IBIMA Publishing
Communications of the IBIMA
http://www.ibimapublishing.com/journals/CIBIMA/cibima.html
Vol. 2011 (2011), Article ID 160480, 22 pages
DOI: 10.5171/2011.160480

Layered Approach to Open Source Software Development Success

Aminat Showole, Shamsul Sahibuddin and Suhaimi Ibrahim

UTM Advanced Informatics School, Universiti Teknologi Malaysia (UTM), KL, Malaysia

Abstract

Open source has emerged as a widely accepted software development phenomenon which has tremendously brought about a significant paradigm shift from traditional software development methodologies such as top down design and stepwise refinement to an unconventional software development approach by means of collaborative software development method among a wide geographically dispersed interested developers and committed project participants while paying less attention to immediate "physical gains". The open source approach focuses on highly diverse views of developer motivations; ranging from ego gratification, ideological satisfaction and gift culture for individual developers and open source motivations may be viewed from spreading the software development risks and associated maintenance costs at corporate organisational level. In this article, a five layered open onion model of open source was broadly examined. Analysis and evaluation were narrowed down to only the initiation layer of the open onion model. Results show that open source success largely depends on the quality associated with successful initiation of the project. Our findings also reveal that the most popular open source license is GPL and that license type has significant impact on project rank. The domain audience has negative impact on project rank and user interface has significantly negative impact on project's domain audience. Open source project topics covered have a significant impact on the domain audience and a negative effect on the user interface. This research has also presented a conceptual framework of open source success tree.

Keywords Software Engineering, Open Onions, Open Source.

Introduction

Software Industry has accepted the fact that changing requirements are simply part of the software development processes. Today, software developments are faced with steadily increasing expectations: software has to be developed faster, cheaper and better. Although user requirements do change middle way at the same time, application complexity increases. Meeting all these demands requires an ability to continuously revamp past codes in order to evolve high quality software.

It could be deduced from (Charles 1992) that quality software is not achievable within reasonable costs and budget time except if it is able to reuse past "reusables". A software artifact that is used in more than one context (projects) with or without modification is considered reusable.

However, in order to revamp an existing software code, it will be required to have full access to the source codes. This way, a software developer could have new functional software evolving quicker, cheaper and with high quality.

 Contact author: Aminat Showole e-maill: aminatshowole@yahoo.com

Various software paradigms have tried to address the issue of evolving quality software faster within reasonable budget. These include but not limited to object orientation and component technology. Open source seems to be the only software development paradigm that allows for free accessibility to the source code for re-modifications and critical study in order to utilize relevant codes and remove irrelevant.

Generally, software reuse is enabled through modular software architectures and the development of generic software components. However, the design of generic components requires substantial investment for a firm that can only pay off in the long run if and when the firm saves development costs through component reuse in software projects. In the software industry, firms that reuse code on more than one project can amortize development costs faster and reduce development time in new projects; (Barnes, Durek et al. 1988), (Barns and Bollinger 1991) and (Banker and Kauffman 1992). Reusing code and components from software libraries also enhances the quality of new software products by allowing for fully tested and debugged software (Knight and Dunn 1998).

In spite of the reported benefits, several studies on software development firms have found that code reuse in software development is problematic and that the success of corporate reuse programs hinges on organizational factors more than on technical factors. In software development firms, corporate reuse programs need to commit an initial investment to (Isoda 1995). On the demand side, it is not surprising to note that many firms and governments have adopted open-source software, since this enables them to reduce costs. However, economists have found it difficult to understand the supply side of open-source innovation, in particular, labour supply (Siegel and Wright 2007). Other suggested possibilities of government adoption of open source in education policy have also been discussed in (Showole, Suhaimi et al. 2008)

Open-source software offers the most astounding range of reusable assets for any software project. Open-source software is available for virtually all activities, runs on every platform, and can be used in almost every business domain for which software is written (Brown and Booch 2002).

Quite a number of researches have described open source quality in different ways. (Aberdour 2007) described open source quality with respect to onion-like arrangements of the open source developers and contributors. Four layers of the onions were presented which are Core team, contributing developers, bug reporters and users. However, the in-depth analysis of the quality related components parts for each of the layers were not presented.

The measurement of project success itself is however elusive. In (Otte, Moreton et al. 2008), it was assumed that projects which were considered successful are those with productive release version, more than two years in the market, whose developer teams consisted of more than five developers and which have a community above fifty participants. Otte however, concluded that the assumption does not reflect the actual project success.

Understanding the structure of software systems can provide useful insights into software engineering efforts and can potentially help the development of complex system models applicable to other domains. (Zheng, Zeng et al. 2008) took a study on open source package and inter-package dependencies as a way of understanding the software structures using dependency graphs in the analysis.

This study suggests that in order to understand the structure of software system, it would be required to understudy and analyse processes surrounding the actual development. Open onions is a framework which can be used to improve the understanding of open source project initiation success.

Open source successes have been largely attributed to the surrounding "people" in connection with the open source development. In (Mockus, Fielding et al. 2002), it was hypothesized that "In successful open source developments, a group larger by an order of magnitude than the core will repair defects, and a yet larger group (by another order of magnitude) will report problems". It was however reported by Audris in subsequently that in the Apache study, the group that adds new functionality is larger than it was expected.

Various onion models have been implemented in many aspects of software and network security (Goldschlag, Reed et al. 1999), (Paul, David et al. 1997), (Yoshifumi and Tatsuaki 2008), (Joan, Aaron et al. 2007), (Malik Ahmad Yar and Darryl 2008). A few research on onion methodology have been presented in the field of open source, mostly in the representation of the arrangements of team structure (de Sousa, Balieiro et al. 2009), (Crowston and Howison 2006) and (Crowston, Annabi et al. 2004). However, the onion-like arrangements of the open source community are yet to be duly explored in terms of setting-up the onion-framework that could form a yardstick for determining the incumbent characteristics that could determine the success or failure of "any" open source development.

The open onions technique, as briefly introduced in (Showole, Suhaimi et al. 2008) is an approach with five layers of the open onions which tends to improve on the existing onion models of open source. The remainder of this article is organized as follows. The main purpose of this article is presented in the next section. Section 3 is focused on preliminary studies. In section 4, the Open Onions approach is presented while section 5 detailed the experimentation. Results summary is presented in section 6. The concluding part is section 7 and the last part, section 8, points at future directions of this research.

Purpose of the Paper

This paper is an extension of our earlier work on open onions model as presented in (Showole, Suhaimi et al. 2008). In this approach, open source project developments have been broken down into five layers of open - onion, which were identified to be the critical factors affecting the development success of long-term sustainability of open source development projects. Table 1 is the open onions descriptive table that shows all the five layers of the model.

It was however discovered that the way an open source project is started has a far-reaching implication on the resulting success/failure of such a project. Ten highly ranked (based on Sourceforge.net ranking of March/April 2009) open source projects were investigated. Data was collected from sourceforge.net repository and analysed with SPSS. Although our approach is based on five fundamental open onions layer, this paper addresses the first layer of the open onions – the project initiation layer.

The initiation layer identifies the success factors in order to achieve a quality open source development project. This means that the initiation strategy goes a long way in affecting the quality of such a project. The first layer of the open-onions, i.e. the project initiation layer is modelled extensively. Table 1 is the open onions description table. It shows all the five open onions layers with their description.

Table 1: Open Onions Description

labels	Description
A	Project Initiation Layer
b	The Maintenance Layer
C	Developers/ Users Layer
d	Observers/Non-Interest Group Layer
E	External Layer

Preliminary Studies

The quest for providing technologies for building software systems faster, with lower cost and higher quality has led to the advances in software technologies such as component based system development, Object Orientation, Service Oriented architecture, software reuse and open source. One of the most important benefits that reuse or revamp delivers is quality. Among all powerful software technologies available today, software reuse is a fundamental approach to accelerate the production of high quality software and this is achievable by its ability to provide the benefit of faster, better and cheaper software development processes. Reuse standards are emphasized in (McClure 2001).

As could be observed, most of the earlier software development technologies have some shortcomings which could be addressed by open source. With object orientation, there is a need for compositionality. That is, OO languages do not support the specification of an explicit typed "Inheritance Interface" for programmers who develop subclasses (Meijler and Nierstrasz 1995). Often, object oriented problems are complete specifications of objects, attributes, structures services and subjects and the degree to which members within a class are related to one another is often difficult to identify.

Another short coming of object orientation is that system modification, maintenance and testing can be difficult because of inheritance and behavior overriding. Replacement of object with a new object that implements changes to the business may impact all other objects that have inherited properties of the replaced object and this may lead to excessive testing of the whole system (Kiczales, Lamping *et al*. 1997).

However, Component Technology also exhibits certain setbacks in the area of determining the level of cohesion and coupling of components. It is also difficult for developers to adapt a component to a new platform if it were not developed for that platform (Rizwan Jameel Qureshi and Hussain 2008). Other difficulties associated with this technology are the architectural mismatch or architectural complexity which results in some other component disadvantages. For example, customization and integration of already developed software components according to the requirement of the new application is a major issue in component technology.

Open source is an alternative paradigm, which encourages open access to source codes for further reuse and modifications. Volunteers who are geographically dispersed usually produce software developed under this approach. Open source has become a subject of focus lately in the software engineering world. It is a collaborative development paradigm characterized by various volunteer whose developers and users broadly geographically dispersed.

It is important however to note that not in all cases that open source developers work for free. Substantial evidence shows that most

developers are volunteers except for some corporate organisations who belief in open source economic model; they go as far as employing paid staff to work on open source projects for such organisations.

Numerous perceived disadvantages of open source model have been identified. Of particular reference is the QinetiQ report where (Peeling and Satchell 2001) discussed that Open Source developers tend to be very passionate about technical issues (in terms of coding).

The study of (Feitelson, Heller et al. 2006) was focused on considering open source quality from the number of downloads criteria only. To this end, we revealed that success of open source development projects goes beyond the technical issues (in terms of coding) alone. Open source development projects also transcend the number of downloads alone as a yardstick for measuring the success of such projects especially at early stages of the project developments.

Nature of the Problem

Open source is obviously a subject for projects in Universities and research institutes. There is a growing interest among Governments in using open source as a mechanism for exploiting research results. The research community gives open source developers free access to a large community of the brightest and freshest minds (Peeling and Satchell 2001). Human engineering artistry creates computer, computation and information systems that enhance everyone's daily lives. The complexity associated with these systems (PC's, Laptops, palm tops, 3D Animation, avatars, the web technology itself, DNA computing, etc) conveys affluence of computing functionality.

The complexities however, also make it difficult to predict even the original system behaviour, let alone anticipating the emergent behaviour of multiple interacting systems. This could then be imagined in the case of open source development where there is no one correct way to run an open-source project in which there are thousands of developers submitting thousands of patches to a single software development project.

Successful open-source projects can be quite different from each other. Some, such as Apache, are very democratic and volunteers are welcome to participate in all activities. Others, such as MySQL, where almost all of the developers work for one company, primarily do their development internally and then release the results; users and developers engage with each other to report bugs, request new features, and generally discuss the software, but development happens less visibly. There are even some projects that do not have a community at all, but consist of just a web page where new versions are made available for people to download and perhaps an email address where comments can be sent (Goldman and . 2005). This makes it clear why traditional software engineering models are not suitable for open source development. In traditional software engineering model, due to budget overrun and late delivery, many projects are forced to be aborted or are missing implementation of some components or are delivered without thorough debugging.

Related Work on Open Source Research

Numerous issues could be identified within the context of the open source development model. Some of which are the Profitability, Security, collaborative, testing, interoperability, legal issues and acceptable software engineering approach for open source development. For the purpose of defining the research boundary, we would be focusing on identification of suitable software process model to support open source development.

Notable academic research activities have been conducted in the field of open source. It was observed from (Madey, Gao et al. 2003), (Gao and Madey 2007), (Xu, Gao et al. 2005;

Jin Xu and Madey 2006), (Oostendorp 2009), (Rajdeep, Gary et al. 2006) and (Zheng, Zeng et al. 2008) that most of the research activities are based on the social network analysis of open source development. (Feitelson, Heller et al. 2006) conducted their research based on the distribution downloads as a yardstick for a successful open source project. (Timo and Virpi 2005) focused on the maintenance process of open source as a way of mapping the maintenance activities of the chosen open source case studies to the existing ISO/IEC maintenance standards. (Scotto, Sillitti et al. 2007) conducted his research on mining the open source repository. In (Zhao and Elbaum 2003), surveys on open source quality assurance activities were mainly based on testing phases of the software development where it was reported that there is a need for more research on identifying the most efficient procedure to deploy and carry out quality assurance activities in open source.

According to (Peeling and Satchell 2001), most investors do not fully understand the open source model. The commercial models have well- defined profit motive, yet they are still developing and consequently unpredictable. Most of the problem with some software that fail the market acceptability is that the development could not have been funded continuously unlike few proprietary software merchants e.g. Microsoft Incorporation which can continuously fund its products.

Open source however solves this problem by having a zero cost, base meaning that License fees are usually at zero cost except for maintenance and other profit models surrounding open source, so running out of capital is not a problem as long as the group of developers maintains their interest; the project can keep on functioning. Also, the ability for users to acquire complete software without having to sign licenses and make financial case to their management; aids initial take off.

The open source is thus an effective practice which had evolved as a set of customs, transmitted by imitation and example, without the theory or language to explain why the practice worked. Raymond (Steven 1999) revealed that lacking open source theory and language hampered the open source community in two ways explaining that it would be difficult to think systematically to improve the development method and it would be very difficult to explain or show the method to anyone else. Most times, open source development is usually described based on case studies; for example in (Mockus, Fielding *et al*. 2002). However, (Aberdour 2007) shows that it is still very difficult to understand why successful open source projects attain high quality. This research explains why and points at the processes involved in the initial take-off of a quality open source project.

Literature on Related Onions Models

The Open onions approach is based on onion model as could be observed from the model name. This article will not be complete without a review of previous onions models. This research has therefore investigated the use of onion models in varying contexts. It was discovered that the use of onion model transcends only the field of software engineering. In the field of chemical engineering, for instance, in (Dominic Chwan yee Foo 2005), onion model was used to simulate chemical process synthesis, where it was stressed that onion model is an alternative way of presenting the hierarchical approach of chemical process design.

There are also repeated uses of onion models in the information security field where each onion layer is said to be a security enforced layer and it makes organizational network security much tighter than the traditional lollipop model which is based solely on building a single wall around an object of value. This implies, according to (Rhodes-Ousley., Roberta. et al. November 10, 2003),

once the lollipop security firewall is attacked, the organizational valuables inside are completely exposed. This is because the lollipop premier security model does not provide different levels of security.

In other security related instances, onions routing has been identified as an effective approach of solving security problems over a simple application of cryptography within a packet-switched network (Paul, David et al. 1997; Goldschlag, Reed et al. 1999). Various researches on onion models have also been conducted especially in the field of computer communications networks. A number of researchers have therefore adopted the use of onions models in addressing core security issues in computer networks (Benjamin, Oliver et al. 2009), Malik (Malik Ahmad Yar and Darryl 2008). Adoption of onion model in implementing black-box model (Joan, Aaron et al. 2007), addressing issues relating to malicious return path in a communications network (Yoshifumi and Tatsuaki 2008) and onions methodology adoption to specify and implement abstract data types (ADT) in a data dominant system (Arun and Paul 1994). The open onions approach is a unique approach to achieving open source quality control by having a layered methodology of open source project development (Showole, Suhaimi *et al*. 2008).

The Open Onions Approach

Quality views are very diverse ranging from transcendent view, to product view, user view, manufacturing view and value-based view (Koch and Neumann 2008). Product metrics however, describes the developers Open onions as represented in Figure 1 as a layered approach to open source development. This approach strives to address open source quality from two perspectives, the transcendent view and the product point of view. This is achieved through the use of statistical method of quantifying software development. (Pressman and Scott 2005) and (Roger S. 2001) have presented valuable benefits of using statistical approach in software quality managements.

It is note-worthy that organisations wanting to invest in open source would want to know the estimated amount of resource they must invest in order to achieve the necessary deliverables (Jai 2005) and this research points at the estimated amount of resources (in terms of domain audience, expected user interfaces, required topics to be covered and the suitable open source license type) necessary for a quality open source.

It is a portable approach which doesn't depend on any software architectural complexity. It scales well in the area of platform and programming language independence because it does not focus on the internal code representation of the software development. Figure 1, with the summary presented on table 1, depicts that Layer 'a' represents the open source project initiation layer which is the main focus of this article.

Layer 'b' is the project maintenance layer which encompasses the core initiators and other technical personnel responsible for acceptance/rejection of submitted patches into the main stream of the core of the project developments including other technical personnel responsible for acceptance/rejection of submitted patches into the main stream of the core of the project developments.

Layer 'c' represents the developers which are also part of the group of end users, where-in all project contributors fall with this group; including the users/core developers. Layer 'd' is the observer layer for those who are not necessarily interested in contributing codes but would like to follow most of the open source development in order to be well informed and updated. They may also wish to play around with the codes in order to learn from its internal workings.

The last layer 'e' represents the external layer. At this layer, we consider various

organisations that actually get hold of the raw codes of open source projects and adapt it to suite their organisational needs whereby evolving new high quality software cheaper and faster. Such organisations may not necessarily release the new resulting software 'open'. The new package therefore becomes a guided asset of such organisation. Table 2 gives the full description of the open onions detailed approach, Table 2 gives the details of the open onions.

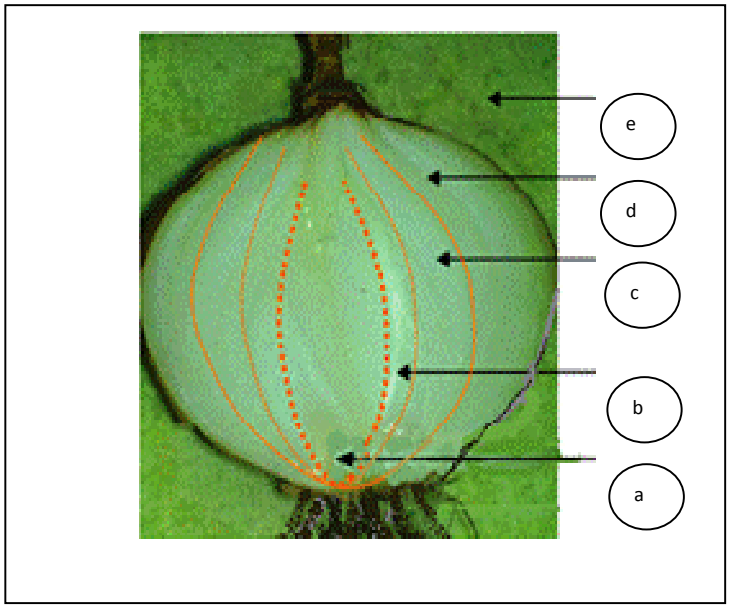

Figure 1: Open Onions Layers

Table 2: Open Onions Details

Layer	Description	Details
A	Open source project initiation layer	This is the open source project initiation phase. The open source project initiator(s) is (are) the person(s) who started the project and eventually are usually referred to as the project leaders among the increased number of core maintenance group.
B	Project maintenance layer	project maintenance layer which encompasses the core initiators and other technical personnel' responsible for acceptance/rejection of submitted patches into the main stream of the core of the project developments
c	Developers	This represents the developer's layer. The members of this group are users with varying degree of authority in the whole project. They also include the group of end users.
d	Observer layer	Observer layer for those who are not necessarily interested in contributing codes but would like to follow most of the open source development in order to be well informed and updated. They may also wish to play around with the codes in order to learn from its internal workings.
E	External layer	Various organisations that actually get hold of the raw codes of open source projects and adapt it to suite their organisational needs whereby evolving new high quality software cheaper and faster. Such organisations may not necessarily release the new resulting software 'open'. The new package therefore becomes a guided asset of such organisation

Empirical studies from literature (DeKoenigsberg 2008), (Fielding 2005), (Devanbu 2008), (Lakhani and Eric 2003), (Aberdour 2007) and (Crowston and Howison 2003) have shown that the successes of open source development projects largely depend on some success factors such as developer skill, programming language support, domain audience addressed, natural language support, to mention but a few. We have packaged all these factors together to form the open source success tree in figure 2.

This research has modeled with fish bone, substantial aspects of the success criteria necessary for starting off an open source development project. "Open source success tree" in Figure 2 points at various factors to be considered while an open source project is initially set up and gradually finds its feet in the high-ranking open source project domains.

First, which is the most important, is to learn from others. This further implies that the community of open source around a given project can only be built and sustained by constantly meeting and communicating with other project leaders. It involves the project leader(s) joining and contributing to at least one on-going open source project, and to search for similar project.

The second factor is to develop leadership and communication skills. Here, it is expected that project developer defines project goals and vision, decision making roles, develops project rules and sets up leader activities.

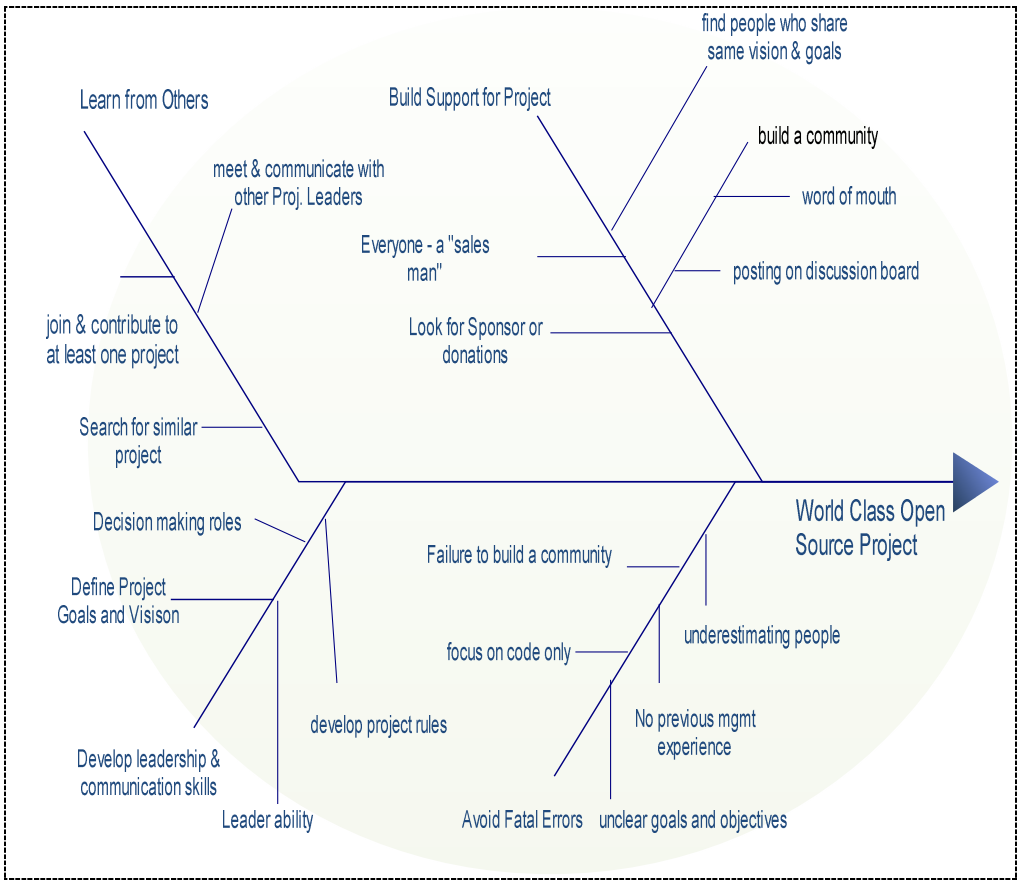

Figure2: Open Source Success Tree

Thirdly is the need to build support for the project. At this stage, the project leaders 'find' people who share same vision and goals, allow every project participant to assume a 'sales man role', quest for sponsor and donations fall within this level; community building by either word of mouth or postings on the discussion boards are crucial at this stage.

Lastly is to avoid fatal errors. This implies that the core project developers must avoid an unclear goals and objectives of the project. It is always required to have previous management skill before embarking on large-scale open source development involving numerous developers to be managed across the project life cycle. It would be a fatal error to underestimate people that are project members. Failure to build a community around an open source project would make such project unpopular which has resulted in negative effect on the project ranking as discussed in section 3 of this article. To yield a required level of project success, it is not recommended to focus solely on code development without considering basically all aspects of the successful take-off of such an open source project. In Figure 2, the open source success tree shows the required necessary aspects of building a strong community around an open source development project.

Experimentation

The open-onions project initiation layer was modeled statistically with SPSS tool. Open-

onion success tree in Figure 2 was used in depicting all component parts of the layered model.

Various aspects of the critical factors were tracked by this model using case studies of ten highly ranked open source projects. Data for the ten open source projects was obtained from sourceforge.net data repository. These case studies were selected from the sourceforge.net software map, under the software development sub-categorization. This is because our research is focused on open source development projects. Sourceforge.net, the largest open source repository, was queried to extract relevant development details on each of the case studies.

Initial Problem Formulation and Conceptualization

The identified critical issues under study were: user interface, license type, topics covered and domain audience for each of the ten case studies. The result was analysed and presented using SPSS statistical analysis tools. In table 3, the necessary parameters that could affect the success and quality of open source project initiation were considered. It was achieved by identifying the first layer of our open – onions model. This first layer, open source project initiation layer, was characterised by various incumbent properties identified from our literature review and case studies.

The 'domain audience' describes types of target audience for each of the open source projects; for example, manufacturing and IT. 'User interface' represents the type of user interfaces being supported e.g. web based and xwindow system; topics covered are the relevant topics being covered by the project e.g. accounting, point of sale and project management, and the License type indicates the type of license(s) binding on the use and adoption of the project e.g. GPL and BSD

Table 3: Project Initiation Parameters

Projects(rank)	domain (Audience)	user interface	topics covered	Licence types
1	6	1	6	2
2	3	1	3	1
3	6	1	3	1
4	2	2	2	1
5	1	5	2	1
6	2	2	2	1
7	2	4	2	3
8	3	2	5	4
9	3	1	3	1
10	3	3	2	5

Project Rank

The project rank is comprised of ordinal data set. The rank order is as obtained from sourceforge.net and it is calculated based on project traffic, development, and communication variables.

Figure 3: Project Rank

Traffic =	**[(log(last_7_days_downloads + 1) / log(highest_7_day_downloads + 1))**
	+ (log(last_7_day_logo_hits + 1) / log(highest_7_day_logo_hits + 1))
	+ (log(last_7_day_site_hits + 1) / log(highest_7_day_site_hits + 1))
	] / 3
=	
	[(log(2,254 + 1) / log(2,191,506 + 1))
	+ (log(22,966 + 1) / log(3,813,755 + 1))
	+ (log(259,460 + 1) / log(1,350,986 + 1))
=	] / 3
	0.69152880089125
Development =	
	[(log(last_7_days_scm_commits + 1) / log(highest_7_day_commits + 1))
	+ (100-days_since_last_file_release / 100)
	+ (100-days_since_last_admin_login / 100)
	] / 3
=	
	[(log(56 + 1) / log(+ 1))
	+ ((100 - min(100, 38)) / 100)
	+ ((100 - min(100, 0)) / 100)
=	] / 3
	0.68048794943448
Communication =	
	[(log(last_7_days_tracker_entries + 1) / log(highest_7_day_entries + 1))
	+ (log(last_7_days_ML_posts + 1) / log(highest_7_day_ml_posts + 1))
	+ (log(last_7_days_forum_posts + 1) / log(highest_7_day_forum_posts + 1))
	] / 3
	[(log(33 +1) / log(481 + 1))
	+ (log(0 + 1) / log(1 + 1))
=	+ (log(232 + 1) / log(690 + 1))
	] / 3
	= 0.46817588056105
Total Score	= (Traffic + Development + Communication) * 20,000,000
	= 36,803,853

Domain Audience

The details in Table 4 are the expanded view of the domain audience. The case studies have shown that at least 3 popular domain applications would be required for successful open source development project. From this study, it was discovered that *Developers, Information Technology and End-user/Desktop Application* domains are the most relevant.

Table 4: Domain Audience Analysis across the Case Studies

Domain Name	Domain Frequency
Developer	9
IT	4
End User/Desktop	4
System Administrators	2
Quality Engineers	2
Manufacturing	2
Finance/Insurance	2
Customer Service	2
Education	1
Non-Government Organization (NGO)	1
Others- Not Specified	2
Total	31

It is therefore evident that the open source audience is mostly software developers' experts as could be deduced from Table 4 above, nine out of the ten case studies have software developers as their main audience. This therefore influences the type of domain audience that needs to be targeted by any successful open source project since quality software could be attributed to end-user satisfaction. Literature studies have shown that overall quality of a software product has a direct relationship to the user satisfaction.

(Glass 1998) has developed an intuitive relationship between the user and the different quality attributes which says:

User satisfaction = compliant product + good quality + delivery within budget and schedule

High quality software according to (Wolfgang, Stefan et al. 2005) is typically defined by quality attributes like customer satisfaction, which is mainly determined by projects being on budget and time which is key priority over other factors.

In this paper, statistical software quality assurance technique Pressman (Roger S. 2001) analysis has been adopted in order to categorize and identify quantitative open source software development quality attributes.

Open Source Licence Type

Table 5 shows the frequency analysis of open source licenses.

Table 5 : Frequency Table for License Types

License Types		Frequency
Valid	GNU Public License (GPL)	6
	Mozilla Public License	1
	Lesser GNU Public License(LGPL)	1
	Barkely (BSD)	1
	Eclipse Public License	1
Total		10

The result from Tables 2 and Table 5 above shows that the most important License for a highly rated open source development projects is GPL License. All other licenses are of relatively lower priority as evident from the two tables. Although, all the licenses adopted by the ten projects fall within the category of Licenses that are popular and widely used or with strong communities, GPL is most prominent as observed on Table 5.

GNU Public License (GPL) is a free software license written by Richard Stallman in the mid-80s. This license pioneered a concept known as copyleft. The GNU General Public License (GPL) is a widely used free software license, originally written by Richard Stallman for the GNU project. The GPL is the most popular and well-known example of the type of strong copyleft license that requires derived works to be available under the same copyleft. In ensuing copyleft provisions, this means that when modified versions of free software are distributed, they must be distributed under the same terms as the original software. Thus, all enhancements and additions to copylefted software must also be distributed as free software. This is sometimes referred to as "share and share alike".

This requires that developers who use GPL code in their product must make the source code available to anyone, including when they share or sell the object code. In this case, the source code must also contain any changes the developers may have made. However, if GPL code is used but not shared or sold, the code is not required to be made available and any changes may remain private. This permits developers and organizations to use and modify GPL code for private purposes without being required to make their changes available to the public.

Supporters of GPL claim that by mandating that derivative works remain free, it fosters the growth of free software and requires equal participation by all users. Hence, scholars and advocates struggle to articulate the legal ground work that makes the GPL license enforceable (Bornfreund 2005).

User Interface

Table 6 suggests that Open source developments are usually web-based. Highest frequency of 4 is associated with 'web based' as the most prominent interfaces. Next in the rank is the win 32 followed by java/java swing which shows that JavaScript, web based and win32 are most appropriate. It therefore implies that the most important user interfaces for a quality open source development project will have to either be web based combined with or without java swing and win32 user interface.

Table 6: User Interface Frequency Analysis

User Interface	User Interface Frequency
web based	4
Java/ java swing	2
win 32	3
Eclipse	1
xwindow sys (x11)	1
MacOS X	1
SDL	1
Command line	1

Topics Covered

Evidences on Table 7 show that projects with at least 2 topics have the highest frequency of 5 out of the ten projects under survey while next in rank is that 3 project have covered 3 topics. On the average, it shows that 3topics could be covered by each project at most and 2 topics at best for quality, in terms of user acceptability as obtained from the download volume and project ranking. open source development projects.

Table 7: Topics Covered

Topics covered		Frequency
Valid	2 topics	5
	3 topics	3
	5 topics	1
	6 topics	1
Total		10

The two topics for a quality open source development project could be any two out of Software Development, Accounting, and ERP as analysed in the Appendix 1 below.

Results

In order to validate our open-onion model, we have used 10-highly ranked source forge project as earlier reported and Table 8 is a Pearson correlation results for the linear data sets, domain audience, user interface and topics covered. From the table, it was evident that in open source development projects, the user interface impacts negatively on the domain audience. The lower the user interface variable, (e.g. 1 ≡ web based, appendix 2) the higher the number domain audience in terms of service industry to support (e.g. Manufacturing, consumer service, finance and insurance industry). Please refer to table 2 for further clarifications.

Table 8: Pearson Correlation Statistics

		domain Audience	user interface	topics covered
Domain Audience	Pearson Correlation	1.000		
	Sig. (2-tailed)			
	N	10.000		
User Interface	Pearson Correlation	-.653*	1.000	
	Sig. (2-tailed)	.041		
	N	10	10.000	
Topics Covered	Pearson Correlation	.661*	-.545	1.000
	Sig. (2-tailed)	.037	.103	
	N	10	10	10.000

*. Correlation is significant at the 0.05 level (2-tailed).

The sig. (2-tailed) implies that the P value is significant at 0.05 (P≤0.05). Meaning that the acceptable confidence level should not be less than 95%.

For this analysis, user interface has a negative correlation with domain audience, the lower the number of domain audience, the higher the required user interfaces and vice versa. This is with a high confidence level of 95.9%. The topics-covered has a significantly positive effect on domain audience at a higher confidence level of 96.3%. However it was discovered that topics-covered does not have any significant effect on user interface and vice versa.

The License types have been categorized broadly into two thus: GPL and Others as shown on Table 9. This is because GPL license ranks highest in the frequency analysis of Table 5, meaning that it is the most popular license amongst all license types, based on these case studies.

Table 9: Licenses Analysed

License Types	GPL	Others
AVG Domain Audience	2.8	3.5
AVG User Interface	2.7	3.3
AVG Topics Covered	2.5	3.8

GPL license was therefore analysed based on its relativity to average domain audience, average user interface and average topics covered across the board for the ten projects. It was discovered that projects with average domain audience of 2.8 would be ideal. The average user interface required would be 2.7 and the average topics covered for a relevant level of acceptance would be 2.5. Projects with averages above these ranges are likely not going to be popular based on these results.

In order to ease the analysis, we have re-categorized the Project ranking into high rank (upper 5) and Low rank (lower 5) Table 10.

Table 10: Analysis by Project Rank

Project Rank	High(upper 5)	Low (Lower 5)
AVG Domain Audience	3.6	2.6
AVG User Interface	2.6	3.2
AVG Topics Covered	3.2	2.8

The higher the license variable gives rise to a higher project rank and vice versa. That is for license code of 1≡ GPL for example, in appendix 1, then we expect to have higher project ranking.

Meanwhile, the domain audience has impact on the project ranking. The higher the number of audience, the better the chances of such projects ranking high. Topics covered are also observed to have significant impact on the Domain audience. License type has effect on the project ranking. The higher the license variable gives rise to a higher project rank and vice versa. That is for license code of 1≡ GPL for example, in appendix 1, then we expect to have higher project ranking.

As evident from Table 8, the topics covered have significant effect on domain audience, meaning that the more the topics which are covered, the higher the expected audience resulting in higher project community building. The user interface also impacts negatively on the topics covered. Meaning that in order to cover more topics, it is needed to reduce the number of user interfaces. For example, web based interface tends to support more project topics than other interfaces.

Conclusion and Future Work

In this article, we summarized the on-going work on various aspects of open source research. In contrast to most work in the field of open source, our approach is focused on the use of open onion model to improve the understanding of open source development.

The contributions of this article include the presentation of open-onion framework, which compensates the lack of adequate understanding of the various components of the open source development, and identifying the correlations between open source project parameters such as Domain Audience, User interface, Topics covered and License types.

Future research includes the validation of open source success tree and all the layers within the open onion model using Delphi's approach

Reference

Aberdour, M. (2007). "Achieving Quality in Open-Source Software," *Software, IEEE* 24(1): 58-64. Arun, P. G. and C. G. Paul (1994). Onion: a methodology for developing data-dominant systems from building blocks. Proceedings of the conference on TRI-Ada '94. Baltimore, Maryland, United States, ACM.

Asundi, J. (2005). "The Need for Effort Estimation Models for Open Source Software Projects," Proceedings of the fifth workshop on Open source software engineering. St. Louis, Missouri, ACM.

Banker, R. D. & Kauffman, R. J. (1992). "Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study," *SSRN eLibrary*.

Barnes, B., T. Durek, et al. (1988). 'A Framework and Economic Foundation for Software Reuse. Software Reuse: Emerging Technology,' *IEEE Computer Society Press*: 77-88.

Barns, B. H. & Bollinger, T. B. (1991). "Making Reuse Cost-Effective," *Software, IEEE* 8(1): 13-24.

Bornfreund, M. (2005). 'OpenSource Law,' Open source legal issues DOI: http://fr.creativecommons.org/articles/canada.htm

Brown, A. W. & Booch, G. (2002). "Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors," *Software Reuse: Methods, Techniques, and Tools*: 381-428.

Crowston, K., Annabi, H., Howison, J. & Masango, C. (2004). "Effective Work Practices for Software Engineering: Free/Libre Open Source Software Development," Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research.

Crowston, K. & Howison, J. (2003). "The Social Structure of Open Source Software Development Teams," *OASIS 2003 Workshop* (IFIP 8.2 WG).

Crowston, K. & Howison, J. (2006). "Assessing the Health of Open Source Communities," *Computer-IEEE Computer Society*- 39(5): 89-89.

Fielding, R. T. (2005). 'Software Architecture in an Open Source World,' Saint Louis, MO, United states, *Institute of Electrical and Electronics Engineers Computer Society*.

Foo, D. C. Y., Selvan, Z. A. M. M. & McGuire, M. L. (2005).' Integrated Process Simulation and Process Synthesis,' *Chemical Engineering Process (CEP) Magazine*.

Gao, Y. & Madey, G. (2007). "Towards Understanding: A Study of the Sourceforge.Net Community Using Modeling And Simulation," *The Spring Simulation Multiconference.*

Glass, R. L. (1998). "Defining Quality Intuitively," *Software, IEEE* 15(3): 103-104, 107.

Goldman, R. & Gabriel, R. P.. (2005). Innovation Happens Elsewhere Open Source as Business Strategy, *Morgan Kaufmann Publishers, Elsevier*.

Goldschlag, D., Reed, M. & Syverson, P. (1999). "Onion Routing for Anonymous and Private Internet Connections," *Communications of the ACM* 42(2): 39-47.

Grewal, R. Lilien, G. L. & Mallapragada, G. (2006). "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems," *Management Science* 52(7): 1043-1056.

Isoda, S. (1995). "Experiences of a Software Reuse Project," *Journal of Systems and Software* 30(3): 171-186.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. & Irwin, J. (1997). "Aspect-oriented Programming," *ECOOP'97 — Object-Oriented Programming*: 220-242.

Knight, J. C. & Dunn, M. F. (1998). "Software Quality through Domain-Driven Certification," *Annals of Software Engineering* 5: 293-315.

Koch, S. & Neumann, C. (2008). "Exploring the Effects of Process Characteristics on Product Quality in Open Source Software Development," *Journal of Database Management* 19(2): 31-57.

Koponen, T. & Hotti, V. (2005). "Open Source Software Maintenance Process Framework," Proceedings of the fifth workshop on Open source software engineering. St. Louis, Missouri, *ACM*.

Krueger, C. W. (1992). "Software Reuse," *ACM Computing Surv*yes 24(2): 131-183.

Lakhani, K. R. & Hippel, E. V. (2003). "How Open Source Software Works: "Free" User-to-User Assistance," *Research Policy* 32(6): 923-943.

Madey, G., Gao, Y., Tynan, R., Hoffman, C. & Freeh, V. (2003). "Agent-Based Modeling and

Simulation of Collaborative Social Networks," *Americas Conference on Information Systems*, AMCIS 2003, Scientific commons. (2008).

Khan, M. A. Y. & Veitch, D. (2008). "Peeling the 802.11 Onion: Separating Congestion from Physical Per," Proceedings of the third ACM international workshop on Wireless network testbeds, experimental evaluation and characterization. San Francisco, California, USA, ACM.

Manabe, Y. & Okamoto, T. (2008). "Anonymous Return Route Information for Onion Based Mix-Nets," Proceedings of the workshop on Applications of private and anonymous communications. Istanbul, Turkey, *ACM*.

McClure, C. (2001). "Software Reuse: A Standards-Based Guide," *Wiley-IEEE computer society press*.

Meijler, T. D. & Nierstrasz, O. (1995). "Beyond Objects: Components," Languages et Modeles a Objects, A Napoli (Ed.).

Mockus, A., R. T. Fielding, et al. (2002). "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Trans. Softw. Eng. Methodol*. 11(3): 309-346.

Oostendorp, N. (2009). "Using Networks to Visualize and Understand Participation on SourceForge.net," *project. Final Year Project Report*.

Otte, T., Moreton, R. & Knoell, H. D. (2008). "Applied Quality Assurance Methods under the Open Source Development Model," *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International, IEEE Computer Society*.

Paul, F., Goldschlag, D. M. & Reed, M. G. (1997). "Anonymous Connections and Onion Routing," Proceedings of the 1997 *IEEE Symposium on Security and Privacy, IEEE Computer Society*.

Peeling, N. & Satchell, J. (2001). "Analysis of the Impact of Open Source Software," *Technical Report*, QinetiQ2001.

Pressman, R. S. (2001). Software Engineering: A Practitioner's Approach, *McGraw-Hill* 207-210.

Pressman, R. S. (2005). Software Engineering: A Practitioner's Approach, *McGraw-Hill Education*.

Raymond, E. S. (1999). The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, Cambridge, MA,.

Rhodes-Ousley., M., B. Roberta., et al. (November 10, 2003). Network Security, The Complete Reference, *McGraw-Hill*.

Qureshi, M. R. J. & Hussain, S. A. (2008). "An Adaptive Software Development Process Model," *Advances in Engineering Software* 39(8): 654-658.

Scotto, M., Sillitti,A. & Succi, G. (2007). "An Empirical Analysis of the Open Source Development Process based on Mining of Source Code Repositories," *International Journal of Software Engineering and Knowledge Engineering* 17(2): 231-247.

Showole, A., Ibrahim, S. & Sahibuddin, S. (2008). "Industrial Application Development with Open Source Approach," The Third International Conference on Software Engineering Advances, 2008. ICSEA '08. , *IEEE*.

Showole, A., Suhaimi, I., et al. (2008). 'Prospects of Open Source Adoption in Education Projects in Nigeria,' *International Journal of Arts and Science (IJAS)*.

Siegel, D. S. & Wright, M. (2007). "Intellectual Property: The Assessment," *Oxford Review of Economic Policy* 23(4): 529-540.

Xu, J., Christley, S. & Madey, G. (2006). "Application of Social Network Analysis to the Study of Open Source Software," The Economics of Open Source Software Development. P. J. H. S. Jurgen Bitzer, Elsevier B.V.: 247 - 270.

Xu, J., Gao, Y., Christley, S. & Madey, G. (2005). "A Topological Analysis of the Open Souce Software Development Community," Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 07, *IEEE Computer Society*.

Zhao, L. & Elbaum, S. (2003). "Quality Assurance under the Open Source Development Model," *Journal of Systems and Software* 66(1): 65-75.

Zheng, X., Zeng, D., Li, H. & Wang, F. (2008). "Analyzing Open-Source Software Systems as Complex Networks," *Physica A: Statistical Mechanics and its Applications* 387(24): 6190-6200.

Zuser, W., Heil, S. & Grechenig, T. (2005). "Software Quality Development and Assurance in RUP, MSF and XP: A Comparative Study," Proceedings of the third workshop on Software quality. St. Louis, Missouri, *ACM*.

Appendices

Appendix 1: Topics Covered Expanded

Project no	Topics covered					
1	code generators	Project Mgmt	Point-of-sale	Accounting	ERP	CRM
2	Software Development.	Enterprise	AJAX			
3	Object oriented	ERP	Accounting			
4	Software Development	text editor				
5	Interpreter	game/ entertainment				
6	Version Control	File management				
7	Quality Assurance	Build tool				
8	build tools	code generators	Compilers	debuggers	interpreters	
9	Software Development	Dynamic content	site management			
10	Testing	Framework				

Appendix 2 User Interface Coding

user interface	Code
web based	1
web based & java swing	2
win 32	3
Command line	4
java & eclipse	5
xwindow sys (x11), win 32, PDA, Cocoa(MacOS X), SDL	6

Appendix 3 Licence Type Coding

license Type	**Code**
GNU Public License (GPL)	1
Mozilla Public License	2
Lesser GNU Public License (LGPL)	3
Barkley (BSD)	4
Eclipse Public License	5

Appendix 4

Rank table snapshot
(number in parentheses represents rank for designated statistics type)

Project	Rank	Score	Downloads	Logo Hits	Site Hits	CVS	SVN	GIT	Release Age	Last Login	Tracker Entries	ML Posts	Forum Posts
Project X	1	39,830,902	4,109 (282)	113,428 (38)	26,444 (312)	0 (0)	0 (10,386)	301 (3)	31 days	1 day	268 (4)	0 (0)	105 (12)