Abstract

In this research paper, the authors attempt to discern from available literature the properties of distributed version control systems that adhere to the principles of open source software development, specifically the ‘bazaar’ model presented by Raymond (2000). Distributed version control systems are discussed in the context of open source software development, with comparisons drawn where relevant to traditional centralised version control systems. It is shown in this research paper that many properties of distributed version control systems can be deemed beneficial for an open source software development project, however, these benefits are contingent upon the requirements of the project and the model of governance employed.

1. Introduction
The success of a software engineering endeavour is contingent upon many factors including, but not limited to, the governance employed, the technologies utilised and the project management methodologies. In this paper, there shall exist a focus on the project management methodologies, specifically the means of source and version control that may be utilised within a project. The purpose of this research paper is to identify and analyse the characteristics of distributed version control that appear cohesive with the ideals and methodologies practised in open source software development. In addition, this research paper shall explore the criticisms levied against distributed version control systems and place them within the context of the discussion in this paper. It is intended that this research serve to critically assess the utilisation of distributed version control systems juxtaposed with the utilisation of traditional centralised version control systems. The paper shall initially commence with a discussion of the history and evolution of distributed version control systems and a discussion of the processes by which distributed version control systems operate. Consequently, after the reader has been provided with a suitable background, a discussion regarding the introduction and utilisation of distributed version control systems in open source software development shall be presented. The scope of this research is intended to be such that a small subset of literature available regarding the introduction and utilisation of distributed version control systems in open source software development shall be utilised to determine the properties of distributed version control systems that conform to the principles of open source software development. As such, it is not intended that this research serve as a comprehensive reference on the subject.

1.1. Version Control System Repositories

A version control system can be utilised to track the historical evolution of a software development project. Ruparelia (2010) states that “[a version control system] stores version information for every file (and the entire project structure) in what is generally called a repository”. There exist four primary repository models that may typically be utilised in version control systems; local, shared folder, centralised and distributed (Koc, 2011). Exploring these models further, one can consider a local repository to be contained entirely within the working environment of a single user. That is, the repository and consequently the entire codebase of a given software project, are located on a single machine utilised by a single user. This concept may be extended by introducing a shared folder on the network topology with provisions for multiple users to gain access to a repository. This method allows collaboration however has the caveat that it is typically local area network specific. To allow for the possibility for developers on a network external to that of which the repository is stored, there exists a centralised repository model whereby a server contains the repository and allows for multiple clients to have a read and write access to the source code contained within the repository. In the case of a distributed repository model, there exist multiple instances of a repository spread over a geographic or network topological area such that each user contains a working, local copy of the repository. This model allows a user to obtain, utilise and maintain a local copy of the repository whereby the source code may be modified and treated as a local repository and then combined with that of other repositories at such a time that it is
deemed necessary (Koc, 2011). This paper shall consider only the centralised and distributed models of version control repositories.

2. Distributed Version Control Systems and Open Source Software Development

Software engineering and development can be seen as communications challenge, because many individuals working on same goal collaborating with each other. Version control systems are said to improve quality of technical communication between developers across room and across continents. Improvements to version control system technology have opened new way of working together, without need active connection to central server and which to use to command the development work. Distributed version control systems are then seen as good fit for open source development, that often is global and distributed by its very nature. (Clatworthy, 2007; Rigby, Barr, Bird, German & Devanbu, 2011)

2.1. The Introduction of Distributed Version Control

Raymond (2008) referers to distributed version control systems as “the third generation”. So called “third generation” version control systems that are distributed by nature started appearing in the late 1990s and early 2000s. One of initial distributed version control systems introduced was the proprietary BitKeeper on which development commenced in 1997 by Larry McVoy (Raymond, 2008) and was utilised for the development of the Linux kernel until the free license to use BitKeeper for Linux kernel development was revoked in 2005 (Rigby et al., 2011; Raymond, 2008). The revocation of the BitKeeper license resulted in the hasty development by Linus Torvalds of a new distributed version control software, called Git, for the purpose of managing the development of the of Linux kernel project (Kuhn 2010, Raymond 2008). Additional open source distributed version control systems that emerged in the 2000s are Mercurial, Bazaar, Darcs and Monotone (Raymond, 2008; Rigby et al., 2011; “monotone”, 2012). As distributed version control systems have gained maturity, more open source projects have and are planning on moving from centralised version control system to decentralised. In their paper, de Alwis & Sillito (2009) list Perl, OpenOffice.org, Python, NetBSD as such high profile projects. Currently, all of the aforementioned projects with the exception of NetBSD have moved to use distributed version control systems (“Getting and Working With the Perl Source”, 2012; "Development » LibreOffice", 2012; "Python Developer's Guide", 2012; "NetBSD CVS Repositories", 2012).

One of the limitations of utilising a centralised version control system is that only small subgroup have write-access to the repository and main branch containing the latest official development state of the project. This subgroup referred to as ‘committers’. In an open source project committers are those persons that have earned this privilege by contributing high-quality patches and gaining, in this regard, respect from the project community (de Alwis & Sillito 2009). With new distributed version control systems, developers are not separated
into separate non-committer and committer groups, but instead each developer is able to create a clone of the repository, a copy that will contain full development history of the project (Orsila, Geldenhuys, Ruokonen & Hammouda, 2009). Orsila et al. (2009) state that this serves to make all developers equal. It can additionally be argued that clones all of the master repository are equal from a technical perspective.

One curious aspect of distributed version control system is that it makes forking a simple task. Each clone may be considered as fork of main branch (Fung, Aurum & Tang, 2012). However, distributed version control systems make the task of merging branches simple whereas, when utilising centralised systems, it is perhaps non-trivial and not without risk (O’Sullivan, 2009). Rigby et al. (2011) however, do not consider forks whereby a developer has created a clone of the master repository as a typical fork, but as a 'light-fork'. They argue that historical forks where source code of large project is copied under different governance differ significantly from forks whereby a developer makes a small, or insignificant change, ensuring that the core of the project remains unmodified. They argue that a fork of Linux Kernel created for Android handheld devices by Google can be considered as such ‘light-fork’. This is well supported by the fact that there exist active processes on integrating modifications in the Android version of the Linux kernel merged into the mainline development branch (Corbet, 2011).

Utilising a distributed version control system, developers can maintain productivity in the absence of an active network connection to a central server. Developers may additionally utilise experimental branches in the version control system repository as creating and destroying a development branch is a non-expensive operation. An ad-hoc workflow and interaction with additional developers is, as a result, simple since merging is a non-difficult operation. The utilisation of ‘feature branches’ with a distributed version control system may improve the perceived quality-of-code over traditional development iterations. From the perspective of the project community, utilising distributed version control system technology is perceived to lower the barrier-to-entry for new contributors and results in a simple merge operation for a local changeset to an upstream development state which previously required significant manual interaction with the version control system (Clatworthy, 2007). Students and inexperienced users of a version control systems may benefit from the simplified project or repository initialisation that is inherent to distributed system. Experience in the utilisation of distributed version control systems allows students to obtain an intuition of the fundamental nature of version control systems. As a result, students may utilise version control systems of a centralised and decentralised nature with a minimised barrier-to-entry (Rocco & Lloyd, 2011).

2.2. Transitioning from Centralised Version Control to Distributed Version Control

In their research paper, “Why Are Software Projects Moving From Centralized to Decentralized Version Control Systems?”, de Alwis & Sillito (2009) identified five anticipated
benefits for an open source project transitioning from a centralised version control systems to a distributed version control system: “To provide first-class access to all developers”, “To support atomic changes”, “Simple automatic merging”, “Improved support for experimental changes” and “Support disconnected operation”. The utilisation of a centralised version control systems requires a bureaucratic component with regards to the provision and revocation of write-access rights to the central repository. With a distributed version control system, each developer has write-access to their clone of repository and are thus able to implement significant modifications without recreating a local centralised system. Therefore, the segmentation of contributors into non-committers and committers is less strict and there exists less formal separation between both groups. Projects additionally wish to avoid the repository corruption issues that they may have experienced using a centralised version control system. As a result, support for atomic modifications was a primary component required in the design and implementation of distributed version control systems. The ability of distributed version control systems to handle frequent merging efficiently was perceived as important since simple merge procedures encourage developers to maintain branch synchronisation with the master branch. Furthermore, the workload for maintainers and master branch committers is typically reduced with the simpler merge procedures introduced by distributed version control systems. Distributed version control systems make branching a non-expensive operation in contrast to a similar operation on a centralised system and, as a result, may perhaps encourage the development of experimental changes by allowing developers to make local work-in-progress commits that may be disregarded if results are undesirable. Thus, disconnected operation and a freedom from the client-server model were perceived as important factors in the design and utilisation of distributed version control systems (de Alwis & Sillito, 2009).

Oezbek & Thiel (2010) explore, in their paper “Radicality and the Open Source Development Model”, discuss transitions from centralised version control systems to distributed version control systems. They perceive open source development as tending toward the encouragement of incremental modifications as opposed to hasty, radical modifications. However, they identify following reasons why the radical transition from centralised version control systems to distributed version control systems have succeeded. For each examined project, the transition was undertaken by the project leadership in a short period of time. The adoption of new tools, however, was spread over a longer period of time and projects observed that contributions utilising the previously employed systems and processes were gradually adjusting to adhere to the newly introduced distributed version control system. There did exist projects that utilised simultaneously the previous and intended systems for a short period of time to allow developers to adapt to the perhaps unfamiliar technology prior to completing the transition. This was observed to reduced the radicality associated with the transition. Additionally, it was observed that developers differ in the ability to adapt to new system, with some developers experiencing the migration as non-difficult but other developers experiencing difficulty. Projects that utilised multiple repositories were seen to initially transition vital sections of the project, followed by non-vital parts. This was observed to result in a simpler transition process for the project maintainers (Oezbek et al., 2010).
de Alwis & Sillito (2009) argue that distributed version control systems better support decentralised workflows that follow the ‘bazaar’ style of open source projects such as Linux kernel (Raymond, 2000). Oezbek & Thiel (2010) observed that projects that transitioned from centralised to distributed version control systems did not always transition the repository structure, and as a result the method for incorporating changes, from a centralised method to a distributed method. However distributed version control systems do not inherently restrict the means by which a system may be implemented, and as a result, such a system may be utilised in a manner more akin to a centralised system and benefit from the other improvements that distributed systems may introduce to the development lifecycle (Rigby et al., 2011).

3. Criticisms

In this section, there shall be conducted a discussion regarding the criticisms and proposed inadequacies of distributed version control systems. The presented discussion points shall be analysed with contradictory assertions from existing literature presented where possible. It is such that there exist no consistently, widely applicable means of managing source code for any non-trivial software development endeavour of significant size and complexity. There shall likely exist specific requirements and needs unique to a given project that dictate the necessary means by which source code control and developer behaviours are influenced. As a result, the criticisms presented in this section are not intended to discount or negate the value of distributed version control systems, nor to extol the value of centralised version control systems but rather to identify areas in which a distributed version control system may be problematic or unsuitable for a given set of project management requirements. It can be shown, however, that many of the criticisms levied against distributed version control systems can be diminished or negated by the use of appropriate governance models.

3.1. Absence of Centralised Server

Koc (2011) states that in the absence of a centralised server, distributed version control systems present a situation in which a backup of the repository Poses a challenge. In the absence of a centralised repository, there may exist multiple repositories at various stages of development that may not have introduced these modifications to the location of the backup procedure. This is a direct consequence of the distributed nature of distributed version control systems. Indeed, there does exist an element of subjectivity to this claim. The absence of a centralised server and repository does not necessarily preclude the absence of a centralised project development location. Rigby et al. (2011) discusses the model in which the Linux kernel is developed. There exists within the linux kernel development community a hierarchical development model whereby there are teams associated with each responsible area, increasing in scope toward the root of the tree. In the case of the Linux kernel, Linus Torvalds, the creator of the Linux kernel, maintains what may be considered as a master or
primary repository through which modifications and developments to the Linux kernel gravitate toward (Rigby et al., 2011). In this model, the governance (Rigby et al., 2011) dictates the potential for centralisation within a distributed software development endeavour. It is interesting to note that this model does not invalidate nor negate the principles and perceived advantages of a distributed software development project. One can employ a hierarchical governance model which a fixed or intended pipeline through which future development and modifications may traverse while encouraging the use of local repositories to track individual and modular development. Rigby et al. (2011) state that in situations whereby each repository evolves in an independent manner and merge the development stage of each repository at such time that it becomes necessary, this is an example of episodic collaboration. Stating that this method of development allows for the potential of parallelised development cycles that can be merged when required (Rigby et al., 2011). Thus, it follows that in such a development model, there need only exist a single backup procedure on the repository at the root of the tree to ensure that the primary project repository may be retrievable. Furthermore, the existence of multiple, distributed local repositories reduce the potential for the existence of a single point of failure, thus further reducing the potential for the loss of the software development endeavour.

3.2. Absence of Understandable Version Numbering System

Koc (2011) cites the absence of an “understandable version numbering system” as one barrier to the comprehension and utilisation of distributed version control systems. This creates a less intuitive user experience due to the potentially unclear methods of tracking modifications including a hash of the changeset or a globally unique identifier as a means of identifying the current state of a software project (Koc, 2011). This may perhaps cause confusion in situations whereby the development effort is distributed with multiple local repositories at various stages of the software development lifecycle. In a centralised system, Koc (2011) argues, “changes are version tracked only when the changes are committed to the server”. While this statement originates in the context of discussing the nature of distributed version control systems and not part of any specific criticism, it is useful to consider how this property of distributed version control systems dictates the traceability and versioning of modifications. In situations where there exists a centralised server containing a single repository, all changes made thus represent the singular state of the software project at a given point in time. However, should this be distributed, there does not exist a single coherent version that can be tracked. This may consequently result in the creation of incoherencies in the merging process should two local repositories be merged with one containing unique modifications to a code base older than that of the merging repository.

In a manner similar to the previous discussion regarding the nature of cloning repositories to additional storage media to ensure persistence in the event of a failure at the site of the repository, the issues regarding a coherent and comprehensible versioning scheme can be mitigated by appropriate use of governance. Considering briefly the developmental hierarchy of the Linux kernel, it follows that when there exists a repository serving as the root of the
software development endeavour, there can exist a coherent versioning system. The Linux kernel development team deploy new versions from the primary repository, that of Linus Torvalds (Rigby et al., 2011). It is from this repository that versioning is applied. As a result, with appropriate governance models applied to the software development lifecycle, it can be shown that there can exist a coherent, systematic versioning convention applied.

3.3. Usability Concerns

Pennington (2006) states that distributed version control systems may be perceived as unwieldy from a user experience perspective, stating further that “[centralised version control systems] may be less flexible but it’s also less confusing”. Rocco & Lloyd (2011), additionally states that in situations where there exist requirements that a version control system be simple to utilise and comprehend and remain unobtrusive to the development cycle, a centralised version control system may be a logical preference. However, in a study of the utilisation of distributed version control systems among introductory computer science students, Rocco & Lloyd (2011) identified that the distributed version control systems were deemed more suitable for utilisation among the sample group studied. Rocco & Lloyd (2011) discuss the notion that centralised version control systems inherit processes akin to that of “client-server architecture and interaction pattern” which may require additional knowledge of these processes and patterns beyond the capabilities of an introductory student. While this observation does not reflect the collective software industry and represents a small sample size, Rocco & Lloyd (2011) do create a case that there may exist barriers to implementing and utilising both centralised and distributed version control systems for software projects and that the barriers may be a consequence of the requirements and restrictions imposed by a given software development project as opposed to a consequence of architecture. Furthermore, Rocco & Lloyd (2011) state that the implementation and utilisation of a centralised version control system can be complicated by the requirement of a server and appropriate administration to allow access to the repositories. This argument is perhaps misguided as there exist centralised, by definition, distributed version control systems that allow the creation of a local repository. However, in the academic context in which the study was conducted, there does exist the possibility that utilising a distributed version control system in a distributed manner allowed for a simplified user experience from the perspective of the faculty staff and students.

3.4. Storage

O’Sullivan (2009) discusses the concern of storage distribution when utilising distributed version control systems to track binary files. In fields such as interactive media development, there may exist the requirement to store binary files that impose a significant storage cost. O’Sullivan (2009) states that on a centralised version control system, this storage overhead is incurred in a single isolated location. However, in such cases that distributed version control systems are utilised, this storage overhead is incurred to the storage mechanisms of each local copy of the repository as, unlike centralised version control systems, each user has
maintains a local repository containing the full history of development activity and modification for the project and thus, the file history of each of the binary files being tracked in the repository (O’Sullivan, 2009). Furthermore, as a direct consequence of the nature of data stored in a binary format, O’Sullivan (2009) states that there are limited means by which modifications to a binary file can be extracted. With a typical, textual source code file, modifications can be trivially extracted and as a result, only those components of the source code file that differ from the original version may be tracked. This is not possible with binary files (O’Sullivan, 2009) and as a result, there exist limitations in the applicable storage minimisation and compression techniques available that may serve to reduce the distributed storage cost of a number of local repositories.

These concerns presented by O’Sullivan (2009) are perhaps the most tangible and quantitative criticisms levied against distributed version control systems. It is certainly such that in the criticisms cited by O’Sullivan (2009), there does exist a significant storage overhead that may perhaps not be incurred with a centralised version control system. O’Sullivan (2009) presents the means by which these issues are handled in a distributed version control system utilising software project, stating that “[distributed version control systems] must rely on social norms (for example, a team policy of only one person ever modifying certain kinds of files”. This approach appears to be indirectly applied to the Linux kernel development cycle whereby there exist a person or persons responsible for individual components. Additionally, from O’Sullivan (2009), it is stated that many software projects typically do not include nor utilise binary files in the development stage. This does not alleviate the incurred storage cost in those projects that do utilise binary files. As a result, means for appropriately tracking binary files in distributed software projects utilising distributed version control systems can be considered as an area for future research and development.

4. Conclusions

From the prior discussion, it can be observed that there exists a significant not only a technological transition required when considering adopting the utilisation of a distributed version control system but also an ideological transition. The adoption of distributed version control systems for a given project is strongly contingent upon the requirements for that individual project such that there can exist no panacea for software development projects. It was shown that in the literature utilised in this research paper that a transition from a centralised version control system to a distributed version control system can perhaps result in a development lifecycle and project structure that is more cohesive with the ideology and practicalities of open source software development. However, affording a lower barrier-to-entry for prospective contributors may adhere to the perceived open state of a software project, there does not exist any fundamental requirement for those projects utilising a distributed version control system to integrate the contributions by those who submit modifications. Thus, it follows that a distributed means of version control does not preclude a
distributed development lifecycle.

The criticisms levied against distributed version controls are typically subjective in nature, reflecting the requirements of a given software development project or personal experience in place of a generalised criticism. This follows, perhaps, from the qualitative nature of the research conducted in this paper. There was no quantifiable criteria for which the success, or lack thereof, of an open source project utilising a distributed or centralised version control such that the effect of the nature of the version control system was immediately apparent. As such, the perceived benefits or costs to a software project incurred from the utilisation of a distributed version control system can only be extrapolated from the adherence to the principles of open source software development when considered as a general case. Should further research be considered on this topic, it would be perhaps necessary to introduce case studies involving participating projects of various complexity, governance and purposes and introduce transitions from a subset of projects from centralised to distributed version control systems and maintain two separate control projects with a centralised version control system and a distributed version control system respectively.

5. References


