Concepts, Benefits and Challenges. *Communications of the Association for Information Systems*, **16**, 756-784. https://doi.org/10.17705/1CAIS.01637

[33] Thompson, C.W. and Jena, R. (2005) Digital Licensing [Software Reuse]. *IEEE Internet Computing*, **9**, 85-88. https://doi.org/10.1109/MIC.2005.77

[34] Joshua, J.V., Alao, D.O., Okolie, S.O. and Awodele, O. (2013) Software Ecosystem: Features, Benefits and Challenges. *International Journal of Advanced Computer Science and Applications*, **4**, 242-247.

[35] Reifer, D.J. (2001) Implementing a Practical Reuse Program for Software Components. In: Heineman, G.T. and Councill, W.T., Eds., *Component-Based Software Engineering*: *Putting the Pieces Together*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 453-466.

[36] Shiva, S.G. and Abou Shala, L. (2007) Software Reuse: Research and Practice. *Fourth International Conference on Information Technology* (*ITNG*'07), Las Vegas, NV, 2-4 April 2007, 603-609. https://doi.org/10.1109/ITNG.2007.182

[37] Kim, W. (2005) On Issues with Component-Based Software Reuse. *The Journal of Object Technology*, **4**, 45-50. https://doi.org/10.5381/jot.2005.4.7.c5

[38] Khalid, H. (2017) Software Reuse: Component-Based Development Issues. *International Journal of Scientific & Engineering Research*, **8**, 201-205.

[39] Ha, W., Sun, H. and Xie, M. (2012) Reuse of Embedded Software in Small and Medium Enterprises. 2012 *IEEE International Conference on Management of Innovation & Technology* (*ICMIT*), Sanur Bali, 11-13 June 2012, 394-399. https://doi.org/10.1109/ICMIT.2012.6225838

[40] Pandey, R.K. and Tiwari, V. (2011) Reliability Issues in Open Source Software. *International Journal of Computer Applications*, **34**, 34-38.

[41] Jha, M. and O'Brien, L. (2009) Identifying Issues and Concerns in Software Reuse in Software Product Lines. *International Conference on Software Reuse*, Springer, Berlin, Heidelberg, 181-190.

[42] Jha, S., Jha, M., O'Brien, L. and Wells, M. (2014) Integrating Legacy System into Big Data Solutions: Time to Make the Change. *Asia-Pacific World Congress on Computer Science and Engineering*, Nadi, 4-5 November, 2014, 1-10. https://doi.org/10.1109/APWCCSE.2014.7053872

[43] Dantas, F. and Garcia, A. (2010) Software Reuse versus Stability: Evaluating Advanced Programming Techniques. 2010 *Brazilian Symposium on Software Engineering*, Salvador, Bahia, 27 September-1 October 2010, 40-49. https://doi.org/10.1109/SBES.2010.13

[44] Griss, M.L. (1993) Software Reuse: From Library to Factory. *IBM Systems Journal*, **32**, 548-566. https://doi.org/10.1147/sj.324.0548

[45] Jha, M., O'Brien, L. and Maheshwari, P. (2008) Identify Issues and Concerns in Software Reuse. *Proceedings of Second International Conference on In-formation Processing,* Banglore, India, 8-10 August 2008, 307-314.