Generalized Procedure for Screening Free Software and Open Source Software Applications

Abstract

Free Software and Open Source Software projects have become a popular alternative tool in both scientific research and other fields. However, selecting the optimal application for use in a project can be a major task in itself, as the list of potential applications must first be identified and screened to determine promising candidates before an in-depth analysis of systems can be performed. To simplify this process we have initiated a project to generate a library of in-depth reviews of Free Software and Open Source Software applications.

Preliminary to beginning this project, a review of evaluation methods available in the literature was performed. As we found no one method that stood out, we synthesized a general procedure using a variety of available sources for screening a designated class of applications to determine which ones to evaluate in more depth. In this paper, we will examine a number of currently published processes to identify their strengths and weaknesses. By selecting from these processes we will synthesize a proposed screening procedure to triage available systems and identify those most promising of pursuit.

To illustrate the functionality of this technique, this screening procedure will be executed against a selected class of applications.

Introduction

There is much confusion regarding Free Software and Open Source Software and many people use these terms interchangeably, however, to some the connotations associated with the terms is highly significant. So perhaps we should start with an examination of the terms to clarify what we are attempting to screen. While there are many groups and organizations involved with Open Source software, two of the main ones are the Free Software Foundation (FSF) and the Open Source Initiative. (OSI).

When discussing **Free Software**, we are not explicitly discussing software for which no fee is charged, rather we are referring to free in terms of liberty. To quote the **Free Software Foundation** (**FSF**)[1]:

A program is free software if the program's users have the four essential freedoms:

- *The freedom to run the program as you wish, for any purpose (freedom 0).*
- *The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.*

- *The freedom to redistribute copies so you can help your neighbor (freedom 2).*
- *The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.*

This does not mean that a program is provided at no cost, or gratis, though some of these rights imply that it would be. In the FSF's analysis, any application that does not conform to these freedoms is unethical. While there is also 'free software' or 'freeware' that is given away at no charge, or gratis, but without the source code, this would not be considered Free Software under the FSF definition.

The **Open Source Initiative** (**OSI**), originally formed to promote Free Software, which they referred to as **Open Source Software** (**OSS**) to make it sound more business friendly. The OSI defines Open Source Software as any application that meets the following 10 criteria, which they based on the Debian Free Software Guidelines:[2]

- Free Redistribution
- Source Code Included
- Must allow derived works
- Must preserve the integrity of the authors source code
- License must not discriminate against persons or groups
- License must not discriminate against fields of endeavor
- Distribution of Licenses
- License must not be specific to a production
- License must not restrict other software
- License must be technology neutral.

Open Source Software adherents take what they consider the more pragmatic view of looking more at the license requirements and put significant effort into convincing commercial enterprises of the practical benefits of open source, meaning the free availability of application source code.

In an attempt to placate both groups when discussing the same software application, the term **Free/Open Source Software** (**F/OSS**) was developed. Since the term Free was still tending to confuse some people, the term libre, which connotes freedom, was added resulting in the term **Free/Libre Open Source Software** (**FLOSS**). If you perform a detailed analysis on the full specifications, you will find that all Free Software fits the Open Source Software definition, while not all Open Source Software fits the Free Software definition. However, any Open Source Software that is not also Free Software is the exception, rather than the rule. As a result, you will find these acronyms used almost interchangeably, but there are subtle differences in meaning, so stay alert. In the final analysis, the software license that accompanies the software is what you legally have to follow.

The reality is that since both groups trace their history back to the same origins, the practical differences between an application being Free Software or Open Source are generally negligible. Keep in mind that the above descriptions are to some degree generalizations, as

both organizations are involved in multiple activities. There are many additional groups interested in Open Source for a wide variety of reasons. However, this diversity is also a strong point, resulting in a vibrant and dynamic community. You should not allow the difference in terminology to be divisive. The fact that all of these terms can be traced back to the same origin should unite us.[3] In practice, many of the organization members will use the terms interchangeably, depending on the point that they are trying to get across.

With in excess of 300,000 FLOSS applications currently registered in SourceForge.net[4] and over 10 million repositories on GitHub[5], there are generally multiple options accessible for any class of application, be it a Laboratory Information Management System (LIMS), an office suite, a data base, or a document management system. Presumably you have gone through the assessment of the various challenges to using an Open Source application[6] and have decided to move ahead with selecting an appropriate application. The difficulty now becomes selecting which application to use. While there are multiple indexes of FOSS projects, these are normally just listings of the applications with a brief description provided by the developers with no indication of the vitality or independent evaluation of the project.

What is missing is a catalog of in-depth reviews of these applications, eliminating the need for each group to go through the process of developing a list of potential applications, screening all available applications, and performing in-depth reviews of the most promising candidates. While once they've made a tentative selection, the organization will need to perform their own testing to confirm that the selected application meets their specific needs, there is no reason for everyone to go through the tedious process of identifying projects and weeding out the untenable ones.

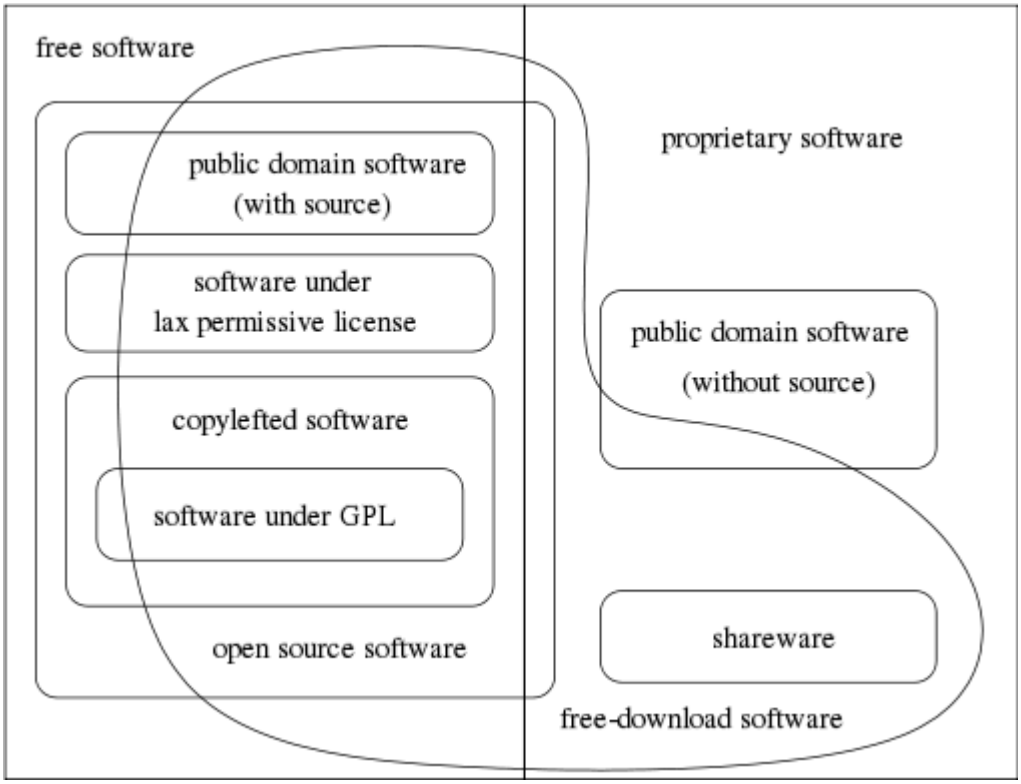

Illustration 1: This diagram, originally by Chao-Kuei and updated by several others since, explains the different categories of software. It's available as a Scalable Vector Graphic and as an XFig document, under the terms of any of the GNU GPL v2 or later, the GNU FDL v1.2 or later, or the Creative Commons Attribution-Share Alike v2.0 or later. [101]

The primary goal of this document is to describe a general procedure capable of being used to screen any selected class of software applications. The immediate concern is with screening FLOSS applications, though allowances can be made to the process to allow at least rough cross-comparison of both FOSS and commercial applications. To that end it we start with an examination of published survey procedures. We then combine a subset of standard software evaluation procedures with recommendations for evaluating FLOSS applications. Because it is designed to screen such a diverse range of applications, the procedure is by necessity very general. However, as we move through the steps of the procedure, we will describe how to tune the process for the class of software that you are interested in.

You can also ignore any arguments regarding selecting between FLOSS and commercial applications. In this context, commercial is referring to the marketing approach, not to the quality of the software. Many FLOSS applications have comparable, if not superior quality, to products that are traditionally marketed and licensed. Wheeler[7] discusses this issue in more detail, showing that by many definitions FLOSS ***is*** commercial software.

The final objective of this process is to document a procedure that can then be applied to any class of FOSS applications to determine which projects in the class are the most promising to pursue, allowing us to expend our limited resources most effectively. As the information available for evaluating FOSS projects is generally quite different from that available for commercially licensed applications, this evaluation procedure has been optimized to best take advantage of this additional information.

RESULTS

A search of the literature returns thousands of papers related to Open Source software, but most are of limited value in regards to the scope of this project. The need for a process to assist in selecting between Open Source projects is mentioned in a number of these papers and there appear to be over a score of different published procedures. Regrettably, none of these methodologies appear to have gained large scale support in the industry. Stol and Babar[8] have published a framework for comparing evaluation methods targeting Open Source software and include a comparison of 20 of them. They noted that web sites that simply consisted of a suggestion list for selecting an Open Source application were not included in this comparison. This selection difficulty is nothing new with FLOSS applications. In their 1994 paper, Fritz and Carter[9] review over a dozen existing selection methodologies, covering their strengths, weaknesses, the mathematics used, and other factors involved.

No.	Name	Year	Orig	Method
1	Capgemini Open Source Maturity Model	2003	I	Yes
2	Evaluation Framework for Open Source Software	2004	R	No
3	A Model for Comparative Assessment of Open Source Products	2004	R	Yes
4	Navica Open Source Maturity Model	2004	I	Yes
5	Woods and Guliani's OSMM	2005	I	No
6	Open Business Readiness Rating (OpenBRR)[10,11]	2005	R/I	Yes
7	Atos Origin Method for Qualification and Selection of Open Source Software (QSOS)	2006	I	Yes
8	Evaluation Criteria for Free/Open Source Software Products	2006	R	No
9	A Quality Model for OSS Selection	2007	R	No

No.	Name	Year	Orig	Method
10	Selection Process of Open Source Software	2007	R	Yes
11	Observatory for Innovation and Technological transfer on Open Source software (OITOS)	2007	R	Yes
12	Framework for OS Critical Systems Evaluation (FOCSE)	2007	R	No
13	Balanced Scorecards for OSS	2007	R	No
14	Open Business Quality Rating (OpenBQR)	2007	R	Yes
15	Evaluating OSS through Prototyping	2007	R	Yes
16	A Comprehensive Approach for Assessing Open Source Projects	2008	R	No
17	Software Quality Observatory for Open Source Software (SQO-OSS)	2008	R	Yes
18	An operational approach for selecting open source components in a software development project[12]	2008	R	No
19	QualiPSo trustworthiness model	2008	R	No
20	OpenSource Maturity Model (OMM)[13]	2009	R	No

Table 1: Comparison frameworks and methodologies for examination of FLOSS applications extracted from Stol and Babar.[8] The selection procedure is described in Stol's and Barbar's paper, however, 'Year' indicates the date of publication, 'Orig.' indicates whether the described process originated in industry (I) or research (R), while 'Method' indicates whether the paper describes a formal analysis method and procedure (Yes) or just a list of evaluation criteria (No).

Extensive comparisons between some of these methods have also been published, such as Deprez's and Alexandre's comparative assessment of the OpenBRR and QSOS techniques[14]. Wasserman and Pal[15] have also published a paper under the title of *Evaluating Open Source Software*, which appears to be more of an updated announcement and in-depth description of the Business Readiness Rating (BRR) framework. Jadhav and Sonar[16] have also examined the issue of both evaluating and selecting software packages. They include a helpful analysis of the strengths and weaknesses of the various techniques. Perhaps more importantly, they clearly point out that there is no common list of evaluation

criteria. While the majority of the articles they reviewed listed the criteria used, Jadhav and Sonar indicated that these criteria frequently did not include a detailed definition, which required each evaluator to use their own, sometimes conflicting, interpretation.

Since the publication of Stol's and Babar's paper, additional evaluation methods have been published. Of particular interest are a series of papers by Pani, et al.[17–20] describing their proposed **FAME** (**F**ilter, **A**nalyze, **M**easure and **E**valuate) methodology. In their *Transferring Fame[20]* paper, they emphasized that all of the evaluation frameworks previously described in the published literature were frequently not easy to apply to real environments, as they were developed using an analytic research approach which incorporated a multitude of factors.

Their stated design objective with FAME is to reduce the complexity of performing the application evaluation, particularly for small organizations. As specified ”The goals of FAME methodology are to aid the choice of high-quality F/OSS products, with high probability to be sustainable in the long term, and to be as simple and user friendly as possible.” They further state that “The main idea behind FAME is that the users should evaluate which solution amongst those available is more suitable to their needs by comparing technical and economical factors, and also taking into account the total cost of individual solutions and cash outflows. It is necessary to consider the investment in its totality and not in separate parts that are independent of one another.”[20]

This paper breaks the FAME methodology into four activities.

1. Identify the constraints and risks of the projects
2. Identify user requirements and rank.
3. Identify and rank all key objectives of the project.
4. Generate a priority framework to allow comparison of needs and features.

Their paper includes a formula for generating a score from the information collected. The evaluated system with the highest 'major score', *Pjtot*, indicates the system selected. While it is a common practice to define an analysis process which condenses all of the information gathered into a single score, I highly caution against blindly accepting such a score. FAME, as well as a number or the other assessment methodologies, is designed for iterative use. The logical purpose of this is to allow the addition of factors initially overlooked into your assessment, as well as to change the weighting of existing factors as you reevaluate their importance. However, this feature means that it is also very easy to unconsciously, or consciously, skew the results of the evaluation to select any system you wish. Condensing everything down into a single value also strips out much of the information that you have worked so hard to gather. Note that you can generate the same result score using significantly different input values. While of value, selecting a system based on just the highest score could potentially leave you with a totally unworkable system.

Pani, et al. also describe a FAMEtool[19] to assist in this data gathering and evaluation. However a general web search, as well as a review of their FAME papers revealed no indication of how to obtain this resource. While this paper includes additional comparisons

with other FLOSS analysis methodologies and there are some hints suggesting that the FAMEtool is being provided as a web service, I have found no URL specified for it. As of now, I have received no responses from the research team via either e-mail or Skype, regarding FAME, the FAMEtool, or feedback on its use.

During this same time frame Soto and Ciolkowski[21,22] also published papers describing the QualOSS Open Source Assessment Model and compared it to a number of the procedures in Stol's and Barbar's table. Their focus was primarily on three process perspectives: product quality, process maturity, and sustainability of the development community. Due to the lack of anything more than a rudimentary process perspective examination, they felt that the following OSS project assessment models were unsatisfactory: QSOS, CapGemni OSMM, Navica OSMM, and OpenBRR. They position QualOSS as an extension of the tralatitious CMMI and SPICE process maturity models. While there are multiple items in the second paper that are worth incorporating into an in-depth evaluation process, they do not seem suitable for what is intended as a quick survey.

Another paper, published by Haaland and Groven[23] also compared a number of Open Source quality models. To this paper's credit, the authors devoted a significant amount of space to discussing the different definitions of quality and how the target audience of a tool might affect which definition was used. Like Stohl and Babar, they listed a number of the quality assessment models to choose from, including OSMM, QSOS, OpenBRR, and others. For their comparison, they selected OpenBRR and QualOSS. They appear to have classified OpenBRR as a first generation tool with a “User view on quality” and QualOSS as a second generation tool with a “business point of view”. An additional variation is that OpenBRR is primarily a manual tool while QualOSS is primarily an automated tool. Their analysis in this article clearly demonstrates the steps involve in using these tools and in highlighting where they are objective and subjective. While they were unable to answer their original question as to whether the first or second generation tools did a better job of evaluation, to me they answered the following even more important, but unasked question. As they proceeded through their evaluation, it became apparent as to how much the questions defined in the methods could affect the results of the evaluations. Even though the authors might have considered the questions to be objective, I could readily see how some of these questions could be interpreted in alternate ways. My takeaway is an awareness of the potential danger of using rigid tools, as they can skew the accuracy of the evaluation results depending on exactly what you want the evaluated application to do and how you plan to use it. These models can be very useful guides, but they should not be used to replace a carefully considered evaluation, as there will always be factors influencing the selection decision which did not occur to anyone when the specifications were being written.

Hauge, et al.[24], have noted that despite the development of several normative methods of assessment, empirical studies have failed to show wide spread adoption of these methods. From their survey of a number of Norwegian software companies, they have noticed a tendency for selectors to skip the in-depth search for what they call the 'best fit' application and fall back on what they refer to as a 'first fit'. This is an iterative procedure with the knowledge gained from the failure of one set of component tests being incorporated into the

evaluation of the next one. Their recommendation is for researchers to stop attempting to develop either general evaluation schemas or normative selection methods which would be applicable to any software application and instead focus on identifying situationally sensitive factors which can be used as evaluation criteria. This is a very rational approach as all situations, even if evaluating the same set of applications, are going to be different, as each user's needs are different.

Ayalal, et al.[25], have performed a study to try to more accurately determine why more people don't take advantage of the various published selection methodologies. While they looked at a number of factors and identified several possible problems, one of the biggest factors was the difficulty in obtaining the needed information for the evaluation. Based on the projects they studied, many did not provide a number of the basic pieces of information required for the evaluation, or perhaps worse, required extensive examination of the project web site and documentation to retrieve the required information. From her paper, it sounded as if this issue was more of a communication breakdown than an attempt to hide any of the information, not that this had any impact on the inaccessibility of the information.

In addition to the low engagement rates for the various published evaluation methods, another concern is the viability of the sponsoring organizations. One of the assessment papers indicated that the published methods with the smallest footprint, or the easiest to use, appeared to be FAME and the OpenBRR. I have already mentioned my difficulty obtaining additional information regarding FAME and OpenBRR appears to be even more problematic. BRR was first registered on SourceForge in September of 2005[26] and an extensive Request For Comments from the founding members of the BRR consortium (SpikeSource, the Center for Open Source Investigation at Carnegie Mellon West, and Intel Corporation) was released[10]. In 2006, in contrast to typical Open Source development groups, the OpenBRR group announced the formation of an OpenBRR Corporate Community group. Peter Galli's story[27] indicates that 'the current plan is that membership will not be open to all.' He quotes Murugan Pal saying “membership will be on an invitation-only basis to ensure that only trusted participants are coming into the system”. However, for some reason, at least some in the group “expressed concern and unhappiness about the idea of the information discussed not being shared with the broader open-source community”.

While the original Business Readiness Rating web site still exists[28], it is currently little more than a static web page. It appears that some of the original information posted on the site is still there, you just have to know what its URL is to access it, as the original links on the web site have been removed. Otherwise, you may have to turn to the Internet Archive to retrieve some of their documentation. The lack of any visible activity regarding OpenBRR prompted a blog post from one graduate student in 2012 asking “What happened to OpenBRR (Business Readiness Rating for Open Source)?”[29]

It appears that at some point, any development activity regarding OpenBRR was morphed into OSSpal[30]. However, background information on this project is sparse as well. While the site briefly mentions that OSSpal incorporates a number of lessons learned from BRR, there is very little additional information regarding the group or the methods procedures.

Their 'All Projects' tab provides a list of over 30 Open Source projects, but the majority simply show 'No votes yet' under the various headings. In fact, as of now, the only projects showing any input at all are for Ubuntu and Mozilla Firefox.

At this point, we'll take a step back from the evaluation methodologies papers and examine some of the more general recommendations regarding evaluating and selecting FLOSS applications. The consistency of their recommendations may provide a more useful guide for an initial survey of FLOSS applications.

In *TechRepublic*, De Silva recommends 10 questions to ask when selecting a FLOSS application[31]. While he provides a brief discourse on each question in his paper to ensure you understand the point of his question, I've collected the ten questions from his article into the following list. Once we see what overlap, if any, are amongst our general recommendations, we'll address some of the consolidated questions in more detail.

1. Are the open source license terms compatible with my business requirements?
2. What is the strength of the community?
3. How well is the product adopted by users?
4. Can I get a warranty or commercial support if I need it?
5. What quality assurance processes exist?
6. How good is the documentation?
7. How easily can the system be customized to my exact requirements?
8. How is this project governed and how easily can I influence the road map?
9. Will the product scale to my enterprise's requirements?
10. Are there regular security patches?

Similarly, in *InfoWorld* Phipps[32] lists 7 questions you should have answered before even starting to select a software package. His list of questions, pulled directly from his article are:

1. Am I granted copyright permission?
2. Am I free to use my chosen business model?
3. Am I unlikely to suffer patent attack?
4. Am I free to compete with other community members?
5. Am I free to contribute my improvements?
6. Am I treated as a development peer?
7. Am I inclusive of all people and skills?

This list of questions shows a moderately different point of view, as it is not just about someone selecting an Open Source system, but looking to be involved in its direct development. Padin[33], of 8th Light, Inc., takes the viewpoint of a developer who might incorporate Open Source software into their projects. The list of criteria pulled directly from his blog include:

1. Does it do what I need it to do?

2. How much more do I need it to do?
3. Documentation
4. Easy to review source code
5. Popularity
6. Tests and specs
7. Licensing
8. Community

Metcalfe[34] of OSS Watch lists his top tips as:

1. Reputation
2. Ongoing effort
3. Standards and interoperability
4. Support (Community)
5. Support (Commercial)
6. Version
7. Version 1.0
8. Documentation
9. Skill setting
10. Project Development Development Model
11. License

In his LIMSexpert blog, Joel Limardo[35] of ForwardPhase Technologies, LLC lists the following as components to check when evaluating an Open Source application:

- Check licensing
- Check code quality
- Test setup time
- Verify extensibility
- Check for separation of concerns
- Check for last updated date
- Check for dependence on outdated toolkits/frameworks

Perhaps the most referenced of the general articles on selecting FLOSS applications is David Wheeler's *How to Evaluate Open Source Software / Free Software (OSS/FS) Programs[36]*. The detailed functionality to consider will vary with the types of applications being compared, but there are a number of general features that are relevant to almost any type of application. While we will cover them in more detail later, Wheeler categorizes the features to consider as the following.

- System functionality
- System cost – direct and in-direct
- Popularity of application, i.e. its market share for that type of application
- Varieties of product support available
- Maintenance of application, i.e, is development still taking place

- Reliability of application
- Performance of application
- Scalability of application
- Usability of application
- Security of application
- Adaptability/customizability of application
- Interoperability of application
- Licensing and other legal issues

While a hurried glance might suggest a lot of diversity in the features these various resources suggest, a closer look at the meaning of what they are saying show a repetitive series of concerns. The primary significant differences between the functionality lists suggested is actually due more to how wide a breadth of the analysis process the authors are considering, as well as the underlying features that they are concerned with.

With a few additions, the high-level screening template described in the rest of this communication is based on Wheeler's previously mentioned document describing his recommended process for evaluating open source software and free software programs. Structuring the items thus will make it easier to locate the corresponding sections in his document, which includes many useful specific recommendations, as well as a great deal of background information to help you understand the *why* of the topic. I highly recommend reading it and following up on some of the links he provides. I will also include evaluation suggestions from several of the previously mentioned procedures where appropriate.

Wheeler defines four basic steps to this evaluation process, as listed below.

- **I**dentify candidate applications
- **R**ead existing product reviews
- **C**ompare attributes of these applications to your needs
- **A**nalyze the applications best matching your needs in more depth.

Wheeler categorizes this process with the acronym **IRCA**. In this paper we will be focusing on the **IRC** components of this process. To confirm the efficacy of this protocol we will later apply it to several classes of Open Source applications and examine the output of the protocol.

Identify Needs

Realistically, before you can perform a survey of applications to determine which ones best match your needs, you must determine what your needs actually are. The product of determining these needs is frequently referred to as the User Requirements Specification (URS)[37,38]. This document can be generated in several ways, including having all of the potential users submit a list of the functions and capabilities that they feel is important. While the requirements document can be created by a single person, it is generally best to make it a group effort with multiple reviews of the draft document and including all of the users who will be working with the application. The reason for this is to ensure that an important

requirement is not missed, When a requirement is missed, it is frequently due to the requirement being so basic that it never occurs to anyone that it specifically needed to be included in the requirements document. Admittedly, a detailed URS is not required at the survey level, but it is worth having if only to identify, by their implications, other features that might be significant.

Needs will, of course, vary with the type of application you are looking for and what you are planning to do with it. Keep in mind that the URS is a living document, subject to change through this whole process. Developing a URS is generally an iterative process, since as you explore systems, you may well see features that you hadn't considered that you find desirable. This process will also be impacted by whether the application to be selected will be used in a regulated environment. If it is, there will be existing documents that describe the minimum functionality that must be present in the system. Even if it is not to be used in a regulated environment, documents exist for many types of systems that describe the recommended functional requirements that would be expected for that type of system.

For a clarifying example, if you were attempting to select a Laboratory Information Management System (LIMS), you can download checklists and standards of typical system requirements from a variety of sources[39–42]. These will provide you with examples of the questions to ask, but you will have to determine which ones are important to, or required for, your particular effort.

Depending on the use to which this application is to be applied, you may be subject to other specific regulatory requirements as well. Which regulations may vary, since the same types of analysis performed for different industries fall under different regulatory organizations. This aspect is further complicated by the fact that you may be affected by more than one countries' regulations if your analysis are applicable to products being shipped to other countries. While some specific regulations may have changed since its publication, an excellent resource to orient you to the diverse factors that must be considered is Siri Segalstad's book, *International IT Regulations and Compliance[43]*. My understanding is that an updated version of this book is currently in preparation. Keep in mind that while regulatory requirements that you must meet will vary, these regulations by and large also describe best practices, or at least the minimal allowed practices. These requirements are not put in place arbitrarily (generally) or to make things difficult for you, but to ensure the quality of the data produced. As such, any deviations should be carefully considered, whether working in a regulated environment or not. Proper due diligence would be to determine which regulations and standards would apply to your operation.

For a LIMS, an example of following best practices is to ensure that the application has a full and detailed audit trail. An audit trail allows you to follow the processing of items through your system, determining who did what and when. In any field where it might become important to identify the actions taken during a processing step, an audit trail should be mandatory. While your organization's operations may not fall under the FDA's 21 CFR part 11 regulations, which address data access and security, including audit trails, it is still extremely prudent that the application you select complies with them. If it does not, then almost anyone could walk up to

your system and modify data, either deliberately or accidentally, and you would have no idea of who made the changes or what changes they made. For that matter, you might not even be able to tell a change was made at all, which likely will raise concerns both inside and outside of your organization. This would obviously cause major problems if they became a hinge issue for any type of liability law suit.

For this screening procedure, you do not have to have a fully detailed URS, but it is expedient to have a list of your make-or-break issues. This list will be used later for comparing systems and determining which ones justify a more in-depth evaluation.

Identify Candidates

To evaluate potential applications against your functional criteria, you must initially generate a list of potential systems. While this might sound easy, generating a *comprehensive* list frequently proves to be a challenge. When initiating the process, you must first determine the type of system that you are looking for, be it a LIMS, a hospital management system, a data base, et al. At this point, you should be fairly open in building your list of candidates. By that, I mean that you should be careful not to select applications based solely on the utilization label applied to them. The same piece of software can frequently be applied to solve multiple problems, so you should cast a wide net and not automatically reject a system because the label you were looking for hadn't been applied to it. While the label may give you a convenient place to start searching, it is much more important to look for the functionality that you need, not what the system is called. In any case, many times the applied labels are vague and mean very different things to different people.

There are a variety of ways to generate your candidate list. A good place to start is simply talking with colleagues in your field. Have they heard of or use a FLOSS application of the appropriate type that they like? Another way is to just flip through journals and trade magazines that cover your field. Any sufficiently promising applications are likely to be mentioned there. Many of the trade magazines will have a special annual issue that covers equipment and software applicable to their field. It is difficult to generate a list of all potential resources, as many of these trade publications are little known outside of their field. Also keep in mind that with the continued evolution of the World Wide Web, many of these trade publications also have associated web sites that you can scan or search. The table below includes just a minor fraction of these sites that are available.[1]

Field	Resource Name	URL
Astronomy	Tech Support Alert	http://www.techsupportalert.com/best-free-astronomy-software.htm

1 We would welcome the suggestion of any additional resource sites that you are aware of. Please e-mail the fields covered, the resource name, and either its general URL or the URL of the specific resource section to the corresponding editor.

Field	Resource Name	URL
Business Intelligence Reporting	Technology Innovation Management Review	http://timreview.ca/article/288
Community Radio	Prometheus Radio Project	http://prometheusradio.org/Free_Open_Source_Tools_C_R
Comprehensive Coverage	Black Duck KnowledgeBase	https://www.blackducksoftware.com/products/knowledgebase
Comprehensive Coverage	The Directory of Open Access Repositories - **OpenDOAR**	http://www.opendoar.org/
Data Storage	InfoStor	http://www.infostor.com/nas/58-top-open-source-storage-project-1.html
Digital Audio Editors	25 Free Digital Audio Editors You Should Know	http://www.hongkiat.com/blog/25-free-digital-audio-editors/
Digital Video Editors	Best free video editing software: our 20 top programs of 2015	http://www.techradar.com/us/news/software/applications/best-free-video-editing-software-9-top-programs-you-should-download-1136264
Divers Range of non-GNU Applications	Savannah Non-GNU	savannah.nongnu.org
Diverse Range of Android Applications	Wikipedia – Android Applications	https://en.wikipedia.org/wiki/List_of_free_and_open-source_Android_applications
Diverse Range of Applications	Microsoft CodePlex	https://www.codeplex.com/
Diverse Range of Applications	Open Hub, Black Duck Software, Inc	https://www.openhub.net/
Diverse Range of Applications – Primarily targeting Samsung products.	Samsung Open Source Release Center	http://opensource.samsung.com/reception.do

Field	Resource Name	URL
Diverse Range of Applications focusing on GNU	Savannah	http://savannah.gnu.org/
Diverse Range of Applications, but site is deprecated.	Google Code	https://code.google.com/p/support/
Diverse Range of Applications, currently 37,168.	Launchpad, The Canonical Group	http://launchpad.net/
Diverse Range of Commercial Open Source Applications and Services	Wikipedia – Commercial Open Source Applications	https://en.wikipedia.org/wiki/List_of_commercial_open-source_applications_and_services
Diverse Range of Top Open Source Applications	Projects and Applications	http://opensource.com/resources/projects-and-applications
Drug Discovery	Drug Discovery Today	http://linkinghub.elsevier.com/retrieve/pii/S1359644605036925
Electronic Engineering	Education Engineering (EDUCON), 2010 IEEE	http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5492430
Embroidery	Machine Embroidery Portal	http://www.k2g2.org/portal:machine_embroidery
Enterprise Resource Planning (ERP)	A Comparison of Open Source ERP Systems	http://www.big.tuwien.ac.at/system/theses/20/papers.pdf?1298476232
Enterprise Resource Planning (ERP)	Open Source ERP Site	http://www.open-source-erp-site.com/list-of-open-source-erps.html
Extensive Coverage – Primary focus is Infrastructure	Open Source Guide by Smile	http://www.open-source-guide.com/en
Geographic Information Systems (GIS)	Open Source GIS	http://opensourcegis.org/

Field	Resource Name	URL
Geophysics Software	Geopsy project	http://www.geopsy.org/
Geophysics Software	Wikipedia, Geophysics Software	https://en.wikipedia.org/wiki/Comparison_of_free_geophysics_software
GIS for Libraries	Library Hi Tech	http://www.emeraldinsight.com/doi/abs/10.1108/07378831011026742
Help Desk Software	CIO	http://www.cio.com.au/article/320110/5_open_source_help_desk_apps_watch/
Highly Diverse	Code NASA	https://code.nasa.gov/#/
Highly Diverse	NASA Open Source Software	http://ti.arc.nasa.gov/opensource/
Laboratory Informatics	Lab Manager	http://www.labmanager.com/vendor-list/2011/02/lims-manufacturer-list
Laboratory Informatics	LIMSwiki	http://www.limswiki.org/index.php?title=LIMS_vendor
Laboratory Informatics	*Scientific Computing*	http://www.scientificcomputing.com/topics/lims-guide
Laboratory Informatics	*Scientific Computing World*	http://www.scientific-computing.com/lig2015/
Learning Management Systems (LMS)	Barry Sampson (Blog)	http://barrysampson.com/2009/04/08/open-source-lms-10-alternatives-to-moodle/
Library Software	FOSS4LIB	https://foss4lib.org/packages/
Limited List of Open Source iOS Applications	Wikipedia, iOS Applications	https://en.wikipedia.org/wiki/List_of_free_and_open-source_iOS_applications
Medical Office Records	ZDNet	http://www.zdnet.com/article/free-and-open-source-healthcare-software-for-your-practice/
Medicine	'Open' Health IT Solutions for the Public & Private Sectors	https://drive.google.com/file/d/0B_TuX9zE68eWcVM1dkdDQTVCdnM/edit
Medicine	50 Successful Open Source Projects That Are Changing Medicine	http://nursingassistantguides.com/2009/50-successful-open-source-projects-that-are-changing-medicine/

Field	Resource Name	URL
Medicine	Datamation	http://www.datamation.com/open-source/50-open-source-replacements-for-health-care-software-1.html
Medicine	Medfloss.org	http://www.medfloss.org/ http://www.apfelkraut.org/download/Medfloss_linuxtag_hschmuhl_2012.pdf
Medicine	Nursing Assistant Guides	http://nursingassistantguides.com/2009/50-successful-open-source-projects-that-are-changing-medicine/
Medicine/Laboratory Informatics	Healthcare Freeware	http://www.healthcarefreeware.com/lim.htm
MIDI Controllers	Open Source MIDI Controllers	http://punkmanufacturing.com/wiki/open-source-midi-controllers
Music	20 great free and open source music making programs	http://www.musicradar.com/tuition/tech/20-great-free-and-open-source-music-making-programs-582934
Patterns- Crochet, Counted Cross Stitch, Origami	Tools for Computer Generated Patterns	https://xxxclairewilliamsxxx.wordpress.com/tools-to-create-and-explore-digital-patterns/
Personal Information Managers (PIM)	Wikipedia – Personal Information Managers (PIMS)	https://en.wikipedia.org/wiki/List_of_personal_information_managers#Open_source_applications
Photography	Open Source Photography	http://opensourcephotography.org/software-list
Project Management Software	Project Management Software	http://www.projectmanagementsoftware.com/
Question Survey Software	Capterra	http://www.capterra.com/survey-software/
Repository Software packages	Free and open-source repository software	http://oad.simmons.edu/oadwiki/Free_and_open-source_repository_software
Seismic Survey Data Sets	Open Seismic Repository	https://opendtect.org/osr/

Field	Resource Name	URL
Seismic Survey Data Sets	SPOT Dept. of Geological Engineering & Sciences (Seismic Petrophysics: Observation & Theory)	http://www.geo.mtu.edu/spot/SeismicData/
Sewing	All Free Sewing	http://www.allfreesewing.com/
Sewing	Open Source Sewing Patterns	http://www.silverseams.com/opensource/
Sewing/Patterns/ Knitting	Ethical Fashion Forum	http://source.ethicalfashionforum.com/article/9-open-source-low-cost-digital-fashion-business-tools
Test Measurement Tools	Software Testing Help	http://www.softwaretestinghelp.com/15-best-test-management-tools-for-software-testers/

Table 2: Examples of focused FLOSS resource sites available on the web.

I also recommend checking some of the general open source project lists, such as the ones generated by Cynthia Harvey at *Datamation[44]*, which has been covering the computer and data-processing industry since 1957. In particular, you might find their article *Open Source Software List: 2015 Ultimate List[45]* useful. It itemizes over 1,200 open source applications, including some in categories that I didn't even know existed. It would also be prudent to search the major open source repositories, such as SourceForge[46] and GitHub[47]. Wikipedia includes a comparison of source code hosting facilities that would be worth reviewing as well.[48] Keep in mind that you will need to be flexible with your search terms, as the developers might be looking at the application differently than you are. While they were created for a different purpose, an examination of the books in *The Architecture of Open Source Applications* might prove useful as well.[49] Other sites where you might find interesting information regarding new Open Source applications, are the various Open-Source award sites, such as the *InfoWorld* **B**est of **O**pen **S**ource **S**oftware **A**wards, colloquially known as the Bossies.[50] When searching the Web, don't rely on just *Google* or *Bing*. Don't forget to checkout all of the journal web sites, such as SpringerLink, Wiley, ScienceDirect, PubMed, and others, as they contain a surprising amount of information on FLOSS. If you don't wish to search each of them individually, there are other search engines out there which can give you an alternate view of the research resources available. To name just two, be sure to try both *Google Scholar[51]* and *Microsoft Academic Search[52]*. These tools can also be used to search for masters theses and doctoral dissertations, which likewise contain a significant amount of information regarding open source.

While working on creating your candidate list, be sure to pull any application reviews[53] that you come across. If done well, these reviews can save you a significant amount of time in screening a potential system. However, unless you're familiar with the credentials of the author, be cautious of relying on them for all of your information. While not common, people have been known to post fake reviews online, sometimes when it is not even April 1st!
Another great resource, both for identifying projects and obtaining information about them, is **Open HUB**[54], an Open Source web site maintained by Black Duck Software, Inc. Open Hub allows you to search by Projects, People, Organizations, Forums, and Code. For example, if I searched for *Bika LIMS*, it would currently return the results for *Bika Open Source LIMS 3* along with some basic information regarding the system. If I were to click on the projects name, a much more detailed page regarding this project is displayed. Moving your mouse cursor over the graphs displays the corresponding information for that date.

Once a list of candidate applications has been generated, the list entries must be compared. Some of this comparison can be performed objectively, but it also requires subjective analysis of some components. As Stol and Babar have shown, there is no single recognized procedure for either the survey or detailed comparison of FLOSS applications

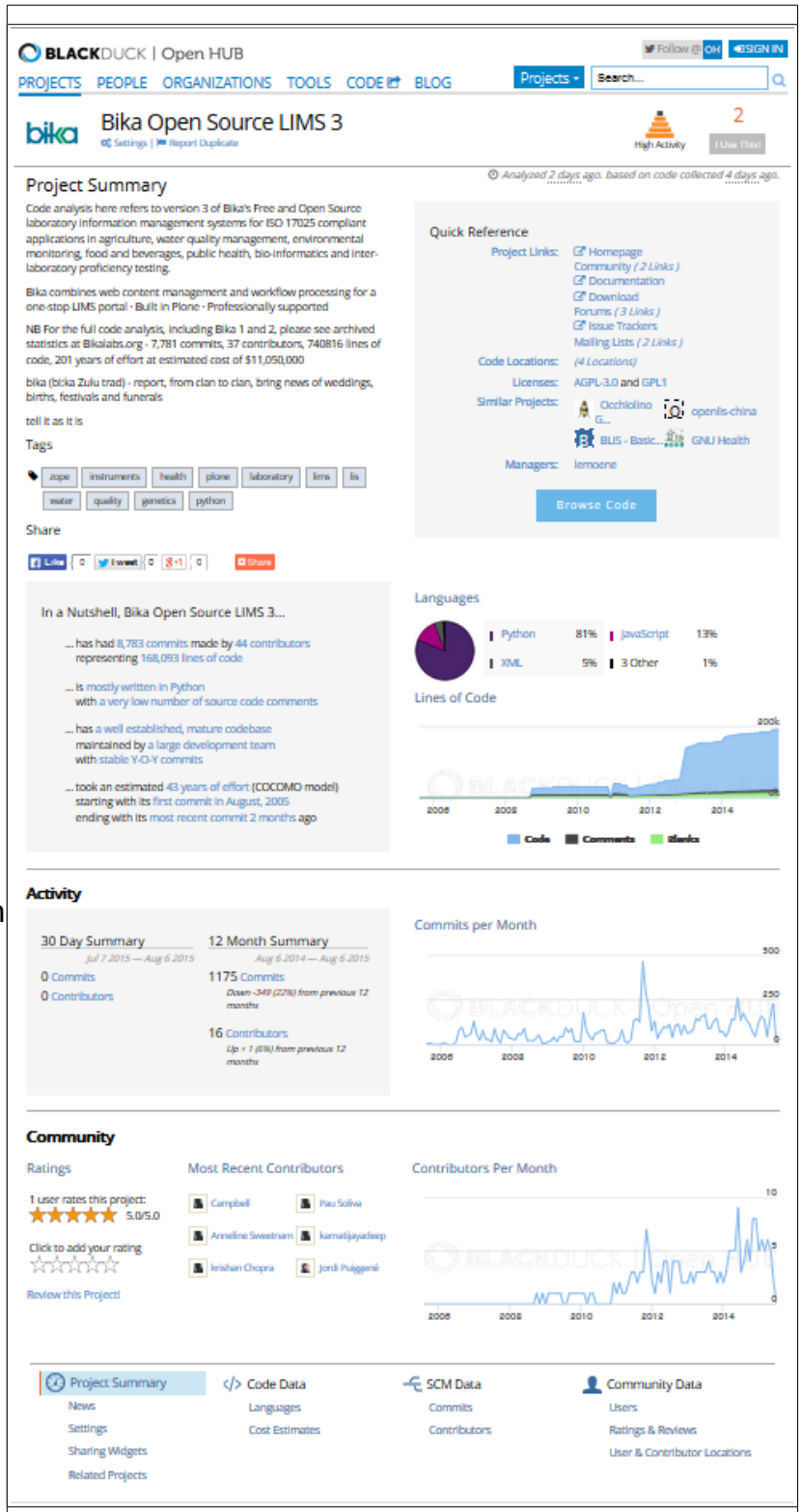

Illustration 2: Result page displayed on Black Duck Software's Open Hub site for the Bika Open Source LIMS 3 project.

that has shown a marked popularity above the others.

The importance of any one specific aspect of the evaluation will vary with the needs of the organization. General system functionality will be an important consideration, but specific aspects of the contained functionality will have different values to different groups. For instance, interoperability may be very important to some groups, while others may be using this application as their only data system and they have no interest in exchanging data files with others, so interoperability is not a concern to them. While you can develop a weighting system for different aspects of the system, this can easily skew selection resulting in a system that has a very good rating, yet is unable to perform the required function. Keep in mind that this is a high level survey, we are asking broad critical questions, not attempting to compare detailed minutia. Also keep in mind that a particular requirement might potentially fall under multiple headings. For example, compliance with 21 CFR part 11 regulations might be included under functionality or security.

Screening Protocol

While in-depth analysis of the screened systems will require a more detailed examination and comparison, for the purpose of this initial survey a much simpler assessment protocol will serve. While there is no single 'correct' evaluation protocol, something in the nature of the three leaf scoring criteria described for QSOS[10] should be suitable. Keep in mind that for this quick assessment we are using broad criteria, so both the criteria and the scoring will both be more ambiguous than that required for an in-depth assessment. Do not be afraid to split any of these criteria up into multiple finer classes of criteria if the survey requires it. This need would be most likely to occur under System Functionality, as that is where most people's requirements greatly diverge.

In this process, we will assign one of three numeric values to each of the listed criteria. A score of zero indicates that the system does not meet the specified criteria. A score of one indicates that the system marginally meets the specified criteria. You can look on this as the feature is present and workable, but not to the degree you'd like it. Finally, a score of two indicates that the system fully meets or exceeds the specified requirement. In the sections below I will list some possible criteria for this table. However, you can adjust these descriptions or add a weighting factor, as many other protocols do, to adjust for the criticality of a given requirement.

Realistically, when it comes down to some of your potential evaluation criteria, the actuality is that for some of them, you can compensate for the missing factor in some other way. For other criteria, their presence or absence can be a drop-dead issue. That is, if the particular criteria or feature isn't present, then it doesn't matter how well any of the other criteria are ranked, that particular system is out of consideration. Deciding which, if any, criteria are drop-dead items should ideally be determined before you start your survey. This will not only be more efficient, in that it will allow you to cut off data collection at any failure point, but it will also help dampen the psychological temptation to fudge your criteria, retroactively deciding

that a given criteria was not that important after all.

At this stage we are just wanting to reduce the number of systems for in-depth evaluation from potentially dozens, to perhaps three or four. As such, we will be refining our review criteria later, so if something isn't really a drop-dead criteria, don't mark it so. It's amazing the variety of feature tradeoffs people tend to make further down the line.

System Functionality

While functionality is a key aspect of selecting a system, its assessment must be used with care. Depending on how a system is designed, a key function that you are looking for might not have been implemented, but in one system it can easily be added, while in another it would take a complete redesign of the application. Also consider the possibility of whether this function must be intrinsic to the application or if you can pair the application being evaluated with another application to cover the gap.

In most cases, you can obtain much of the functionality information from the projects web site, or occasionally, web sites. Some projects have multiple web sites, usually with one focused on project development and another targeting general users or application support. There are two different types of functionality to be tested. The first might be termed *general functionality*, that would apply to almost any system. Examples of this could include the following:

- User Authentication
- Audit Trail – (I'm big on detailed audit trails, as they can make the difference between between being in regulatory compliance and having a plant shut down.) Even if you aren't required to have them, they are generally a good idea, as the records they maintain may be the only way for you to identify and correct a problem.
- Sufficient system status display that the user can understand the state of the system.
- Ability to store data in a secure fashion.

We might term the second as *application functionality*. This is functionality specifically required to be able to perform your job. As the subject matter expert, YOU will be the one to create this list. Items might be as diverse as the following

- For a LIMS you might be looking for things like
  - Can it print bar coded labels.
  - Can it track the location of samples through the laboratory, as well as maintain the chain-of-custody for the sample.
  - Can it track the certification of analysts for different types of equipment, as well as monitor the preventive maintenance on the instruments.
  - Can it generate and track aliquots of the original sample
- Geographic Information Systems (GIS)[55]
  - What range of map projections is supported.
  - Does the system allow you to create custom parameters.
  - Does it allow import of alternate system formats.

- Can it directly interface with Geographic Positioning Devices (GPS)?

- Library Management Software System (LMSS) [or Integrated Library System (ILS), if you prefer, or even LIMS for Library Information Management System]. (Have you ever noticed how scientists love to reuse acronyms, even within the same field?)
  - Can you identify the location of any item in the collection at a given time.
  - Can you identify any items that were sent out for repair and when.
  - Can it identify between multiple copies of an item and between same named items on different types of media.
  - Can it handle input from RFID tags?
  - Can it handle and differentiate different clients residing at the same address?
  - If needed, can it correlate the clients age with the particular item they are requesting, in case you have to deal with any type of age appropriate restrictions.
    - If so, can it be overridden where necessary (maintaining the appropriate records in the audit trail as to why the rule was overridden)?
- Archival Record Manager – This classification can cover a lot of ground, due to all of the different ways that 'archival' and 'record' are interpreted.
  - In some operations, a record can be any information about a sample, including the sample itself. By regulation, some types of information must be maintained essentially for ever. In others, you might have to keep information for 5 years, while you have to maintain data for another type for 50 years.
    - Can the application handle tracking records for different amounts of time?
    - Does the system automatically delete these records at the end of the retention period or does it ask for confirmation from a person.
    - Can overrides on a particular sample be applied, so that records are not allowed to be deleted, either manually of because they are past their holding date, such as those that might be related to any litigation. Again, maintaining all information about the override and who applied the override in the audit trail.
  - In other operations, an archival record manager may actually refer to the management of archival records, be these business plans, architectural plans, memos, art work, etc.
    - Does the system keep track of the type of each record.
    - Does the system support appropriate meta data on different types of records?
      - Does it just record the items location and who is responsible for it, such as a work of art?
      - If a document, does it just maintain an electronic representation of a document, such as a PDF file, or does it record the location of the original physical document, or can it do both?
      - Can it manage both permanently archived items, such as a work of art or a historically significant document, and more transitory items, where your record retention rules say to save it for 5, 10, 15 years, etc., and then destroy it.
        - In the later case, does the system require human approval before destroying any electronic documents or flagging a physical item for disposal? Does it require a single human sign-off or must a chain of people sign-off on it, just to confirm that it is something to be discarded

by business rules and not an attempt to hide anything?

- Medical Records[56,57] – This is a challenging quagmire, with frequently changing regulations and requirements. Depending on how you want to break it down, this heading can be segmented into two classes: Electronic Medical Records (EMR) which can constitute an electronic version of the tracking of the patients health and Electronic Health Records (EHR) which contains extensive information on the patient, including test results, diagnostic information, and other observations by the medical staff. For those who want to get picky, you can also subdivide the heading into Imaging Systems, such as X-rays and CAT scans and other specialized systems.
  - Can all records be accessed quickly and completely under emergency situations.
  - What functionality is in place to minimize the risk of a Health Insurance Portability and Accountability Act (HIPAA) violation[58]
  - What functionality exists for automated data transfer from instruments or laboratory data systems to minimize transcription errors.
  - How are these records integrated with any billing or other medical practice system?
    - If integrated with the EMR and EHR record systems, does this application apply granular control over who can access these records and what information they are able to see.
- Enterprise Resource Planning System (ERP)
  - Davis[59] has indicated that a generally accepted definition of an ERP is: “ERP systems are complex, modular software integrated across the company with capabilities for most functions in the organization.“ I believe this translates as 'good luck', considering the complexity of the systems an ERP is designed to model and all of the functional requirements that go into that. It is perhaps for that reason that successful ERP implementations generally take several years.
  - ERP systems generally must be integrated with other informatics systems within the organization. What types of interfaces does this system support?
    - Is their definition of plug-and-play that you just have to configure the addresses and fields to exchange?
    - Is their definition of interface that the system can read in information in a specified format and export the same, leaving you to write the middle ware program to translate the formats between the two systems?

From the above, it is easy to see why researchers have encountered difficulty in developing a fixed method that can be used to evaluate anything. At this point it is quite acceptable to group similar functions together, as this is a high level survey to identify which systems will definitely NOT be suitable, so we can focus our researches on those that might be. Many researchers, such as Sarrab and Rehman[60], summarize system functionality as “achieving the user's requirements, correct output as user's expectations and verify that the software functions appropriately as needed.”

Suggested ratings are:

- Zero – Application does not support required functionality.
- One – Application supports the majority of functionality to at least an useable extent.
- Two – Application meets or exceeds all functional requirements.

Community

Many of the researchers that I've encountered have indicated that Community is the most critical factor of a FLOSS project. There are a number of reasons for this.

- The health and sustainability of a FLOSS project is indicated by a large and diverse community that is both active and responsive.[31]
- Generally, the core programmers in a FLOSS project are few, it is the size of the project community that determines how well reviewed the application is, ensuring quality control of the projects code.
- The size of the project community correlates with the lifetime of the project.

Suggested ratings are:

- Zero – No community exists. No development activity is observable. Project is dead.
- One - Community is small and perhaps insular. May consist of just one or two programmers with perhaps a small number of satellite users.
- Two – Community is large and dynamic, with many contributors and active communication between the core developers and the rest of the community. Community is responsive to outside inquiries.

System Cost

Since the main goal of this survey procedure is to evaluate FLOSS products, the base system cost for the software will normally be low, frequently $0.00. However, this is not the only cost you need to consider. Many of these costs could potentially be placed under multiple headings, depending on how your organization is structured. No matter how it itemizes them, there will be additional costs. Typical items to consider include the following:

- Cost of supporting software – e.g. does it require a commercial data base such as Oracle or some other specialized commercial software component?
- Cost of additional hardware – e.g. does the system require the purchase of additional servers or storage systems? Custom hardware interfaces?
- Cost of training – e.g. How difficult or intuitive is the system to operate? This will impact the cost of training that users must receive. Keep in mind this cost will exist whether you are dealing with an Open Source or proprietary system. Are costs for system manuals and other required training material included? Some proprietary systems don't, or might perhaps send a single hardcopy of the manuals. Who will perform the training? Whether you hire someone from outside or have some of your own people do it, there will be a cost, as you would be pulling people away from from their regular jobs.
- Cost of support – e.g. is support through a commercial organization or the Open Source development group? If the former, what are their contract costs? While harder to evaluate, what is the turnaround time from when you request the support?

Immediate? Days? Sometime? The amount of time you have to wait for a problem to be fixed, is definitely a 'cost', whether it means your system is dead in the water or just not as efficient and productive as it could be.

The primary issue here is to be realistic in your evaluation. It's hard to believe that anyone would assume that there were no associated costs with using FLOSS, or proprietary software for that matter, but apparently there are. Foote[61] does a good job of exploring and disproving this belief, showing all of the items that go into figuring the total cost of ownership (TCO), which should be representative of FLOSS applications. I wouldn't call them hidden costs, at least not with the FLOSS systems, just costs that are overlooked, as are so many other things when people focus on a single central item. To quote Robert Heinlein[62], "**TANSTAAFL**!"

The important thing here is to pay attention to all of the interactions taking place. For example, if you wished to interface a piece of equipment to your FLOSS application, remember to factor in the cost of the interface. Despite what some advertisers think, data bits don't just disappear from one place and magically appear in another. It is very easy to lose track of where the costs are. Keep in mind that many items, such as training, you will have a cost either way you go. If a proprietary vendor says they will provide free training, you can be assured that the cost for it is included in the contract. But if you take that route, be very careful to read the contract thoroughly, as not all vendors include any training at all. It would be very easy to end up having to pay for 'optional' training.

Suggested ratings are:

- Zero – Installation and support costs are excessive and greatly exceed any available budget.
- One – System may require purchase of additional hardware or customization. These costs, along with training costs, are within potential budget.
- Two – Installation and support costs are relatively minor, with no additional hardware required. System design is relatively intuitive with in-depth documentation and active support from the community.

Popularity

This heading can be somewhat confusing in terms of how it is interpreted, even though most of the recommendations we looked at include it. Popularity is sometimes considered to be similar to market share. That is, of the number of people using a specific application in a given class of Open Source applications, what percentage do they represent out of all people running applications in the class? If the majority of people are using a single system, this *might* indicate that it is the better system, or it might just indicate that other systems are newer and, even if potentially better, people haven't migrated over to them yet. An alternate approach to examining it is to ask how many times the application has been downloaded. In general, the larger the market share or the number of downloads, the more likely that a given product is to be usable. This is not an absolute, as people may have downloaded the

application for testing and then rejected it or downloaded it simply to game the system, but it is a place to start. The point of this question usually isn't really to determine how popular a particular application is, but rather to ensure that it is being used and it is a living, as opposed to an abandoned, project. Be leery of those applications with just a few downloads. If there is a large group of people using the application, there s a higher probability that the application works as claimed.

At the same time, learning who some of the other users of this application are can give you some perception of how well it actually works. As Silva[31] reminds us, “the best insight you can get into a product is from another user who has been using it for a while.”

Suggested ratings are:

- Zero – No other discernible users or reality of listed users is questionable.
- One – Application is being used by multiple groups, but represents only a small fraction of its 'market'. Appears to be little 'buzz' about the application.
- Two – Application appears to be widely used and represents a significant fraction of its 'market'. This rating is enhanced if listed users include major commercial organizations.

Product Support

Product support can be a critically important topic for any application. Whether you are selecting a proprietary application or a FLOSS one, it is vitally important to ensure that you will have reliable support available. Just because you purchased a proprietary program will not ensure that you have the support you need. Some vendors include at least limited support in their contracts, others don't. However, over the years I've found that even purchasing a separate support contract doesn't ensure that the people who answer the phone will be able to help you. When making the final decision, don't make assumptions, research!

Support can be broken down into several different sub-categories:

- User Manuals – Do they exist? What is their quality?
- System Managers Manuals – Do they exist? What is their quality?
- Application Developers Manuals – Do they exist? What is their quality?
- System Design Manuals – Do they exist? What degree of detail do they provide?
- For data base related projects, is an accurate and detailed entity relationship (ERD) diagram included?
- Have any third party books been written about this application? Are they readily available, readable, and easy to interpret?
- Is product support provided directly by the application development community?
- Is product support provided by an independent Users Group, separate from the development group?
- Is commercial product support available?
    - If so, what are their rates?
    - Are on-site classes available?

    - Are on-line training classes available?
- A frequently overlooked type of product support is how well documented the program code is. Are embedded comments sparse or frequent and meaningful? Are the names of program variables and functions arbitrary or meaningful?

Another factor in evaluating product support is whether you have anyone on your team with the expertise to understand it. That is not a derogatory statement, depending on the issue, someone might have to modify an associated data base, the application code, or the code in a required library, whether to correct an error or add functionality. Do any of your people have expertise in that language? Would someone have to start from scratch or do you have the budget to hire an outside consultant? Even if you have no desire to modify the code, having someone on the project that understands the language used can be a big help in discerning how the program works., as well as determining how meaningful the program comments are.

Suggested ratings are:

- Zero – Limited or no support available. Documentation essentially non-existent, source code minimally documented, no user group support, and erratic response from the developers.
- One – Documentation scattered and of poor quality. Support from user group discouraged and no commercial support options exist.
- Two - Excellent documentation, including user, system manager, and developer documentation. Enthusiastic support from the user community and developers. Commercial third-part support available for those desiring it. Third party books may also have been released documenting the use of this product.

Maintenance

This is another item that can be interpreted in multiple ways. One way to look at it is how quickly the developers respond to any bug report. Depending on the particular FLOSS project, you may actually be able to review the problem logs for the system and see what the average response time was between a bug being reported and the problem resolved. In some cases this might be hours or days, in others it is never resolved. To be fair, don't base your decision on a single instance in the log file, as some bugs are much easier to find and fix than others. However, a constant stream of open bugs or bugs that have only been closed after months or years should make you leery.

To others this question is to determine whether development is still taking place on the project or if it is dead. Alternately, it is like asking if anybody is maintaining the system and correcting bugs when they are discovered. There are several ways of addressing this issue. Examining the problem logs described above is one way of checking for project activity, another is the check the release dates for different versions of the application. Are releases random or on a temporal schedule? How long has it been since the application was last updated? If the last release date was over a year or two ago, this is cause for concern and should trigger a closer look. Just because there hasn't been a recent release does not mean that the project has

been abandoned. If the development of the app has advanced to the point where it is stable and no other changes need to be made you may not see a recent release because none is needed. However, the later is very rare, both because bugs can be so insidious and because a lot of programmers can't resist just tweaking things, to make them a wee bit better.

If a project is inactive, but everything else regarding the project looks good, it might be possible to work with the developers to revitalize it. While this course of action is feasible, it is important to realize that it is taking on a great deal of responsibility and an unknown amount of expense. The latter is particularly true as you may be having to assign one or more developers to work on the project full time.

Suggested ratings are:

- Zero – No releases, change log activity, or active development discussion in message forums in over two years.
- One – No releases, change log activity or active development discussions in message forums for between one and two years.
- Two – A new version has been released within the year, change logs show recent development activity, and there is active development discussion in the message forums.

Reliability

Reliability is the degree to which you can rely on the application to function properly. Of course, the exact definition becomes somewhat more involved. The reliability of a system is defined as the ability of an application to operate properly under a specified set of conditions for a specified period of time. Fleming[63] states that “One aspect of this characteristic is *fault tolerance* that is the ability of a system to withstand component failure. For example if the network goes down for 20 seconds then comes back the system should be able to recover and continue functioning.”

Because of its nature, the reliability of a system is hard to measure, as you are basically waiting to see how frequently it goes down. While we'd like to aim for never, one should probably be satisfied if the system recovered properly after the failure. In most instances, unless you are actually testing a system under load, the best that you can hope for is to observe indicators from which you can infer its reliability. As a generality, the more mature a given code base is, the more reliable it is, but keep in mind that this is a generality, there are always incidents that can occur to destabilize every thing. Unfortunately, it frequently feels as if the problem turns out to be something that you would swear was totally unrelated. Face it, Murphy[64] is just cleverer than you.

Wheeler[36] also reminds us that “Problem reports are not necessarily a sign of poor reliability - people often complain about highly reliable programs, because their high reliability often leads both customers and engineers to extremely high expectations.” One thing that can be very reassuring is to see that the community takes reliability seriously by continually testing

the system during development.

Suggested ratings are:

- Zero – Error or bug tracking logs show a high incident of serious system problems. Perhaps worse, no logs of reported problems are kept at all, particularly for systems that have been in release for less than a year.
- One – Error logs show relatively few repeating or serious problems, particularly if these entries correlate with entries in the change logs indicating that a particular problem has been corrected. System has been in release for over a year.
- Two - Error logs are maintained, but show relatively few bug reports, with the majority of them being minor. A version of the system, using the same code base, has been in release for over two years. Developers both distribute and run a test suite to confirm proper system operation.

Performance

Performance of an application is always a concern. Depending on what the application is trying to do and how the developers coded the functions, you may encounter a program that works perfectly, but is just too unresponsive to use. Sometimes this is a matter of hardware, other times it is just inefficient coding, such as making sequential calls to a data base to return part of a block of data, rather than making a single call to return all of the block at once. Performance and Scalability are usually closely linked.

You might be able to obtain some information on the systems actual performance from the project web site, but it is hard to tell if this is for representative or selected data. Reviewing the project mailing list may provide a more accurate indication of the systems performance or any performance problems encountered. Testing the system under your working conditions is the only way to make certain what the systems actual performance is. Unfortunately, the steps involved in setting up such a test system require much more effort than a high level survey will allow. If any user reviews exist, they may give an insight into the systems performance. Locating other users through the project message board might be a very useful resource as well, particularly if they handle the same projected work loads that you are expecting.

It is difficult to define performance ratings without having knowledge of what the application is supposed to do. However, for systems that interact with a human operator, the time lapse between when a function is initiated and when the system responds can be suggestive. If the project maintains a test suite, particularly one containing sample data, reviewing its processing time can give an insight to the systems performance as well. Response delays of even a few seconds in frequently executed functions will not only kill the overall process performance, but result in users resistive to using the system. Suggested ratings are:

- Zero – A system designed to be interactive fails to respond in an acceptable time frame. For many types of applications it is reasonable to expect an almost instantaneous response, particularly for screen management functions. It is not reasonable for a system to take over a minute, or even 5 seconds to switch screens or

acknowledge an input, particularly in regards to frequently executed functions such as results entry, modification, or review.. A system that batch processes data maxes out under data loads below that of your current system.
- One – A system designed to be interactive appears to lag behind human entry for peripheral functions, but frequently accessed functions, such as results entry, modification, or review appear to respond almost instantaneously. A system that batch processes data maxes out under your existing data loads.
- Two – System is highly responsive, showing no annoying delayed responses. A system that batch processes data can process several times your current data load before maxing out.

Scalability

Scalability ensures that the application will operate over the data scale range you will be working with. In general, it means that if you test the system functionality with a low data load, the application will 'scale up' to handle larger data loads. This may be handled by expanding from a single processor to a larger cluster or parallel processing system. Note that for a given application, throwing more hardware at it may not resolve the problem, as the application needs to be designed to take advantage of that additional hardware. Another caveat is to carefully examine the flow of data through your system. The processor is not the only place you can encounter road blocks limiting scalability. Other possibilities include how quickly the system can access the needed data. If the system is processing the data faster than it can access it, adding more computer power will not resolve the problem. The limiting issues might be the bandwidth of your communication lines, the access speed of the devices that the data is stored on, or contention for needed resources with other applications. As with many aspects of selecting a system and getting it up and running, making assumptions is the real killer.

Silva[31] indicates that many Open Source applications are built on the LAMP (Linux, Apache, MySQL, PHP/Perl/Python) technology stack and that this is one of the most scalable configurations available. However, you should ensure that there is evidence that the application has been successfully tested that way. While performing surveys and testing is never a good time to start making assumptions. A look at the applications user base will likely identify someone who can provide this feedback.

Suggested ratings are:
- Zero – System does not support scaling, whether due to application design or restriction of critical resources, such as rate of data access..
- One – System supports limited scaling, but overhead or resource contention, such as a data bottleneck, results in a quick performance fall off.
- Two - System is balanced and scales well, supporting large processing clusters or cloud operations, without any restrictive resource pinch-points..

Usability

Usability means pretty much what it says. The concern here is not how well the program works, but rather, how easy is it to learn and use. The interface should be clear, intuitive, and help guide the user through the programs operation. Despite Steve Jobs, there is a limit to how intuitive an interface can be, thus the operation of the interface should be clearly documented in the user manual. Ideally the system will support a good context sensitive help system as well. The best help systems may also provide multimedia support, so that the system can actually show you how something should be done, rather than trying to tell you. I've found that frequently a good video can be worth well more than a thousand words! No matter how much time is spent writing the text for a manual or help system, it will always be unclear to somebody, if only because of the diversity of the backgrounds of people using it.

The interface between the operator and the computer may vary with the purpose of the application. While with new applications you are more likely to encounter a graphic user interface (GUI), there are still instances where you may encounter a command-line interface. Both types of interfaces have their advantages and there are many times when something is actually easier to do with a command-line interface. The important thing to remember is that it IS your interface with the system. It should be easy to submit commands to the system and interpret its response without having to hunt through a lot of extraneous information. This is normally best done by keeping the interface as clean and uncluttered as possible. As Abran, et al.[65], have pointed out, the usability of a given interface varies with “the nature of the user, the task and the environment”.

If you would prefer a somewhat drier set of definitions, Abran, et al., also extracted the definitions of usability from a variety of ISO standards and they are included in the following table:

ISO Usability Definitions
“The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.” (ISO/IEC9126-1, 2000)
“The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.” (ISO 9241-11, 1998)
“The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.” (IEEE Std. 610.12-1990)

Table 3: ISO Usability Definitions[65]

Fleming translates this into the somewhat more colloquial statement: “Usability only exists with regard to functionality and refers to the ease of use for a given function.” For those interested in learning more about the usability 'debate', I suggest that you check out Andreasen, et al.[66] and especially Saxena and Dubey[67].

In addition to usability, it has been highly recommended to me that, if budget exists, it is also helpful to have a User Experience (UX)[68,69] specialist on the review team as well.

Suggested ratings are:

- Zero – System is difficult to use, frequently requiring switching between multiple screens or menus to perform a simple function. Operation of system discourages use and can actively antagonize users.
- One – System is useable, but relatively unintuitive regarding how to perform a function. Both control and output displays tend to be cluttered, increasing the effort required to operate the system and interpret its output.
- Two – System is relatively intuitive and designed to help guide the user through its operation. Ideally, this is complemented with a context sensitive help system to minimize any uncertainties in operation, particularly for any rarely used functions.

Security

This heading overlaps with Functionality and is usually difficult to assess from a high-level evaluation. While it is unlikely that you will observe any obvious security issues during a survey of this type, there are indicators that can provide a hint as to how much the applications designers and developers were concerned with security.

The simplest approach is to simply take a look at whether they've done anything that shows a concern for possible security vulnerabilities. The following are a few potential indicators that you can look for, but just being observant when seeing a demonstration can also tell you a lot.

- Is there any mention of security in the systems documentation? Does it describe any potential holes that you need to guard against or configuration changes you might make to your system environment to reduce any risks.
- Do the manuals describe any type of procedure for reporting bugs or observed system issues?
- If you have access to the developers, discuss any existing process for reporting and tracking security issues.
- Check their error logs and see if any security related issues are listed. If they are, what was the turnaround time to have them repaired, or were they repaired?
- If the developers are security conscious, they will almost certainly want to prove to whomever received the program that it hadn't been modified by a third party. The basic way of doing this is by separately sending you what is known as an MD5 hash. This is a distinctive number generated by another program from your applications code. If you generate a new MD5 hash from the code you receive, these numbers should match. If they don't match, that means that something in the code has been altered. For developers with more concern, they might generate a cryptographic signature incorporating the code. This will tell you who sent the code, as well as indicate whether the program was altered.
- Depending on the type of program, does it allow a 21 CFR part 11 compliant

implementation or conform to a similar standard?
- Depending on the type of application, does it include a detailed audit trail and security logs?

Suggested ratings are:
- Zero – System shows no concern with security or operator tracking. Anyone can walk up to it and execute a function without having to log in. System doesn't support even a minimal audit trail. Any intermediate files are easily accessible and modifiable outside of the system.
- One - System shows some attempt at user control, but supports only a minimal audit trail. It may support a user table, but fails to follow best practices by allowing user records to be deleted. Audit trail is modifiable by power users.
- Two – Maintaining system security is emphasized in the user documentation. A detailed audit trail is maintained that logs all system changes and user activities. Application is distributed along with an MD5 hash or incorporated into an electronic signature by the developer.

Flexibility/Customizability

The goal of this topic is to identify how easily the functionality of this application can be altered or how capable it is of handling situations outside of its design parameters.. Systems are generally designed to be either configurable or customizable, sometimes with a combination of both.
- Configurability - This refers to how much, or how easily, the functionality of the system can be altered by changing configuration settings. Configurable changes do not require any changes to the application code and generally simplify future application upgrades.
- Customizability - This refers to whether the functionality of the system must be altered by modifying the applications code. As we are targeting open source systems, the initial assumption might be that they are all customizable, however, this can be affected by the type of license that the application is released under. More practically, how easily an application can be customized depends on how well it is designed and documented. While in theory you might be able to customize a system, if it is a mass of spaghetti code and poorly compartmentalized, it might be a nightmare to do. In any case, if you customize the system code, you may not be able to take advantage of any system upgrades without having to recreate the customizations in them.
- Extendability – While you won't find this term in most definitions, it is a hybrid system that is both configurable and customizable. It is normally configured using the same approaches as a standard configurable system. However, the ability to be upgraded remains by feeding any code customizations through an Application Program Interface (API). As long as this API is maintained between upgrades, any extension modules should continue to work.

In addition, a well designed application is usually modular, which makes program changes easier. In an ideal world, any application that you may have to customize will be specifically

designed to make customization simple. There are a variety of ways of doing this. Perhaps the easiest, for an application that is designed to be modular, would be to support optional software plug-in modules that added extra functionality. Unless these were 'off-the-shelf' modules, you would need to confirm that there was appropriate documentation regarding their design and use. This would most likely be done via an API, as discussed above. Depending on the systems, you could transfer data through the API or have one system control another. The caveat again being that you need to have thorough documentation of the API and its capabilities.

In the majority of situations, I strongly encourage you to stick with a configurable system, if you can find one that meets your needs. Customizing a system is rarely justified, unless you are working situation. While almost everyone feels that their needs are unique, the reality is that a vell designed configurable system can generally meet your needs.

Suggested ratings are:

- Zero – Application does not support configuration and shows evidence of being difficult to customize. The later being indicated by use of spaghetti code rather than modular design, poorly named variables and functions, along with cryptic or no embedded comments. In worst case situations, the source code has been deliberately obfuscated to make the system even less customizable.
- One – Application supports minor configuration capabilities or is moderately difficult to modify. The later might be due to minimal application documentation or poor programming practices, but not deliberate obfuscation.
- Two – Application is highly configurable and accompanied by detailed documentation guiding the user through its configuration. Code is clearly documented and commented. It also follows good programming practices with highly modularized functionality, simplifying customization of the programs source code, ideally via an API.

Interoperability/Integration

Determine whether this software will work with the rest of the systems that you plan to use. Exactly what to check for is up to you, as you are the only one who has any idea what you will be doing with it. The following is a list of possible items that might conceivably fall under this heading:

- Does it understand the data and control protocols to talk to and control external equipment, whether a drill press, telescope, or sewing machine? (as appropriate)
- Is it designed to conform to standards, both electronic and physical, to avoid being locked into a single supplier.
- Does it handle localization to avoid conflicts with local systems?

Suggested ratings are:

- Zero – System provides no support for integration with other applications. File formats and communication protocols used are not documented.

- One – System is not optimized for either interoperability or integration with other systems. However, it does use standard protocols so that other applications can interpret its activities. Application likely does not include an API or any existing API is undocumented.
- Two – Application is optimized for interoperability and integration with other systems. All interfaces and protocols, particularly for any existing API, are clearly documented and accompanied with sample code.

Legal/License Issues

This section refers to the type of license that the application was released under and the associated legal and functional implications. With proprietary software many people never bother to read the software license, either because they don't care or think they have no choice but to accept them. Be that as it may, when selecting a FLOSS application, it is wise to take the time to read the accompanying license, as it can make a big difference in what you can do with the software. First, if the software has no license, legally you have no right to even download the software, let alone run it.[70,71]

While you have no control over which license the application was released under, you definitely control whether you wish to use it under the terms of the license. Which types of licenses are acceptable strongly depends on what you plan to do with the application? Do you intend to use the application as is or do you plan to modify it? If the later, what do you pla to do with the modified code? Do you want to integrate this FLOSS application with another, either proprietary or Open Source? Does this license clash with theirs? Your right to do any of these things is controlled by the license, so it must be considered very carefully, both in the light of what you want to do now and what you might want to do in the future.

One of the first things to do is to confirm that the application even is Open Source, just being able to see the source code is insufficient. To qualify as Open Source the license must comply with the 10 points listed in the Open Source Definition maintained by the Open Source Initiative (OSI)[2]. Pulling just the headers, their web site lists these as the required criteria:

1. Free Redistribution
2. Source Code
3. Derived Works
4. Integrity of The Author's Source Code
5. No Discrimination Against Persons or Groups
6. No Discrimination Against Fields of Endeavor
7. Distribution of License
8. License Must Not Be Specific to a Product
9. License Must Not Restrict Other Software
10. License Must Be Technology-Neutral

While somewhat cryptic to look at cold, each of the headings is associated with a longer definition, which primarily boils down to the freedom to use, modify, and redistribute the

software. For those wanting to know the justification for each item, there is also an annotated version of this definition[72]. At present, OSI recognizes 71 distinct Open Source licenses, not counting the WTFPL[73].

One of the functions of the OSI is to review perspective licenses to determine whether they meet these criteria and are indeed Open Source. The OSI web site maintains a list of popular licenses, along with links to all approved licenses[74], sorted by name or category. These lists include full copies of the licenses. While clearer than most legal documents, they can still be somewhat confusing, particularly if you are trying to select one. If the definitions seem to blur, you might want to check out the *Software Licenses Explained in Plain English* web page maintained by TL;DRLegal[75]. As long as the license is classified as an Open Source license and you aren't planning to modify it yourself or integrate it into other systems, you probably won't have any problem. However, if you have any uncertainty at all it might be worth making the investment to discuss the license with an intellectual property lawyer who is familiar with OSS/FS before you inadvertently commit your organization to terms that conflict with their plans.

If your plans include the possible creation of complementary software, I suggest a quick read of Wheeler's essay on selecting a license[76]. The potential problem here is that most Open Source and proprietary applications contain multiple libraries or sub-applications, each with their own license. Depending on which licenses the original developers used, they may be compatible with other applications you wish to use, or they may be incompatible. The following figure illustrates some of these complications.

Just to be absolutely clear, the license of FLOSS and proprietary applications generally disavow any type of warranty that the program will work and disclaims any liability for any

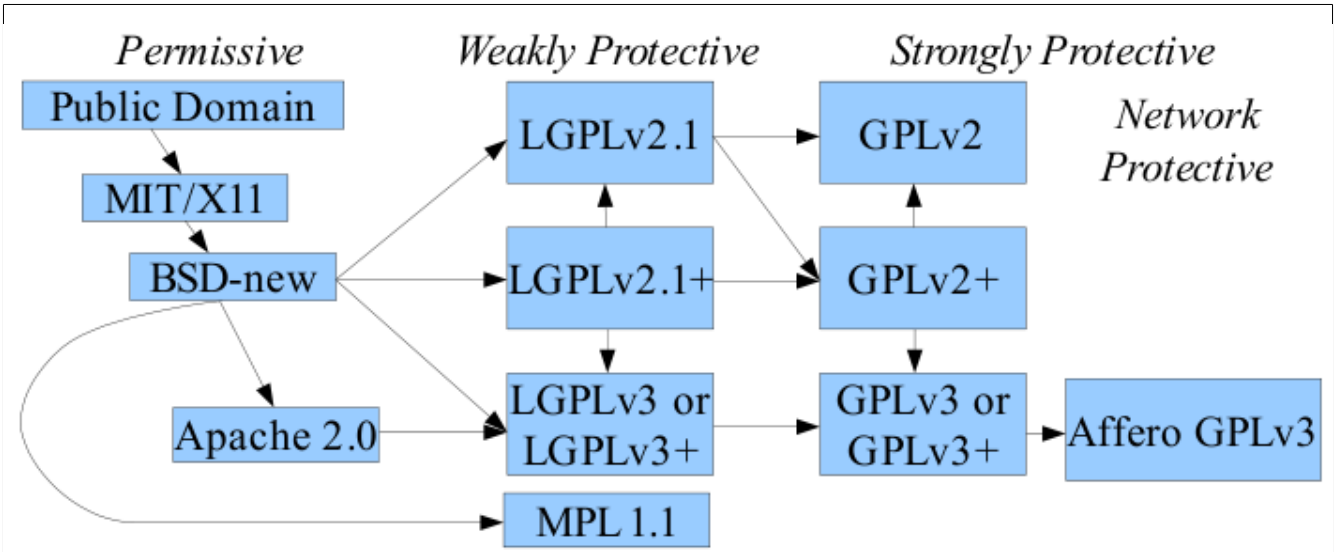

Illustration 3: License compatibility between common FOSS software licenses according to David A. Wheeler (2007): the vector arrows denote an one directional compatibility, therefore better compatibility on the left side than on the right side.[100]

damage or injuries that result from the programs use. Now, having said that, you may be able to purchase a warranty separately, just don't anticipate any legal recourse in the event of a system failure.

Suggested ratings are:

- Zero – Application does not include a license or license terms are unacceptable or incompatible with those of other applications being used.
- One – Application includes a license, but contains potential conflicts with other licenses or allowable use that will need to be carefully reviewed.
- Two – License is fully open, allowing you to freely use the software.

Caveats

Other than independent reviews, one of the best ways to obtain some of the above information is direct testing of a system. Usually that is impractical in a survey situation because of the time it would take to install and configure an instance of the application. However, a new factor has entered the picture which may change this. Docker[77] is a new service that combines an application, along with all of its dependencies into a single distributable package, which they term a Container. Because everything required is in the container, it is guaranteed to run the same on any system. Docker specifically states that their containers “are based on open standards allowing containers to run on all major Linux distributions and Microsoft operating systems with support for every infrastructure.” Using only a fraction of the resources that a Virtual Machine (VM) would require, you can easily run multiple containers on a laptop and switch back and forth while testing[78].

While the Docker web site contains repositories for a number of different containers, there are multiple web sites that also host containers configured with a variety of applications. If you can locate one that contains the application you are surveying, this makes it a simple matter to try it out. Probably the easiest way to check is to run a web search containing the terms 'docker', 'container, and the name of the application that you are seeking.

Screening Tabulation

While you can perform this application survey in many ways, to keep your defined scoring criteria in front of the evaluators and to simplify scoring the survey, it might be prudent, to generate a document such as the following table with columns representing the criteria being evaluated, your rating definitions, and the numeric rating that your evaluators actually assign to the system. If you have decided to take the approach of using weighting factors instead of adjusting your definitions, you will also need to include columns for your weighting factors and a column containing the results of the evaluation after applying the weighting factor.

Criteria	Rating = 0	Rating = 1	Rating =2	Score
System Functionality				
Community				
System Cost				
Popularity				
Product Support				
Maintenance				
Reliability				
Performance				
Scalability				
Usability				
Security				
Flexibility/ Customizability				
Interoperability/ Integration				
Legal/License Issues				
Summary Score				

Table 4 Potential screening criteria to filter prospective FLOSS applications. Rows can be added or dropped as required by your needs. For example, if it is important that the application be coded in a specific programming language or combination of languages, you could add a row for this. The most likely anticipated change would be to subdivide the System Functionality row so that specific information on the functionality of various system components can be displayed. To minimize the risk of confusion, you can transfer your selected scoring criteria for each rating into the cell corresponding to that rating and the item being evaluated, though ths would admittedly generate a long check list.

Demonstration Surveys

To demonstrate how this survey procedure is intended to be used, this section will apply the defined protocol to a block of Open Source LIMS applications suitable for use in a chemical laboratory. Note that in this instance, we are using LIMS to refer to a Laboratory Information Management System, not a Labor Information Management System, a Legislative Information Management System, or any other permutation that matches the LIMS acronym. The specific criteria used will quite likely be different from the ones used in your screening as the types of systems or the specific functionality required will be different.

For the purpose of this demonstration, we will not attempt to screen all of the Open Source LIMS available. Instead, we will select a subset of the systems that have been announced and attempt to apply our protocol to them, hopefully screening out the systems not meeting our requirements, so that the number of in-depth evaluations can be reduced. Part of the difficulty here is the broad range of fields that the term LIMS covers. There are general purpose LIMS, which can be configured to handle a wide range of samples, as well as targeted LIMS, which are designed to fill a particular niche. As such, you can find LIMS targeting specific areas as drinking water/waste water analysis, mining, radioisotopes, proteomics, and genetic analysis. Whether you make it a formal part of the screening process or isolate them while collecting the applications to be surveyed, you will need to filter out the systems which will not handle the type of samples you are dealing with. This issue is particularly prominent with LIMS, but will likely be encountered when screening other types of software as well. The following are the systems that we will include in this attempt:

- Bika LIMS
- eyeLIMS
- Open-LIMS

For simplicity in comparing results, I've reordered the screening table so that the first column contains the criteria being evaluated and the other columns correspond to the evaluation results for the LIMS being evaluated, with the bottom row reserved for the corresponding score summary. I have also added two subdivisions under System Functionality for the Programming Language and the Operating System used. Depending on the types of systems you are surveying, you will likely be including additional subdivisions. Scoring can be handled several ways. In this example, you might list the programming language in the appropriate cell. Depending on whether you have a team member competent in that language, you can use the results to set the value for the main criteria, e.g. System Functionality. Alternately, while you will record the information for programming language in your survey notebook, you can insert an actual numerical result into the corresponding cell, so that the sub-criteria can be evaluated separately, or used to generate a value for the main criteria field.

Normally, the best approach starting out is to check for existing reviews of systems, but here we have a problem in finding any. While references to Bika LIMS were common, and it was referenced in a number of scientific papers, actual reviews of the product were hard to come by and were usually for older versions[79]. No reviews were found for eyeLIMS and for Open-LIMS, the closest thing I found to an impartial review has been two postings on Joel Limardo's LIMSexpert blog[80,81].

Criteria	Bika LIMS 3.1.8	eyeLIMS	Open-LIMS[2]
System Functionality	2	0 – Project appears dead, no activity since 2008.	1 – System functional for a very specific application only.
Programming Language	Python and PLONE CMS	eyeOS	PHP
Operating System	platform-independent	eyeOS	Any OS supporting PostgreSQL and PHP
Community	2	0	1
System Cost		0	1
Popularity	2	0	0
Product Support	2	0	1
Maintenance	2	0	1
Reliability		0	
Performance		0	
Scalability		0	
Usability	2	0	1
Security	2	0	1
Flexibility/ Customizability	2	0	1
Interoperability/ Integration	2? Supports import and export of CSV files.	0	0
Legal/License Issues	2 - AGPL-3.0 and GPL1	2 - GNU Affero GPL v3	2 - GNU General Public License version 3.0 (GPLv3)
Summary Score	**22**	**2**	**10**

Based on the above Summary Scores, we would definitely filter out both eyeLIMS and Open-LIMS, while Bika LIMS would justify a more in-depth evaluation.

Summary

2 Note: The rating for Open-LIMS may need to be revisited, as it appears that major development work on this system has been taken over by Joel Limardo of ForwardPhase Technologies, so several of the rated parameters may be subject to major shifts. It is currently unclear whether this is a joint project with the original developer or a fork.

In support of a project to prepare published evaluations of various FLOSS applications, we have reviewed the FLOSS literature, focusing particularly on any assessment or evaluation documents. While many described proposed evaluation methods, none of them appear to be particularly popular or have developed an active community around them. Several review papers on the topic, while identifying multiple methods and their advantages, found flaws in all of them, particularly in terms of being able to perform a quality assessment on any FLOSS application. Many of the described systems were explicitly focused on a single class of FLOSS applications, such as Library Management Systems.

By consolidating suggestions and procedures from a number of these papers, we synthesized a general survey process to allow us to quickly assess the status of any given type of FLOSS applications, allowing us to triage them and identify the most promising candidates for in-depth evaluation. Note that this process is designed for performing high-level surveys, it is not designed to perform the in-depth evaluations required for product selection.

As a minor aside, in the course of researching this article I was surprised by the high percentage of the published papers on FLOSS which were published in classic subscription journals, as opposed to any of the various Open Source journals available[82]. Somehow it seems like a curious disconnect not to publish articles on Open Source Software in Open Source journals. Whether this is just habit of submission or due to more considered reasons would be interesting to know.

Glossary[3]

.ogg	File Extension for **Ogg Vorbis**, an open source patent-free audio compression format
21 CFR part 11	United States Food and Drug Administration Electronic Records and Electronic Signatures rule.
Academia.edu	A web site to allow academics to share research papers and exchange information. [https://www.academia.edu/]
Accuracy	Accuracy defines how well the results of an analysis or measurement conforms to the actual, or “correct”, value.
AEQ	**A**nalytical **E**quipment **Q**ualification
agnostic	A term applied to both hardware and software that indicates that the item is interoperable with different systems. The correct term should probably be technology independent or technology neutral, but the use of the term agnostic appears to be well entrenched.
AHP	**A**nalytic **H**ierarchy **P**rocess (**AHP**) - A structured technique developed by Thomas Saaty in the 1970's for organizing and analyzing complex decisions.
AIQ	**A**nalytical **I**nstrument **Q**ualification (**AIQ**) – Term used in the pharmaceutical industry for the process of ensuring that an instrument meets the requirements for its intended application.
algorithmic transparency	
Android OS	
android sdk	
Ant	
Apache	
Apache Cassandra	
Apache Hadoop YARN	**Y**et **A**nother **R**esource **N**egotiator is a technology for managing resources on a cluster of computers.

3 Some definitions have been pulled from Wikipedia in an attempt to keep all description nuances intact. In such cases, the definition will be followed by a reference number linking back to the full definition in Wikipedia.

Apache Lucene	A software library for information retrieval from fields of text contained within document files.
ASE	**A**daptive **S**erver **E**nterprise
ATA over Ethernet (AoE)	
Audit Trail	A log of records documenting the sequence of activities that have been performed on a system.
Binary Package Distribution	A compilation of the compiled version of a program and all related documentation designed for release to the end user.
boot loader	
BREW	**B**inary **R**untime **E**nvironment for **W**ireless – A runtime and application development environment from Qualcomm that isolates portable applications from the hardware interface of mobile phones employing Code Division Multiplex Access.
BRR	**B**usiness **R**eadiness **R**ating
Cassandra	See Apache Cassandra
CDS	**C**hromatography **D**ata **S**ystem (**CDS**)
CFR	United States **C**ode of **F**ederal **R**egulations
Change Log	A log documenting the changes made to a software product. May include a list of new features, changes to behavior, or elimination of software bugs.
COC	**C**hain **o**f **C**ustody (**COC**) – A documentation trail, whether paper or electronic documenting responsibility of a sample. This is required for legal reasons under various regulatory programs, as well providing information used to track faulty or contaminated items back to their source
Committer	Represent the Quality Control of the community. They control what changes are included in the originally licensed version, though users are free to make any changes they want in their own copies of the program
Community	
Compiere	A suite of Open Source applications targeting small to medium sized businesses that provides a number of business support applications.
Configuration	
Container	

Copy Left	**Copyleft** (a play on the word copyright) is the practice of offering people the right to freely distribute copies and modified versions of a work with the stipulation that the same rights be preserved in derivative works down the line. Copyleft is a form of licensing, and can be used to maintain copyright conditions for works ranging from computer software, to documents, to art. In general, copyright law is used by an author to prohibit recipients from reproducing, adapting, or distributing copies of their work. In contrast, under copyleft, an author may give every person who receives a copy of the work permission to reproduce, adapt, or distribute it, with the accompanying requirement that any resulting copies or adaptations are also bound by the same licensing agreement. Copyleft licenses (for software) require that information necessary for reproducing and modifying the work must be made available to recipients of the binaries. The source code files will usually contain a copy of the license terms and acknowledge the author(s). Copyleft type licenses are a novel use of existing copyright law to ensure a work remains freely available. The GNU General Public License, originally written by Richard Stallman, was the first software copyleft license to see extensive use, and continues to dominate in that area. Creative Commons, a non-profit organization founded by Lawrence Lessig, provides a similar license provision condition called ShareAlike. [83]
Cosmos	**C# O**pen **S**ource **M**anaged **O**perating **S**ystem
COTS	**C**ommercial, **O**ff-**t**he-**S**helf (**COTS**)
Creative Commons	
CROMERR	**Cro**ss-**M**edia **El**ectronic **R**eporting **R**ule (**CROMERR**) This is an Environmental Protection Agency (EPA) rule specifying how electronic reporting should be performed for the EPA's various regulatory programs.
Customization	
Cygwin	A collection of tools that emulate a Linux environment, allowing Linux applications to be compiled for and executed in a MS Windows environment.
Data Loading	The loading of data into static tables. This data includes test definitions, sample container descriptions, location information, etc
database-agnostic	Indicates an application capable of running with data base systems from any vendor.

digital commons	Name given to a collaboratively developed on-line resource that is managed by a community of people.
DMOZ	Also known as the **O**pen **D**irectory **P**roject (**ODP**), this is an attempt to create the largest human curated open-content directory of web links. **DMOZ** comes from one of its earlier domain names, ***d****irectory.****moz****illa.org*.
Docker	Docker is a system designed to package an application with all of its dependencies into a standardized software container. It takes an alternate architectural approach than a virtual machine, while providing similar resource isolation in a smaller footprint.[77]
Documentation, Administrator	Documentation for a system designed to be issued to a System Administrator. It generally is designed to provide information on configuring and operating the application.
Documentation, Developer	Documentation for a system designed to be issued to a System Developer. This type of documentation includes detailed information on how the system operates and is structured. It provides information to allow a developer to alter the system code and to extract data from the system in an ad hoc. manner.
Documentation, User	Documentation for a system designed to be issued to a user. It is designed to help guide the user through using the application.
DOI	**D**igital **O**bject **I**dentifier – A character string identifier used to uniquely identify an electronic document. (ISO 26324) In many instances a scientific/technical publication will have a **DOI** printed on it, allowing you to access or purchase the article on-line.
Drizzle	A MySQL 6.0 derived data base optimized for cloud computing.
Drupal	An Open Source application designed for creating and managing a variety of web sites.
Eclipse	A Java-based integrated development environment that can be customized via use of community developed plug-ins.
EDD	**E**lectronic **D**ata **D**elivery/ **E**lectronic **D**ata **D**eliverable (**EDD**)
EHR	**E**lectronic **H**ealth **R**ecord (**EHR**)
ElasticSearch	An Open Source search engine, employing a RESTful (see **REST**) interface, built on Apache Lucene.
ELN	**E**lectronic **L**aboratory **N**otebook (**ELN**)
EMR	**El**ectronic **M**edical **R**ecord (**EMR**)
EPA	United States **E**nvironmental **P**rotection **A**gency [http://www.epa.gov/]

ERP	**E**nterprise **R**esource **P**lanning (**ERP**) generally constitutes a variety of integrated applications, with a shared data base, that integrates critical buisiness functions, such as accounting, human resources, customer relationship management, inventory and order tracking, et al., into a single system.
Evolvability	
F/OSS	**F**ree**/O**pen **S**ource **S**oftware. See FOSS
FAME	**F**ilter, **A**nalyze, **M**easure, and **E**valuate (**FAME**) methodology for evaluating Open Source applications.
FDA	United States **F**ood and **D**rug **A**dministration [http://www.fda.gov/]
Fedora	A version of the Linux operating system sponsored by Red Hat
Firefox	One of a number of Open Source web browsers. Can be highly customized via the use of member developed 'plug-ins'.
FLOSS	"**FLOSS**" was used in 2001 as a project acronym by Rishab Aiyer Ghosh for **free/libre and open-source software**. Later that year, the European Commission (EC) used the phrase when they funded a study on the topic. Unlike "libre software", which aimed to solve the ambiguity problem, "FLOSS" aimed to avoid taking sides in the debate over whether it was better to say "free software" or to say "open-source software".[84]
FLOSShub	As per their web site, "**FLOSShub** is a portal for free/libre and open source software (FLOSS) research resources and discussion. FLOSShub's goal is to provide a central location for connecting researchers and FLOSS community members to research papers, data, tools, and most importantly, community." [85]
Forge	In FLOSS development communities, a **forge** is a web-based collaborative software platform for both developing and sharing computer applications. (The word derives from the metalworking forge, used for shaping metal parts.) A forge platform is generally able to host multiple independent projects. For software developers it is a place to host, among others, source code (often version-controlled), bug database and documentation for their projects. For users, a forge is a repository of computer applications. Software forges have become popular, and have proven successful as a software development model for a large number of software projects. The term forge refers to a common prefix or suffix adopted by various platforms created after the example of SourceForge (such as GForge and FusionForge).[86]

Forking	This term is used to describe the process where a group takes an Open Source applications source code and starts developing it in an independent direction from the original program.
FOSS	**Free and open-source software** (**FOSS**) is computer software that can be classified as both free software and open-source software.[a] That is, anyone is freely licensed to use, copy, study, and change the software in any way, and the source code is openly shared so that people are encouraged to voluntarily improve the design of the software.[3] This is in contrast to proprietary software, where the software is under restrictive copyright and the source code is usually hidden from the users. [87]
Free Software	See Software, Free
Free Software Foundation (FSF)	The **Free Software Foundation** (**FSF**) is a 501(c)(3) non-profit organization founded by Richard Stallman on 4 October 1985 to support the free software movement, which promotes the universal freedom to study, distribute, create, and modify computer software, with the organization's preference for software being distributed under copyleft ("share alike") terms, such as with its own GNU General Public License. The FSF was incorporated in Massachusetts, USA, where it is also based. From its founding until the mid-1990s, FSF's funds were mostly used to employ software developers to write free software for the GNU Project. Since the mid-1990s, the FSF's employees and volunteers have mostly worked on legal and structural issues for the free software movement and the free software community. Consistent with its goals, only free software is used on the FSF's computers.[88]
Freeware	**Freeware** (portmanteau of "free" and "software") is computer software that is available for use at no monetary cost, which may have restrictions such as redistribution prohibited, and for which source code is not available. Freeware, although itself free of charge, may be intended to benefit its owner, e.g. by encouraging sales of a more capable version. According to the Free Software Foundation (FSF), "freeware" is a loosely defined category and it has no clear accepted definition, although FSF asks that free software (libre; unrestricted and with source code available) should not be called freeware. Examples of closed-source freeware include Adobe Reader and Skype.[89]

Functional Requirements	In software engineering (and systems engineering), a **functional requirement** defines a function of a system and its components. A function is described as a set of inputs, the behavior, and outputs (see also software). Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases. Functional requirements are supported by non-functional requirements (also known as quality requirements), which impose constraints on the design or implementation (such as performance requirements, security, or reliability). Generally, functional requirements are expressed in the form "system must do <requirement>", while non-functional requirements are "system shall be <requirement>". The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture.[90]
Functional Specification Requirements	
GALP	Good Analytical Laboratory Practices
GCP	Good Clinical Practice
GLP	Good Laboratory Practice
GMP	Good Manufacturing Practice
GNU	GNU’s Not Unix (a project to create a FLOSS operating system)

GNU General Public License	The **GNU General Public License** (**GNU GPL** or **GPL**) is the most widely used[6] free software license, which guarantees end users (individuals, organizations, companies) the freedoms to run, study, share (copy), and modify the software. Software that allows these rights is called free software and, if the software is copylefted, requires those rights to be retained. The GPL demands both. The license was originally written by Richard Stallman of the Free Software Foundation (FSF) for the GNU project. In other words, the GPL grants the recipients of a computer program the rights of the Free Software Definition and uses copyleft to ensure the freedoms are preserved whenever the work is distributed, even when the work is changed or added to. The GPL is a copyleft license, which means that derived works can only be distributed under the same license terms. This is in distinction to permissive free software licenses, of which the BSD licenses and the MIT License are the standard examples. GPL was the first copyleft license for general use.[91]
GNU Project	The **GNU Project** is a free software, mass collaboration project, announced on 27 September 1983, by Richard Stallman at MIT. Its aim is to give computer users freedom and control in their use of their computers and computing devices, by collaboratively developing and providing software that is based on the following freedom rights: users are free to run the software, share it (copy, distribute), study it and modify it. GNU software guarantees these freedom-rights legally (via its license), and is therefore free software; the use of the word "free" always being taken to refer to freedom. In order to ensure that the entire software of a computer grants its users all freedom rights (use, share, study, modify), even the most fundamental and important part, the operating system (including all its numerous utility programs), needed to be written. The founding goal of the project was, in the words of its initial announcement, to develop "a sufficient body of free software [...] to get along without any software that is not free." Stallman decided to call this operating system GNU (a recursive acronym meaning "GNU's not Unix"), basing its design on that of Unix; however, in contrast to Unix which was proprietary software, GNU was to be freedom-respecting software (free software) that users can use, share, study and modify. Development was initiated in January 1984. The goal of making a completely free software operating system was achieved in 1992 when the third-party Linux kernel was released as free software, under version 2 of the GNU General Public License, to be used with the GNU software stack.[92]
Google Scholar	A version of Google search optimized for searching and retrieving scholarly papers.[51]

Governance	
GPL	GNU **G**eneral **P**ublic **L**icense (the most common FLOSS license)
HDL	**H**ardware **D**escription **L**anguage -
HL7	A non-profit organization working to define standard methods for the exchange and retrieval of medical information.
IEC	**I**nternational **E**lectrotechnical **C**ommission
illegal operation	Term for an operating system command that is unknown to the operating system or processor.
Interface	
ISO	**ISO** (International Organization for Standardization) is an independent, non-governmental membership organization and the world's largest developer of voluntary International Standards.
ISO/JTC 1/SC 7, ISO/IEC 9126 Software Engineering - Product Quality (Part1-4)	
Jabber	An XML based instant messaging platform
LIMS	**L**aboratory **I**nformation **M**anagement **S**ystem – An informatics system designed to track samples and analytical results through a laboratory.
LIS	**L**aboratory **I**nformation **S**ystem – An informatics system designed to track samples and test results through a clinical laboratory
LMS	**L**ibrary **M**anagement **S**ystem
Maturity Level	
Maturity Rating	
Microsoft Academic Search	An experimental Microsoft search project focusing on scientific and technical information.[52]
Modular	A design process where an application is broken down into smaller parts, frequently grouped by function.
MongoDB	An Open Source NoSQL document-oriented data base.
native code	This is the name given to a program that is designed to run with a specific computers hardware program codes. It can only be run on a different computer/processor via the use of an emulator.
Non-Functional Requirements	List of requirements describing how the system works and how it should behave.

OALib	An Open Access journal [http://www.oalib.com/]
ODBC	**O**pen **D**ata**b**ase **C**onnectivity (ODBC) allows an application to connect with any data base.
ODF	The **O**pen **D**ocument **F**ormat is an XML based Open Source document format developed for office suite use. Primarily found in the OpenOffice and LibreOffice projects.
ODP	See **O**pen **D**irectory **P**roject.
Open **D**irectory **P**roject	**ODP**. See **DMOZ**
Open Source Development Labs	**O**pen **S**ource **D**evelopment **L**abs (**OSDL**) was conceived as a facility to allow Open Source developers to collaborate to create standardized implementations of Linux and Linux Applications by by IBM, Intel, and Computer Associates
Open Source Initiative (OSI)	The **O**pen **S**ource **I**nitiative (**OSI**) is a global non profit organization promoting Open Source and preventing abuse of the Open Source concept. It is also the steward of the **O**pen **S**ource **D**efinition (**OSD**).
OpenBRR	**Open B**usiness **R**eadiness **R**ating i(**OpenBRR**) is a methodology for evaluating Open Source applications.
OpenDOAR	**D**irectory of **O**pen **A**ccess **R**epositories
OpenLogic Exchange	**S**oftware-**a**s-**a**-**S**ervice (**SaaS**) governance platform for comprehensive governance and provisioning of open source software
OpenSSL	A general purpose encryption library used in many web servers. Among other features, it provides an implementation of the **I**nternet's **S**ecure **S**ockets **L**ayer (**SSL**) and **T**ransport **L**ayer **S**ecurity (**TLS**) protocols
OpenVPN	**O**pen source **Vi**rtual **P**rivate **N**etwork – See VPN
OPML	**O**utline **P**rocessor **M**arkup **L**anguage – An XML based file format used for creating outlines.
OS	**O**perating **S**ystem
OSDL	See **O**pen **S**ource **D**evelopment **L**abs (OSDL)

OSS	**Open-source software** (**OSS**) is computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose. Open-source software may be developed in a collaborative public manner. Open-source software is the most prominent example of open-source development and often compared to (technically defined) user-generated content or (legally defined) open-content movements.[93]
OSS-Watch	[http://oss-watch.ac.uk/]
PDA	**P**arenteral **D**rug **A**ssociation
PDF	Adobe **P**ortable **D**ocument **F**ormat
Perl	**Perl** is an interpreted script programming language invented by Larry Wall that is popular for creating common gateway interface (CGI) programs.
Precision	Precision is defined by how closely a set of measurements agree with their average.
Proprietary Software	See Software, Proprietary
PROSE	**P**romoting **O**pen **S**ource in **E**urope [http://opensourcepr ojects.eu/]
QA	**Q**uality **A**ssurance – Responsible for auditing the QC process and ensuring that all SOPs and standards were being followed.
QC	**Q**uality **C**ontrol – Responsible for ensuring all parts of a process were within designed specifications. It is this group that is responsible for designated testing and inspection.
QMS	**Q**uality **M**anagement **S**ystem
QSOS	**Q**ualification and **S**election **O**pen **S**ource (**QSOS**) backed by Atos Origin is a methodology for evaluating Open Source applications.
Repeatability	
Representational State Transfer	**Re**presentational **S**tate **T**ransfer (**REST** or occasionally **ReST)** is an architectural style commonly used for APIs that is defined by six constraints: Uniform Interface, Stateless, Cacheable, Client-Server, Layered System, and Code on Demand.[94,95]
Reproducibility	

ResearchGate	A professional network allowing scientists and researchers to exchange papers and collaborate. [http://www.ResearchGate.net/]
REST	See Representational State Transfer.
Revenue trigger	Revenue triggers are ways of bringing in money to support the program development. Options range from ad sales to training, consulting contracts, and support subscriptions.
RFI	**R**equest **f**or **I**nformation - A document sent by potential customers to vendors requesting detailed information regarding their system, usually in relation to selecting a hardware or software system to acquire.
RFID	**R**adio **F**requency **I**dentification **D**evice
RFP	**Request for Proposal** – Document sent to vendors specifying application requirements and requesting a proposal of what they can supply.
Ruby	**Ruby** is an open source object-oriented scripting/programming language created by Yukihiro Matsumoto.
Samba	Samba is an implementation of **S**erver **M**essage **B**lock (SMB) and **C**ommon **I**nternet **F**ile **S**ystem (CIFS) client/server protocols that allows shared access of resources over a network.

Shareware	Shareware is a type of proprietary software which is provided (initially) free of charge to users, who are allowed and encouraged to make and share copies of the program, which helps to distribute it. The word "shareware" is a portmanteau combining the words "share" and "software". Shareware is often offered as a download from an Internet website or as a compact disc included with a magazine. There are many types of shareware, and while they may not require an initial up-front payment, all are intended to generate revenue in one way or another. Some limit use to personal non-commercial purposes only, with purchase of a license required for use in a business enterprise. The software itself may be limited in functionality or be time-limited. Or it may remind you that payment would be appreciated. Shareware is available on all major personal computer platforms. Titles cover a very wide range of categories including: business, software development, education, home, multimedia, design, drivers, games, and utilities. Because of its minimal overhead and low cost, the shareware model is often the only one practical for distributing non-free software for abandoned or orphaned platforms such as the Atari ST and Amiga. The term shareware is used in contrast to open-source software, in which the source code is available for anyone to inspect and alter, and freeware, which is software distributed at no cost to the user but without source code being made available. Note that two types of shareware, donationware and freemiums, are also types of freeware.[96]
Smalltalk	An experimental language developed at Xerox in the 1970's to investigate the concept of object-oriented programming
SME	**S**ubject **M**atter **E**xpert (**SME**) – A person who is an authority on the particular subject area or topic that a software application addresses.

Software, Free	**Free software**, **software libre**, or **libre software** is computer software that gives users the freedom to run the software for any purpose as well as to study, modify, and distribute the original software and the adapted versions. The rights to study and modify free software imply unfettered access to its source code. For computer programs which are covered by copyright law this is achieved with a software license where the author grants users the aforementioned freedoms. Software which is not covered by copyright law, such as software in the public domain can also be free if the source code is in the public domain (or otherwise available without restrictions). Other legal and technical aspects such as software patents and DRM may impede users from exercising these rights, and thus prevent software from being free. Free software may be developed collaboratively by volunteer computer programmers or by corporations; as part of a commercial activity or not. Free software is primarily a matter of liberty, not price: users, individually or collectively, are free to do whatever they want with it – this includes the freedom to redistribute the software free of charge, or to sell it (or related services such as support or warranty) for profit.[8] Free software thus differs from proprietary software (such as Microsoft Windows), which to varying degrees prevents users from studying, modifying and sharing the software. Free software is also distinct from freeware, which is simply a category of proprietary software which does not require payment for use. Proprietary software (including freeware) uses restrictive software licences or user agreements and usually does not provide access to the source code. Users are thus prevented from modifying the software, and this results in the user becoming dependent on software companies to provide updates and support (vendor lock-in). Users can also not necessarily reverse engineer, modify, or redistribute proprietary software.[97]
Software, Open Source	**O**pen-**S**ource **S**oftware (**OSS**) is computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose.[Open-source software may be developed in a collaborative public manner. Open-source software is the most prominent example of open-source development and often compared to (technically defined) user-generated content or (legally defined) open-content movements.[82]

Software, Proprietary	**Proprietary software**, non-free software (in the sense of missing freedoms[1]), or closed-source software is software, where the developers or distributors reserve all freedoms and rights. Among the freedoms and rights that proprietary software deprives (to end-users), are: the freedom to analyze the software, and to change it (often deprived through intentional non-availability of sourcecode, or through Non-disclosure agreements (NDA)) the freedom to share the software (often deprived through copy prohibition via EULA (End User License Agreement) or NDA) the freedom to run the software for any purpose (often deprived through user-restrictions via EULA). In contrast to proprietary software, free software, is software that grants a user all these freedoms, on reception of the software. Proprietary software is licensed under legal right of the copyright holder, with the intent that the licensee is given the right to use the software only under certain conditions, and restricted from other uses, such as modification, sharing, studying, redistribution, or reverse engineering. Usually the source code of proprietary software is not made available. Complementary terms include free software, licensed by the owner under more permissive terms, and public domain software, which is not subject to copyright and can be used for any purpose. Proponents of free and open-source software use proprietary or non-free to describe software that is not free or open-source. A related, but distinct categorization in the software industry is commercial software, which refers to software produced for sale, but without meaning it is closed-source.[98]
SOP	**S**tandard **O**perating **P**rocedure
Source Code	A version of the program in human readable form.
Source Package Distribution	This is a consolidation of all of the source code and documentation relevant to a specific software release. It is intended for the application developer and/or maintainer.
SourceForge	A software repository on the web providing support for Open Source developers and for distribution of their application

Spaghetti code	A contemptuous phrase applied to unstructured applications, particularly those with GOTO statements or other control structures bouncing control around different portions of the program.
SQA	**Software Quality Assurance** - Group responsible for auditing the SQC process and ensuring that all SOPs and standards were being followed.
SQC	**Software Quality Control** – Group responsible for ensuring all parts of a process were within designed specifications. It is this group that is responsible for designated testing and inspection.
SQL	**S**tructured **Q**uery **L**anguage
Subversion	A program designed by Karl Fogel and Ben Collins-Sussman to serve as a version control system that tracks changes made to files and folders.
System Suitability Testing	The concept of System Suitability Testing, as applied to analytical procedures, is that everything involved in the analysis (analytical equipment, data capture electronics, analytical procedures, and analytical samples) constitute a system and can be evaluated as such.
tcpdump	A command line tool for monitoring network traffic. It can be very useful in troubleshooting network problems.
Tsunami UDP	A file transfer protocol enabling high speed data transfers over networks with large end-to-end delays. It is often used for bulk data transfers.
Uniform Resource Identifier	A **U**niform **R**esource **I**dentifier (**URI**) is simply a string of characters used to identify the name of a resource. There are multiple types of URIs, including **U**niversal **R**esource **L**ocator (**URL**), which defines a resource on the World Wide Web and its location, and **U**niversal **R**esource **N**ame (**URN**), which identifies a resource on the World Wide Web, but not its location.[99]
United States Pharmacopeial Convention	According to The **U**nited **S**tates **P**harmacopeial Convention (**USP**) web site, “The U.S. Pharmacopeial Convention (USP) is a scientific nonprofit organization that sets standards for the identity, strength, quality, and purity of medicines, food ingredients, and dietary supplements manufactured, distributed and consumed worldwide. USP’s drug standards are enforceable in the United States by the Food and Drug Administration, and these standards are used in more than 140 countries.”
Universal Resource Locator	A **U**niversal **R**esource **L**ocator (**URL**) is a specific type of **U**niform **R**esource **I**dentifier (**URI**). A URL identifies objects on the World Wide Web, including their address.
URI	See **U**niform **R**esource **I**dentifier (**URI**)
URL	See **U**niversal **R**esource **L**ocator (**URL**)

URS	**User Requirements Specification** – Document that lists all of the requirements that the users require the system to support. This is frequently considered the key document in regards to the development life cycle of the application.
User Experience (UX)	In simplest terms, this describes the overall aspects that an end-user experiences when interacting with the application. It is distinct from the functionality of the User Interface
USP	See **U**nited **S**tates **P**harmacopeial Convention (**USP**)
Validation	Confirming that the application works as designed.
vendor neutral	A business approach designed to avoid lock-in with a particular supplier and ensure broad compatibility and interchangeability of products and technologies.
Versioning	In relation to software, **versioning** is assigning a unique name or number to a given compilation of the software. Frequently this is in terms of a major and minor version number.
VM	**Virtual Machine** – A program running on another computer that emulates a full computer system. Originally developed to allow more efficient use of idle time on existing hardware servers.
WAN	**W**ide **A**rea **N**etwork - A computer network connecting multiple locations, as opposed to a Local Area Network t(LAN) hat services one location.
Wiki	A server program designed to promote collaborating development. Wiki's have been used in a diverse variety of ways, including system documentation.
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
XBRL	**E**xtensible **B**usiness **R**eporting **L**anguage
XML	E**x**tensible **M**arkup **L**anguage

Table 5 Glossary of terms.

References

[1] Free Software Foundation, What is free software? - GNU Project - Free Software Foundation, GNU Proj. - Free Softw. Found. (2015). http://www.gnu.org/philosophy/free-sw.html (accessed June 17, 2015).
[2] Open Source Initiative, The Open Source Definition, Open Source Initiat. (2015). http://opensource.org/osd (accessed June 17, 2015).
[3] B. Schießle, Free Software, Open Source, FOSS, FLOSS - same same but different, Free Softw. Found. Eur. FSFE. (2012). https://fsfe.org/freesoftware/basics/comparison.en.html (accessed June 5, 2015).
[4] M. Date, RepOSS: A Flexible OSS Assessment Repository, (2012). http://events.linuxfoundation.org/images/stories/pdf/lceu2012_date.pdf (accessed May 5, 2015).
[5] B. Doll, 10 Million Repositories, GitHub. (2013). https://github.com/blog/1724-10-million-repositories (accessed August 8, 2015).
[6] M. Sarrab, M. Elbasir, L. Elgamel, The technical, non-technical issues and the challenges of migration to free and open source software, Int J Comput Sci Issues IJCSI. 10 (2013) 3.
[7] D.A. Wheeler, Free-Libre / Open Source Software (FLOSS) is Commercial Software, David Wheel. (2011). http://www.dwheeler.com/essays/commercial-floss.html (accessed May 28, 2015).
[8] K.-J. Stol, M. Ali Babar, A Comparison Framework for Open Source Software Evaluation Methods, in: P. Ågerfalk, C. Boldyreff, J.M. González-Barahona, G.R. Madey, J. Noll (Eds.), Open Source Softw. New Horiz., Springer Berlin Heidelberg, Berlin, Heidelberg, 2010: pp. 389–394. http://link.springer.com/10.1007/978-3-642-13244-5_36 (accessed March 24, 2015).
[9] C.A. Fritz, B.D. Carter, A Classification And Summary Of Software Evaluation And Selection Methodologies, Mississippi State University, Mississippi State, MS, 1994. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.4470.
[10] OpenBRR, Business Readiness Rating for Open Source: A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software, OpenBRR, 2005. http://docencia.etsit.urjc.es/moodle/file.php/125/OpenBRR_Whitepaper.pdf (accessed April 13, 2015).
[11] A.I. Wasserman, M. Pal, C. Chan, The Business Readiness Rating: a Framework for Evaluating Open Source, in: Proc. Workshop Eval. Framew. Open Source Softw. EFOSS Second Int. Conf. Open Source Syst., Lake Como, Italy, 2006: pp. 1–5. http://web.archive.org/web/20070111113722/http://www.openbrr.org/como-workshop/papers/WassermanPalChan_EFOSS06.pdf (accessed April 15, 2015).
[12] A. Majchrowski, J.-C. Deprez, An Operational Approach for Selecting Open Source Components in a Software Development Project, in: R.V. O'Connor, N. Baddoo, K. Smolander, R. Messnarz (Eds.), Softw. Process Improv., Springer Berlin Heidelberg, Berlin, Heidelberg, 2008: pp. 176–188. http://link.springer.com/10.1007/978-3-540-85936-9_16 (accessed May 27, 2015).
[13] E. Petrinja, R. Nambakam, A. Sillitti, Introducing the OpenSource Maturity Model, in: IEEE, 2009: pp. 37–41. doi:10.1109/FLOSS.2009.5071358.

[14] J.-C. Deprez, S. Alexandre, Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS, in: A. Jedlitschka, O. Salo (Eds.), Prod.-Focus. Softw. Process Improv., Springer Berlin Heidelberg, Berlin, Heidelberg, 2008: pp. 189–203. http://link.springer.com/10.1007/978-3-540-69566-0_17 (accessed May 2, 2015).

[15] A.I. Wasserman, M. Pal, Evaluating Open Source Software, (2010). http://oss.sv.cmu.edu/readings/EvaluatingOSS_Wasserman.pdf (accessed May 31, 2015).

[16] A.S. Jadhav, R.M. Sonar, Evaluating and selecting software packages: A review, Inf. Softw. Technol. 51 (2009) 555–563. doi:10.1016/j.infsof.2008.09.003.

[17] F.E. Pani, D. Sanna, FAME, A Methodology for Assessing Software Maturity, in: Atti Della IV Conf. Ital. Sul Softw. Lib. 1112 Giugno 2010 Cagliari, Cagliari, 2010.

[18] F.E. Pani, G. Concas, D. Sanna, L. Carrogu, The FAME Approach: An Assessing Methodology, in: TELE-INFO10 Proc. 9th WSEAS Int. Conf. Telecommun. Inform., World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2010: pp. 67–73. http://dl.acm.org/citation.cfm?id=1844648.1844658.

[19] F.E. Pani, G. Concas, D. Sanna, L. Carrogu, The FAMEtool: an automated supporting tool for assessing methodology, World Sci. Eng. Acad. Soc. WSEAS. 7 (2010) 1078–1089.

[20] F.E. Pani, D. Sanna, M. Marchesi, G. Concas, Transferring FAME, a Methodology for Assessing Open Source Solutions, from University to SMEs, in: A. D'Atri, M. De Marco, A.M. Braccini, F. Cabiddu (Eds.), Manag. Interconnected World, Physica-Verlag HD, Heidelberg, 2010: pp. 495–502. http://www.springerlink.com/index/10.1007/978-3-7908-2404-9_57 (accessed May 2, 2015).

[21] M. Soto, M. Ciolkowski, The QualOSS open source assessment model measuring the performance of open source communities, in: ESEM 09 Proc. 2009 3rd Int. Symp. Empir. Softw. Eng. Meas., IEEE, 2009: pp. 498–501. doi:10.1109/ESEM.2009.5314237.

[22] M. Soto, M. Ciolkowski, The QualOSS Process Evaluation: Initial Experiences with Assessing Open Source Processes, in: R.V. O'Connor, N. Baddoo, J.C. Gallego, R.R. Muslera, K. Smolander, R. Messnarz (Eds.), Softw. Process Improv., Springer Berlin Heidelberg, 2009: pp. 105–116. http://link.springer.com/chapter/10.1007/978-3-642-04133-4_9 (accessed June 3, 2015).

[23] K. Haaland, A.-K. Groven, R. Glott, A. Tannenberg, Free/Libre Open Source Quality Models - a comparison between two approaches., in: Norsk Regnesentral/Norwegian Computing Center, Jena, 2010. http://www.nr.no/en/nrpublication?query=/file/5444/Haaland_-_Free_Libre_Open_Source_Quality_Models-_a_compariso.pdf (accessed April 15, 2015).

[24] O. Hauge, T. Osterlie, C.-F. Sorensen, M. Gerea, An empirical study on selection of Open Source Software - Preliminary results, in: Proc. 2009 ICSE Workshop Emerg. Trends Free. Source Softw. Res. Dev., IEEE, 2009: pp. 42–47. doi:10.1109/FLOSS.2009.5071359.

[25] C. Ayala, D.S. Cruzes, X. Franch, R. Conradi, Towards Improving OSS Products Selection – Matching Selectors and OSS Communities Perspectives, in: S.A. Hissam, B. Russo, M.G. de Mendonça Neto, F. Kon (Eds.), Open Source Syst. Grounding Res., Springer Berlin Heidelberg, Berlin, Heidelberg, 2011: pp. 244–258. http://link.springer.com/10.1007/978-3-642-24418-6_17 (accessed April 14, 2015).

[26] C. Chan, enugroho, T. Wasserman, Business Readiness Rating (BRR) | SourceForge.net, SourceForge.net. (2013). https://sourceforge.net/projects/openbrr/

(accessed April 21, 2015).
[27] P. Galli, OpenBRR Launches Closed Open-Source Group, eWeek. (2006). http://www.eweek.com/c/a/Linux-and-Open-Source/OpenBRR-Launches-Closed-OpenSource-Group (accessed April 13, 2015).
[28] OpenBRR, Welcome to Business Readiness Rating: A FrameWork for Evaluating OpenSource Software, OpenBRR. (n.d.). http://www.openbrr.org/ (accessed April 14, 2015).
[29] L. Arjona, What happened to OpenBRR (Business Readiness Rating for Open Source)?, Bright Side. (2012). https://larjona.wordpress.com/2012/01/06/what-happened-to-openbrr-business-readiness-rating-for-open-source/ (accessed April 13, 2015).
[30] OSSPAL, Welcome to OSSPAL, OSSPAL. (n.d.). http://osspal.org/content/welcome-osspal (accessed April 18, 2015).
[31] C. de Silva, 10 questions to ask when selecting open source products for your enterprise - TechRepublic, TechRepublic. (2009). http://www.techrepublic.com/blog/10-things/10-questions-to-ask-when-selecting-open-source-products-for-your-enterprise/ (accessed April 13, 2015).
[32] S. Phipps, 7 questions to ask any open source project, InfoWorld. (2015). http://www.infoworld.com/article/2872094/open-source-software/seven-questions-to-ask-any-open-source-project.html (accessed April 10, 2015).
[33] S. Padin, How I Evaluate Open-Source Software, 8th Light. (2014). https://blog.8thlight.com/sandro-padin/2014/01/03/how-i-evaluate-open-source-software.html (accessed June 1, 2015).
[34] R. Metcalfe, Top tips for selecting open source software, OSS Watch. (2013). http://oss-watch.ac.uk/resources/tips (accessed March 23, 2015).
[35] J. Limardo, DIY Evaluation Process, LIMSexpert. (2013). http://www.limsexpert.com/cgi-bin/bixchange/bixchange.cgi?pom=limsexpert3&iid=readMore;go=1363288315&title=DIY%20Evaluation%20Process (accessed February 7, 2015).
[36] D.A. Wheeler, How to Evaluate Open Source Software / Free Software (OSS/FS) Programs, David Wheel. (2011). http://www.dwheeler.com/oss_fs_eval.html (accessed March 19, 2015).
[37] Validation-Online, User Requirements Specification (URS)., Wwwvalidation-Onlinenet. (2015). http://www.validation-online.net/user-requirements-specification.html (accessed August 8, 2015).
[38] G. O'Keeffe, How to Create a Bullet-Proof User Requirement Specification (URS), Askaboutgmp. (2015). http://www.askaboutgmp.com/296-how-to-create-a-bullet-proof-urs (accessed August 8, 2015).
[39] ASTM International, Subcommittee E13.15 on Analytical Data, E1578 − 13 Standard Guide for Laboratory Informatics, (2013). http://www.astm.org/Standards/E1578.htm (accessed March 14, 2015).
[40] Autoscribe, User Requirements Checklist, Autoscribe. (n.d.). http://www.autoscribeinformatics.com/services/user-requirements (accessed April 10, 2015).
[41] Laboratory Informatics Institute, The Complete Guide to LIMS & Laboratory Informatics – 2015 Edition | LiMSbook.com, 2015th ed., Laboratory Informatics Institute, 2015.

http://www.limsbook.com/the-complete-guide-to-lims-laboratory-informatics-2015-edition/ (accessed April 10, 2015).
[42] FDA, Search for FDA Guidance Documents > Part 11, Electronic Records; Electronic Signatures — Scope and Application, Food Drug Adm. (2014). http://www.fda.gov/regulatoryinformation/guidances/ucm125067.htm (accessed June 10, 2015).
[43] S.H. Segalstad, International IT Regulations and Compliance: Quality Standards in the Pharmaceutical and Regulated Industries, First, John Wiley & Sons, Inc., Chichester, West Sussex, UK, 2008. http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470758821.html (accessed April 9, 2015).
[44] C. Harvey, Articles By Cynthia Harvey, Datamation. (2015). http://www.datamation.com/author/Cynthia-Harvey-6460.html (accessed April 12, 2015).
[45] C. Harvey, Open Source Software List: 2015 Ultimate List, Datamation. (2015). http://www.datamation.com/open-source/open-source-software-list-2015-ultimate-list-1.html (accessed April 6, 2015).
[46] SourceForge, SourceForge - Download, Develop and Publish Free Open Source Software, SourceForge.net. (2015). https://sourceforge.net/ (accessed June 14, 2015).
[47] GitHub, GitHub · Build software better, together., GitHub. (2015). https://github.com/ (accessed June 14, 2015).
[48] Wikipedia, Comparison of source code hosting facilities, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Comparison_of_source_code_hosting_facilities&oldid=682090863 (accessed September 28, 2015).
[49] AOSA Book, The Architecture of Open Source Applications, AOSA Book. (2015). http://aosabook.org/en/index.html (accessed September 28, 2015).
[50] E. Knorr, 5 key trends in open source, InfoWorld. (2015). http://www.infoworld.com/article/2986769/open-source-tools/5-key-trends-in-open-source.html (accessed September 28, 2015).
[51] Google, Google Scholar, Google Sch. (n.d.). https://scholar.google.com/ (accessed August 8, 2015).
[52] Microsoft, Microsoft Academic Search, Microsoft Acad. Search. (2015). http://academic.research.microsoft.com/ (accessed August 8, 2015).
[53] S.N. Wasike, Selection Process of Open Source Software Component, (2010). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.227.5951.
[54] Black Duck Software, Inc, Open Hub, the open source network, Black Duck Open HUB. (2015). https://www.openhub.net/ (accessed August 10, 2015).
[55] M. Neteler, M.H. Bowman, M. Landa, M. Metz, GRASS GIS: A multi-purpose open source GIS, Environ. Model. Softw. 31 (2012) 124–130. doi:10.1016/j.envsoft.2011.11.014.
[56] khanine, “Meaningful Use” Regulations of Medical Information in Health IT, Toad World. (2015). http://www.toadworld.com/platforms/oracle/b/weblog/archive/2015/05/07/quot-meaningful-use-quot-regulations-of-medical-information-in-health-it.aspx (accessed June 12, 2015).
[57] khanine, Open-Source Medical Record Systems of 2015, Toad World. (2015). http://www.toadworld.com/platforms/oracle/b/weblog/archive/2015/05/06/open-source-

medical-record-systems-of-2015.aspx (accessed May 28, 2015).
[58] Wikipedia, Health Insurance Portability and Accountability Act, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/wiki/Health_Insurance_Portability_and_Accountability_Act (accessed June 16, 2015).
[59] A. Davis, Enterprise Resource Planning Under Open Source Software, in: C. Ferran, R. Salim (Eds.), Enterp. Resour. Plan. Glob. Econ. Manag. Issues Chall., IGI Global, 2008: pp. 56–76. http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-59904-531-3 (accessed June 13, 2015).
[60] Mohamed Sarrab -, Osama M. Hussain Rehman -, Selection Criteria of Open Source Software: First Stage for Adoption, Int. J. Inf. Process. Manag. 4 (2013) 51–58. doi:10.4156/ijipm.vol4.issue4.6.
[61] A. Foote, The Myth of Free: The Hidden Costs of Open Source Software, Dalhous. J. Interdiscip. Manag. 6 (2010). doi:10.5931/djim.v6i1.31.
[62] Wikipedia, There ain't no such thing as a free lunch, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/wiki/There_ain%27t_no_such_thing_as_a_free_lunch (accessed June 13, 2015).
[63] I. Fleming, ISO9126 - Software Quality Characteristics, SQA Softw. Qual. Assur. (2014). http://www.sqa.net/iso9126.html (accessed June 18, 2015).
[64] Murphy, Murphy's Laws, Dave Taylors Educ. Guid. Couns. Serv. (2014). http://davetgc.com/Murphys_Law.html (accessed August 8, 2015).
[65] A. Abran, A. Khelifi, W. Suryn, A. Seffah, Usability Meanings and Interpretations in ISO Standards, Softw. Qual. Control. 11 (2003) 325–338. doi:10.1023/A:1025869312943.
[66] M.S. Andreasen, H.V. Nielsen, S.O. Schrøder, J. Stage, USABILITY IN OPEN SOURCE SOFTWARE DEVELOPMENT: OPINIONS AND PRACTICE, Inf. Technol. CONTROL. 35 (2006) 303–312.
[67] S. Saxena, S.K. Dubey, Impact of Software Design Aspects on Usability, Int. J. Comput. Appl. 61 (2013) 48–53.
[68] All About UX, User experience definitions, UX. (2015). http://www.allaboutux.org/ux-definitions (accessed August 8, 2015).
[69] J. Gube, What Is User Experience Design? Overview, Tools And Resources, Smashing Mag. (2010). http://www.smashingmagazine.com/2010/10/what-is-user-experience-design-overview-tools-and-resources/ (accessed August 8, 2015).
[70] A.K. Reitz, Choosing a License, Hitch. Guide Python. (2014). http://docs.python-guide.org/en/latest/writing/license/ (accessed May 13, 2015).
[71] J. Atwood, Pick a License, Any License, Coding Horror Program. Hum. Factors. (2007). http://blog.codinghorror.com/pick-a-license-any-license/ (accessed May 13, 2015).
[72] The Open Source Initiative, The Open Source Definition (Annotated), Open Source Initiat. (2015). http://opensource.org/docs/definition.php (accessed May 28, 2015).
[73] S. Hocevar, WTFPL – Do What the Fuck You Want to Public License, WTFPL – What Fuck You Want Public License. (2015). http://www.wtfpl.net/ (accessed June 16, 2015).
[74] Open Source Initiative, Open Source Licenses, Open Source Initiat. OSI. (2015). http://opensource.org/licenses (accessed May 13, 2015).
[75] TL;DRLegal, TLDRLegal - Software Licenses Explained in Plain English, TLDRLegal. (2015). https://tldrlegal.com/ (accessed June 16, 2015).
[76] D.A. Wheeler, Make Your Open Source Software GPL-Compatible. Or Else., David

Wheel. (2014). http://www.dwheeler.com/essays/gpl-compatible.html (accessed March 20, 2015).
[77] Docker, What is Docker?, Docker. (2015). https://www.docker.com/whatisdocker (accessed June 18, 2015).
[78] E. Knorr, When will we see Docker in production?, InfoWorld Ahead Curve. (2015). http://www.infoworld.com/article/2900333/cloud-computing/scenes-from-the-docker-revolution.html (accessed June 18, 2015).
[79] H. Emadeen, Bika : Free , Open source LIMS - Laboratory Information Management System for Windows , Linux and Mac OSX, Goomedic.com. (n.d.). http://www.goomedic.com/bika-free-open-source-lims-laboratory-information-management-system-for-windows-linux-and-mac-osx.html (accessed July 21, 2015).
[80] J. Limardo, Open-LIMS Analysis Results, LIMSexpert. (2011). http://www.limsexpert.com/cgi-bin/bixchange/bixchange.cgi?pom=limsexpert3&iid=readMore;go=1304895280&title=Open-LIMS%20Analysis%20Results (accessed February 7, 2015).
[81] J. Limardo, Scoring Open-LIMS, LIMSexpert. (2011). http://www.limsexpert.com/cgi-bin/bixchange/bixchange.cgi?pom=limsexpert3&iid=readMore;go=1305075855&title=Scoring%20Open-LIMS (accessed February 7, 2015).
[82] H. D. Glover, Academia Should Embrace Open Access Scholarly Publishing, Open J. Account. 02 (2013) 95–96. doi:10.4236/ojacct.2013.24012.
[83] Wikipedia, Copyleft, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Copyleft&oldid=674443822 (accessed August 9, 2015).
[84] Wikipedia, Alternative terms for free software, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Alternative_terms_for_free_software&oldid=672842398 (accessed August 3, 2015).
[85] FLOSShub, FLOSShub | free/libre and open source software research resources, FLOSShub. (2015). http://flosshub.org/ (accessed August 9, 2015).
[86] Wikipedia, Forge (software), Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Forge_(software)&oldid=674218812 (accessed August 9, 2015).
[87] Wikipedia, Free and open-source software, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Free_and_open-source_software&oldid=673395320 (accessed August 3, 2015).
[88] Wikipedia, Free Software Foundation, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Free_Software_Foundation&oldid=672578474 (accessed August 9, 2015).
[89] Wikipedia, Freeware, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Freeware&oldid=671740569 (accessed August 9, 2015).
[90] Wikipedia, Functional requirement, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Functional_requirement&oldid=669297630 (accessed August 9, 2015).
[91] Wikipedia, GNU General Public License, Wikipedia Free Encycl. (2015).

https://en.wikipedia.org/w/index.php?title=GNU_General_Public_License&oldid=675202761 (accessed August 9, 2015).

[92] Wikipedia, GNU Project, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=GNU_Project&oldid=670291710 (accessed August 9, 2015).

[93] Wikipedia, Open-source software, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Open-source_software&oldid=673820216 (accessed August 3, 2015).

[94] T. Fredrich, What is REST Anyway?, RestApiTutorial. (2015). http://www.restapitutorial.com/lessons/whatisrest.html (accessed September 27, 2015).

[95] R.T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, DOCTOR OF PHILOSOPHY in Information and Computer Science, UNIVERSITY OF CALIFORNIA, IRVINE, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[96] Wikipedia, Shareware, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Shareware&oldid=664844862 (accessed August 9, 2015).

[97] Wikipedia, Free software movement, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Free_software_movement&oldid=672888941 (accessed August 3, 2015).

[98] Wikipedia, Proprietary software, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Proprietary_software&oldid=673807377 (accessed August 9, 2015).

[99] Wikipedia, Uniform Resource Identifier, Wikipedia Free Encycl. (2015). https://en.wikipedia.org/w/index.php?title=Uniform_Resource_Identifier&oldid=682133619 (accessed September 27, 2015).

[100] D.A. Wheeler, The Free-Libre / Open Source Software (FLOSS) License Slide, David Wheel. (2007). http://www.dwheeler.com/essays/floss-license-slide.html (accessed May 28, 2015).

[101] GNU.org, Categories of free and nonfree software, GNU Oper. Syst. (2014). http://www.gnu.org/philosophy/categories.html (accessed August 3, 2015).