The Phoenix Process Framework

Process Framework for Open-Source Medical Devices

1 Introduction

1.1 *Purpose*

This paper helps answer the question "what do I need?" or "what is missing?" in either of two scenarios:

- Starting an open-source system development effort, or
- Joining an existing open-source project and getting lost.

This document describes a process framework – a library "of process components that are used to construct endeavor-specific processes," where process components include things such as work products, roles, and activities. [Firesmith] The framework is not a process. Each endeavor must decide its own process, and will do so based on the technical and communal requirements of the project.

This document describes a framework but without necessarily justifying it. The framework should make sense to engineers and engineering managers.

Use the framework to guide the design of a project's methodology, organization, and infrastructure. The breadth of the framework is intended to guide the discussions and with a little forethought help the project avoid frustrations.

1.2 *Context*

The framework was created for and named after the Phoenix Project[1], a study group of the Twin Cities Section of the Institute of Electrical and Electronics Engineers (IEEE). The Phoenix Project is developing an ambulatory blood pressure monitor for the Halberg Chronobiology Center, at the University of Minnesota Medical School. The Halberg Center needs a monitor that is suitable for long-term use, that can be used on a massive scale to obtain health metrics, and with which the Center can encourage the development of diagnostic, prevention and treatment techniques.

1.3 *Overview*

The document is organized as follows:

Section 2	The class of endeavors for which the Phoenix Framework is intended
Section 3	The framework's basic components – the framework metadata
Section 4	The framework matrix – a tabular summary
Sections 5–8	Details of the elements specified in the matrix
Section 9	Framework attributes supporting regulated products
Appendix	• A list of potential enhancements to the Phoenix Framework • A list of success factors for open-source projects

About this draft. This draft does not yet address regulatory issues, which are expected to demand an additional area (quality assurance) and an additional role (quality engineer).

The framework matrix, in Section 4, is adapted from [Rothfuss]. The matrix is interesting mainly because it brings the basic components – areas, roles, activities – into a single view. Areas are very useful for grouping activities, but the author is not convinced that correlating all three components is useful. The matrix might be deleted from the framework.

The author considers this an unfinished work. Any constructive criticism of this draft is welcome.

[1] Phoenix Project homepage: http://www.phoenix.tc-ieee.org

2 Endeavors Targeted by the Framework

The framework applies to projects with the following attributes:

Project Attribute	Ramifications
Open-source product	• Public access via the Internet • Virtual, globally disbursed team – network archetype [Sawyer]; software as a human, social activity • Self-organizing team – individuals take on roles by volunteering for specific tasks • Community-driven rather than plan-driven • Inclination to rapid, evolutionary product development • Predisposition to avoid documentation, though many open-source projects suffer from too little documentation
Non-developer clients	• The developers are neither the sponsors nor the users • Developers are not expected to be masters of the problem domain • Requirements elicitation, analysis and documentation must be explicit
Regulated product[2]	• Regulatory constraints generally translate into certain expectations for quality assurance and design activities: o Requirements engineering o Architectural design o Critical-path peer reviews o System testing against requirements o Closed-loop defect and issue tracking o Record-keeping for all of the above Medical devices are particularly subject to the FDA’s Quality System regulation, which includes Good Manufacturing Practice requirements. • Regulations may place specific requirements on the product. Sufficient quality assurance, verification, and validation must be in place to assure such requirements are satisfied. • See Section 9, “Framework Attributes Supporting Regulated Products”, for details

[2] The Phoenix Ambulatory Blood Pressure Monitor is a FDA Class II medical device.

3 Framework Components

Role	A role models a cohesive collection of responsibilities of one or more persons on an endeavor. Multiple persons may play a role, and a person may play multiple roles. A role is named for its responsibilities, not for the position within the endeavor of anyone playing the role.
Area	An area is a grouping of related processes. Areas are the coarsest entities in the framework. In corporate enterprises, areas usually correspond to disciplines and perhaps to organizations. In the framework, areas are more abstract. They overlap, sharing processes and tools. They enable the highlighting of connections between activities that the literature usually addresses in isolation.
Process	A process is a cohesive collection of endeavor-specific process components (stages, conventions, producers, work units, work products, etc.) that documents and constrains how the endeavor is executed. All work activities can be abstracted as processes. In open-source projects, they do not connote formal structures or written rules but rather signify recurring units of work. A specific project decides whether and how to formalize and document a given process.
Tool	A tool is a software application that is used by one or more persons playing one or more roles to create, modify, evaluate, or manage versions of work products, where a work product may be in the final project output or may be an intermediate artifact. Tools support processes. The framework specifies classes of tools, leaving specific selections to each project.

4 The Framework Matrix

The framework matrix is split into two tables.

- The first table shows which areas own which processes and which tools, where ownership indicates a decisive influence in a process's design or a tool's selection.
- The second table shows which roles participate in which areas.

Area	Processes	Tools	
Marketing	❖ Project promotion	❖ Project homepage ❖ Online fundraising	
Human Resource Management	❖ Staff & team development	❖ Knowledge sharing weblogs ❖ Skills matrix ❖ Opportunity matrix	
Systems Management	❖ Communication	❖ Asynchronous communication	• E-mail • Mailing lists with archives • Project web site • Discussion forums: ○ News server ○ Web forums • How-to guides • FAQs • Web content management system (Wiki – collaborative webpage editing)
		❖ Real-time communication	• Instant messaging • Web telephony • Meetings
	❖ Backup & mirroring ❖ Access control ❖ Logging	❖ Authentication systems ❖ Backup systems ❖ Site-mirroring systems	
Quality Assurance			
Verification & Validation	❖ Reviews ❖ Testing	❖ Testing frameworks	
Engineering	❖ Requirements engineering ❖ Prototyping ❖ Hazard analysis ❖ Version management ❖ Release management ❖ Deployment ❖ Documentation ❖ Bug triaging	❖ Revision control system ❖ Bug/issue tracker ❖ Build systems ❖ Design & code generation	
		❖ Quality assurance tools	• Unit testing frameworks • Code checkers • Remote code review tools
		❖ Collaborative development environment	
Project Management	❖ Schedule development ❖ Schedule control ❖ Responsibility assignment	❖ Issue tracker ❖ Calendar ❖ Product backlog	

Increasing involvement in Open-Source Project ➔ Roles

Increasing involvement in Open-Source Project ➔ Areas

→ Roles / ↓ Areas		Contributors										
	User	**Supporter**	**Webmaster**	**Administrator**	**Analyst**	**Maintainer**	**Tester**	**Developer**	**V&V Engineer**	**Quality System Engineer**	**Architect**	**Manager**
Marketing	x	x	x	x	x	x	x	x			x	x
Human Resource Management		x	x	x	x	x	x	x			x	x
Systems Management	x	x	x	x	x	x	x	x			x	x
Quality Assurance				x						x		
Verification & Validation				x			x		x			
Engineering		x		x	x	x		x			x	
Project Management						x		x			x	x

5 Roles

A role is a cohesive collection of responsibilities representing a single part that a person can play within the project. An individual may perform multiple roles, and multiple individuals may perform a single role. Every member of an open-source project fills various roles.

User	The User role is played when a person uses the product once it is placed into service in its production environment. A user is non-vocal but can take on the Supporter role by submitting bug reports, requesting new features, or participating in discussion forums. Users do not directly contribute to the content of the product but some persons playing the User role eventually become content workers.
Supporter	The Supporter role is played when by various means a person assists other project members, particularly users of the product.[3] A supporter may answer questions on mailing lists, on discussion groups, in live chats, or even in person (at a gathering, for instance). Supporters maintain frequently asked questions, file bug reports, and spread the word about a project. Analysts may rely on supporters as user representatives during requirements analysis. Supporters are the talent pool from which many eventual developers are selected.
Administrator	The Administrator role is played when a person establishes and maintains shared project resources. An administrator makes sure these resources are operational, is concerned with security issues, and is in charge of the tools supporting a project.
Webmaster	The Webmaster role is played when a person is responsible for the development and maintenance of some or all of the web pages at the project web site. The role is closely linked to that of administrator, but is more focused on design rather than operational issues. The Webmaster significantly affects the quality of a project's marketing and communications.
Analyst	The Analyst role is play when a person processes feature requests for a project. "The goal of an analyst's work is to produce useful recommendations for developers (specifications, priorities, feature lists) to allow developers to focus on developing. All users are analysts to some degree, and many users submit feature requests if given the opportunity. However, most of these feature requests are badly thought out, unrealistic, or do not fit well within the scope of the project. The most important function analysts can provide to a project is to provide reasons to reject proposed functionality and thus avoid feature creep. Analysts mainly work with bug tracking software and may use mailing list and content management systems to organize their work." [Rothfuss]
Maintainer	The Maintainer role is played when, in the presence of multiple parallel development tracks, a person oversees the changes to a stable track in order to prevent regressions. The role only becomes necessary when the project is mature enough to have multiple tracks. The major activities of the role are communication, reviews, bug tracking, and planning. A maintainer is passive until approached with an implemented change.
Tester	The Tester role is played when a person installs and uses the product to actively look for deficiencies, and reports the results in a useable format, usually with the help of bug tracking software.

[3] In a closed-source organization, the counterpart to the Supporter role could be a helpdesk, though who is helped by such a facility is too frequently a dubious point.

Developer	The Developer role is played when a person creates a product component, hopefully by some engineering discipline. "Many other roles solely exist to facilitate the work of developers, like architect, maintainer, analyst and administrator. Projects with many non-developer contributors may develop conflicts about the future direction. These conflicts are usually resolved by the actions of developers, who influence the project by their changes to the [product]. This 'rule of the developers' contrasts sharply with closed-source projects. Even the manager of an open-source project cannot rule by fiat and has to bow to the accumulated knowledge and judgment of developers." [Rothfuss] For a typical open-source project, a developer is concerned with software. A medical device, however, is a complex system composed of hardware, firmware, protocols, documentation, packaging, and facilities, as well as software. Such a system requires a variety of advanced disciplines. The framework, however, does not need the finer granularity, particularly when juxtaposing the Developer role with other roles.
Architect	The Architect role is played when a person is responsible for the overall technical vision of a project and has the ability and power to resolve technical debates. Large open-source projects are sometimes impossible to oversee for a single individual. In that case, the project is divided into smaller modules with well-defined interfaces, and shared responsibility. The architect focuses on the interaction of modules and keeps the big picture in mind without getting lost in details. Having at least two levels of developers within a project is common practice, allows projects to scale, and frees key contributors to focus on the long-term viability of the project. Modularity is one of the most important architectural features of successful open-source projects.
V&V engineer	The Verification and Validation (V&V) Engineer role is played when a person is responsible for the design and execution of the project's verification and validation process. The role is comparable to that of the Architect but over a different aspect of the project. For regulatory reasons, the V&V Engineer role must be played independently of (that is, staffed separately from) the Developer and the Project Manager roles.
Quality system engineer	The Quality System Engineer role is played when a person is responsible for the design and implementation of the project's Quality System Records, which include such artifacts as the design history file and device master records. The role is comparable to that of the Architect but over a different aspect of the project. For regulatory reasons, the Quality System Engineer role must be played independently of the Developer and the Project Manager roles.
Project manager	The Project Manager role is played when a person "coordinates the efforts of the various roles, provides a vision for the project (project managers are often founders and have been with a project since the beginning), and resolves conflicts. A humble approach works best, as volunteer participants have no obligation to bow to orders issued by the project manager (or anyone, for that matter). A project manager may also be involved with advocacy for a project, may try to secure funding for key infrastructure, and serves as a spokesperson for the project." [Rothfuss]

6 Areas

Involvement in an open-source project increases as one moves along the areas listed below, which means most projects give most of their attention to the latter areas and often ignore the former areas.

Marketing	Open-source projects depend on marketing to attract volunteers for their project. The more a project relies on skills other than computer programming and on users who are not programmers, the more varied must be the appeals to entice non-programmers to the project. Marketing is also necessary to attract sponsors who provide technical or financial resources, which cannot be ignored even if all labor is free. The Web is littered with open-source projects that failed or are failing for lack of marketing.
Human resource management	Human resource management is a relatively new area of open-source endeavors. It is about aligning the skills available to a project with the skills needed by the project. Human resource management overlaps with marketing by recruiting people with desired skill-sets. Human resource management then continues with the personal development, coaching, and training needed to retain those volunteers. By retaining its volunteers a project can avoid the anxious reshuffling of responsibilities when members leave the project, and can reduce the effort needed to recruit and retrain replacements.
Systems management	Missing resources are a key vulnerability for open-source projects. Systems management ensures that all necessary resources for a project are available. This means the web servers are up and the data they contain are intact. Systems management permeates all areas of open-source projects due to a heavy reliance on tools.
Engineering	Engineering is “the application of a systematic, disciplined, quantifiable approach to structures, machines, products, systems, or processes.” [IEEE 610] Most open-source projects disregard engineering best practice either because its formality detracts from the project’s “fun” or because technical prowess is cherished over careful design. The attributes that distinguish Phoenix-like projects from typical open-source projects (system complexity, client-driven requirements, a regulated product) compel a whole-hearted approach to engineering. Success depends on the usage of methodologies that are correct *both* from the general perspective of best practice *and* from the unique perspective of open source. The trick is to promote fun by offering an environment in which to learn the trade. Because a medical device is an inherently complex system, the Engineering area encompasses multiple disciplines: electrical, mechanical, biomedical, software, etc. Expect each to employ its own methodologies and tools, and expect each to find its own balance between the demands of best practice and those of open source. This may be particularly difficult for systems engineering. On the one hand, systems engineering is appropriate for medical devices because they are complex, safety-critical systems. On the other hand, systems engineering is intended for large system development, which can make much of the practice inappropriate for many open-source projects.
Project management	An open-source project relies heavily on leadership that keeps the project from floundering. Project management is mostly influence management; the leader suggests rather than directs. Due to the non-committal nature of most contributions, planning can only concern itself with short-term goals. The risks inherent in open-source can be addressed with a highly modular architecture and a hierarchical organization that allow extensive delegation of small, well-defined tasks; with publication of volunteer opportunities; and with status tracking that is more like nurturing than monitoring.

7 Processes

Processes do not connote formal structures or written rules but rather signify recurring units of work. A specific project decides whether and how to formalize and document a given process.

7.1 Marketing

7.1.1 Project Promotion

Involved Roles	All
Involved Tools	asynchronous communication, knowledge sharing web logs, online fundraising, project homepage

"The mechanics of popularity are poorly understood. There is no recipe for making a project popular, but getting the word out about a project is widely understood as a key ingredient for its success. Most projects do not promote themselves in any way, and wait for users (and future contributors) to find them instead. This lack of marketing can often be explained with the technical background of most contributors, and their negative experiences with corporate marketing… Project promotion is really a task for which all members of an open-source project are responsible, maybe the one task that they all share." [Rothfuss]

7.2 Human Resource Management

7.2.1 Staff and Team Development

Involved Roles	All contributors
Involved Tools	asynchronous communication, knowledge sharing web logs, opportunity matrix, project homepage, skills matrix

Staff and team development is all about attracting, developing, motivating, organizing, and retaining talented people. To this end, traditional organizations rely on initiatives like reengineering or process improvement, improved information sharing, clearly communicating the organization's mission, instituting employee involvement programs, establishing formal complaint resolution procedures, instituting gain-sharing or other incentive plans, workforce training, formal performance management and feedback.

The trick is to learn how to apply conventional human resource methods to the virtual environments of open-source projects.

An easy way of awarding status is to list the names of significant contributors in the project homepage's keywords meta statement and thus giving the contributor visibility on the Web.

7.3 Systems management

7.3.1 Communication

Involved Roles	All
Involved Tools	asynchronous communication, bug/issue tracker, knowledge sharing web logs, real-time communication, revision control system, web content management system

Information in its various forms (software, documents, URLs, etc.) is the most important resource for an open-source project. The amount of information grows daily. Distributing it efficiently and effectively poses some challenges:

- Who might be interested?
- What is the best way to notify interested persons?
- How should information be archived for later retrieval?
- How should relevant information be separated from noise?

Keep a broad perspective when addressing these questions. (For example, don't forget marketing and human resources.) Communication processes encompass all roles and affect all areas in a project.

Many different communication channels are available, each with its own properties, strengths, and weaknesses:

- **Internet** – Internet-based communications are the backbone of geographically disbursed virtual teams and stem intrinsically from tools:
  - one-way channels
    - web pages
    - web logs
  - multi-way channels
    - e-mail
      - person to person
      - public mailing lists with web-accessible archive
    - discussion forums
      - newsgroups
      - web content management systems
    - instant messaging (e.g., IRC)
  - next-generation channels
    - voice chat
    - internet telephony (VoIP, Voice over Internet Protocol)
    - video chat and video conferencing
    - shared white boards
    - remote access solutions

  A project must be sensitive to each member's ability to meet a tool's requirements for processing and connection speed.
- **Meetings** – The main strength of a meeting is its ability to resolve in hours difficult issues that would take weeks or months with mailing lists. A meeting also allows participants to bond; having personal relations helps tremendously with communications, as it reminds participants that there are humans at the other end of e-mails. Some small projects, particularly when initiated, are located in a single metropolitan area, making meetings easy. However, successful open-source projects usually need to expand across the globe. Meetings then become expensive and rare. Additionally, if a pattern of local meetings is continued, they can be divisive rather than unitive. Meetings can be effectively collocated with conferences, and address major issues for a project.
- **Documentation** – Participants and users learn about a project at different times. Having asynchronous communication is therefore crucial to transfer knowledge to new contributors. Documentation is the result of informal knowledge exchange, often collected first as FAQ (frequently asked questions) and later refined into documentation.
- **Source Code** – Due to the openness of open source, the source code itself is one of the most important channels of communication. Source code is the canonical document in technical matters, and reading source code can greatly expedite questions for seasoned programmers.

- **Revision Control Logs, Bug Trackers, Other Log Files** – Investigating the log messages of the version management system (e.g. CVS), bug reports of bug tracking system, or information archived by some other tools can be helpful. These log messages contain the complete trail of all changes made to a project, and are very informative if you are interested in the historical context of some functionality.
- **Traditional Communication** – Written letters are rarely used in open-source projects. Telephone conversations can be a very efficient means of getting to know a fellow participant, and of converging on solutions quickly. However, long-distance telephone charges prevent this means of communication from being used widely. Meanwhile, some voice chat systems exist that allow for voice communication over the Internet.

7.3.2 Backup and Mirroring

Involved Roles	Administrator, Developer, Maintainer, Tester
Involved Tools	backup systems, revision control system, site-mirroring systems

Computer systems are generally vulnerable to loss of data due to system failure, operator error, and unauthorized access. Open-source projects are particularly vulnerable to data loss to the extent that they rely on shared resources connected to the Internet. Central backups preserve the work of an open-source project.

Besides backups by administrators of the shared resources, there is also a second source of backups. As users and developers download copies of their project, they indirectly hedge against data loss at the central location. Tools like CVS (to retrieve copies of source code) and site mirroring (to retrieve copies of web sites) help with this task. Note, however, that any reliance on these informal backups becomes riskier as a project grows.

7.3.3 Access Control

Involved Roles	All contributors
Involved Tools	authentication systems

Most information in an open-source project is by its nature public and does not need special protection. Access control protects shared repositories for code and content by restricting write-access to administrators or the project manager. Some projects build a public key infrastructure to prevent attacks on their source code.

7.3.4 Logging

Involved Roles	All contributors
Involved Tools	facilities of the operating system, bug tracker, issue tracker, revision control system, web content management system

Logging is the practice of maintaining a time-sequential record of events, particularly of changes to the open-source product. With logging, one can determine which individual contributed which changes. "Logging exposes actions that are controversial or damaging for a project. As open-source projects are meritocracies, logging also provides raw data to spot the major contributors, and allows the project manager to reward these individuals with more status." [Rothfuss]

7.4 Quality Assurance

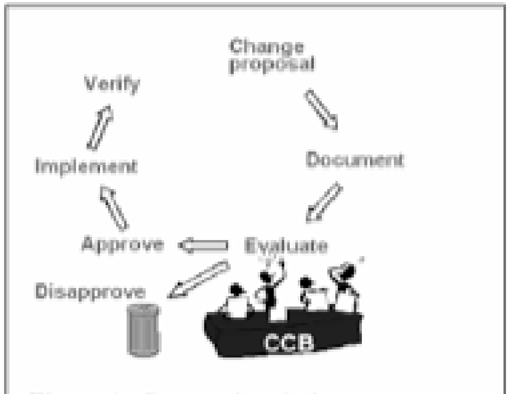
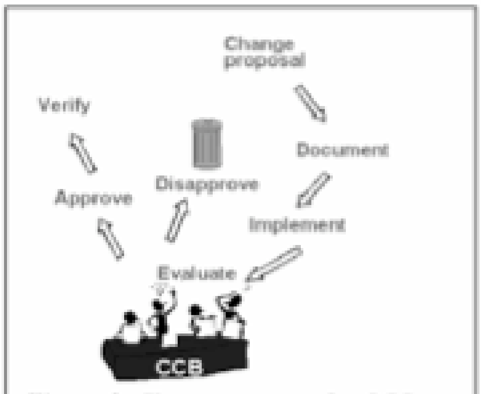
[to be determined]

7.5 Verification and Validation

[this section needs more work]

7.5.1 Reviews

Involved Roles	Analyst, Architect, Developer, V&V Engineer
Involved Tools	asynchronous communication, bug tracker, real time communication, remote code review tools, revision control system

 Change proposal Document Evaluate Approve Disapprove Implement Verify CCB Figure 1. Conventional change process  Change proposal Document Implement Evaluate Disapprove Approve Verify CCB Figure 2. Change process for OSS
Open-Source versus Closed-Source Change Cycles [Rothfuss, Asklund]

Because medical devices are safety critical systems, peer review must be in the critical path – code can be released for production only after it has been evaluated and verified.

7.5.2 Testing

Involved Roles	Developer, Supporter, Tester, V&V Engineer
Involved Tools	asynchronous communication, bug tracker, revision control system, unit testing frameworks

Providing easier and more efficient ways for participants to run tests will improve software quality tremendously. Many projects hesitate to simplify this process out of fears that less than able contributors might add many false alarms to their bug tracking systems. This risk is overrated. Combined with efforts to regularly prune bogus bug reports, widespread testing uncovers many issues that do not manifest themselves in the limited tests run by developers. A good testing policy needs to be tied in with sensible change management policies. Tools like CVS allow a wide range of processes to be implemented, from the "four eyes commits" rule (every line of code must be seen by at least two people – four eyes) up to highly complex review and approval processes.

Regulated project in particular must support acceptance as well as unit and testing.

7.6 Engineering

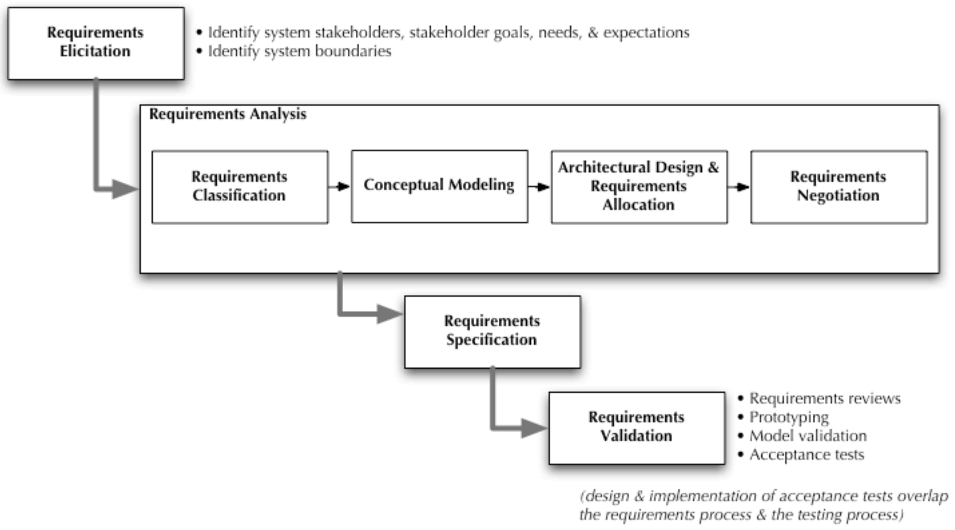
7.6.1 Requirements Engineering

Involved Roles	Analyst, Architect, Developer, Supporter, Tester
Involved Tools	asynchronous communication, bug tracker, web content management system

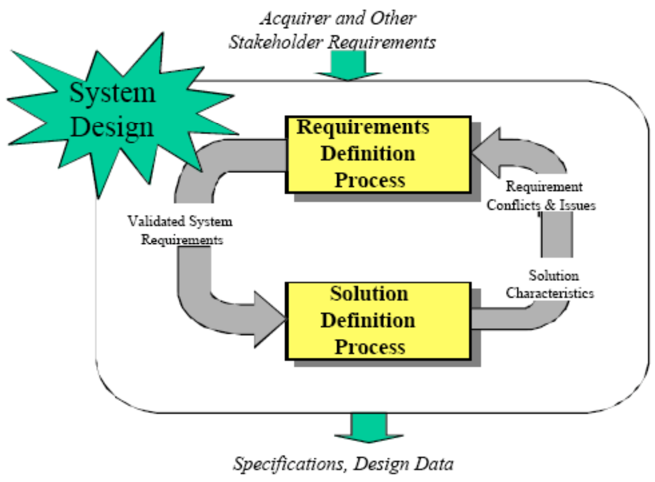
Requirements are:
- Expressions of stakeholder needs of a system to achieve particular goals
- Expressed in the vocabulary of the problem domain, rather than the system (solution) domain

Requirements engineering is about:
- Discovering stakeholder goals, needs, and expectations
- Adjusting stakeholder expectations
- Communicating the results to system implementers

The following two figures illustrate the classic requirements process, as described in [SWEBOK], and how the requirements process fits into the broader system design process.


Requirements Elicitation
• Identify system stakeholders, stakeholder goals, needs, & expectations
• Identify system boundaries
Requirements Analysis
Requirements Classification
Conceptual Modeling
Architectural Design & Requirements Allocation
Requirements Negotiation
Requirements Specification
Requirements Validation
• Requirements reviews
• Prototyping
• Model validation
• Acceptance tests
(design & implementation of acceptance tests overlap the requirements process & the testing process)

Requirements Process


Acquirer and Other Stakeholder Requirements
System Design
Requirements Definition Process
Requirement Conflicts & Issues
Validated System Requirements
Solution Characteristics
Solution Definition Process
Specifications, Design Data

System Design Process [INCOSE]

Open-source projects typically fit this classic process, but only informally. "Requirements engineering efforts are implied activities that routinely emerge as a by-product of community discourse about what their software should or should not do, as well as who will take responsibility for realizing such requirements." [Scacchi] The projects have strong requirements processes because it is so easy for users to make their voice heard. The downside of this manifests itself in the glut of requirements that many projects have, and a lack of priorities to respond to these requirements.

A typical, open-source requirements process:

1. **Requirements elicitation**. Users submit feature requests or report bugs. These requests are of very low quality on average.
2. **Requirements analysis**. "Since all feature requests are publicly visible, interested parties can voice their support for a feature. This allows gauging interest in a feature. Some projects, like the Mozilla project use voting to assign priorities to features. Analysts support this collaborative requirements gathering process by combining similar requests, rewording requests to make more sense, and starting the thought process about integrating a particular feature. Developers then use these refined requirements lists to assess feasibility of and assign priorities to features. If they feel so inclined, they may start on the implementation of some requests, or come up with proposals for farther-reaching changes. Local changes are usually the domain of individual developers, while global changes and major additions require the approval of the architect." [Rothfuss]
3. **Requirements specification**. "Requirements for open software are co-mingled with design, implementation, and testing descriptions and software artifacts, as well as with user manuals and usage artifacts (e.g., input data, program invocation scripts). Similarly, the requirements are spread across different kinds of electronic documents including Web pages, sites, hypertext links, source code directories, threaded e-mail transcripts, and more. In each community, requirements are described, asserted, or implied informally… *Software informalisms* are the preferred scheme for describing or representing open software requirements." [Scacchi]
4. **Requirements validation**. Requirements for open software are co-mingled with informal, beta testing.

[Scacchi] identifies eight types of software informalisms for open software system requirements:

1. **Community communications**: discussion forums and e-mail list servers
2. **Scenarios of usage as linked Web pages**: screenshots, guided tours, or navigational click-through sequences with supplementary narratives
3. **How-to Guides**: system tutorials and FAQs
4. **External publications**: books, and professional and academic articles
5. **Open software Web sites and source webs**:
   a) Community Web site for publishing and sharing open descriptions of the software
   b) Web-based source code directories, files, or compositions for download, build and/or installation
6. **Software bug reports and issue tracking**.

7. **Traditional software system documentation**: online system documentation (e.g., help pages) or documents intended to be printed (user or developer manuals)
8. **Software extension mechanisms and architectures**.

For regulated systems:
- It is not clear whether software informalisms are adequate for stakeholder objectives.
- It is not clear how software informalisms might be refined into formal requirements that are packaged into software requirements specifications

For client-driven systems, regulated or not, a minimal amount of documentation is needed and must be readily accessible by developers:
- Conceptual vision statement, in the vocabulary of the user
- Human-machine interface description, with navigation diagrams that relate wire-frame or prototype screen renderings
- Architectural description, in the vocabulary of the developer and at least at a high level of abstraction

Developing open software requirements is a community building process. However, for marketing, human resource, and regulatory reasons, finding and recognizing requirements should not demand familiarity and immersion within the community and its discussions.

7.6.2 Prototyping

Involved Roles	Analyst, Architect, Developer
Involved Tools	asynchronous communication, revision control system

Rapid prototyping is observed in, and is probably one of the strongest attributes of, most open-source projects. It is accepted way of validating requirements. Given an architecture that is sufficiently modular and is configurable with sufficiently small units, it is conceivable that developers can present the brunt of their software in prototypes.

7.6.3 Hazard Analysis

[to be detemined]

7.6.4 Version Management

Involved Roles	Administrator, Developer, Maintainer, Tester
Involved Tools	revision control system

The goals of version management are:
- Record Keeping – any previous version available at any moment
- Collaboration – easy synchronization between developers

Because open-source projects are so greatly distributed and loosely staffed, they are best served with version management that:
- Focuses on file revisions rather than change sets
- Follows the copy-modify-merge model rather than the lock-modify-unlock model

7.6.5 Release Management

Involved Roles	Maintainer
Involved Tools	backup systems, revision control system, site-mirroring systems

Release management is the process of providing some named (or otherwise uniquely identified) files to others for use. In a typical close-source project, the others may be a partner or a customer organization. In an open-source project, the others are the project users and contributors.

Managing a release means you know:
- What went into it
- Where it went
- Why it went there
- How to deal with it when bugs are reported

Competent release management includes:
- Integration of build and version management
- Frequent and periodic (e.g., nightly) builds – "release early, release often"
- Rational build identification (build numbers, branch naming, work status, release naming, etc.)
- Provision for custom builds by individual contributors
- Fully scripted building (such as with Make or Ant)
- Automatic build-numbering and tagging of source files
- Build scheduling
- Build-event notification via e-mail
- HTML interface to build system
- Database storage of build metadata (build number, build date, work status, etc.)

7.6.6 Deployment

Involved Roles	Administrator, Developer, Maintainer
Involved Tools	backup and mirroring systems, revision control system, web content management system

Deployment extends release management to encompass packaging, dependencies management, upgrades policy, and installation. Poor deployment is a large barrier to entry for open source software. Projects that make the extra effort to provide easy installation attract more users, and in turn, more developers.

Because of the very high reuse inherent in open source, managing dependencies between packages is a major issue in open-source projects.

7.6.7 Documentation

Involved Roles	Developer, Supporter
Involved Tools	asynchronous communication, web content management system

Documentation is as critical for an open-source system intended for non-programmers as for a close-source system. [Rothfuss] lists the following needs for documentation:
- **Internal and on-line**: Non-programmers insist that context-sensitive, on-line help must be provided with an application. Non-programmers want screen-shots in the on-line help. They do not care if it increases an application's file size. Non-programmers utilize on-line help as a quick reference, so indexes and search functions are important. Non-programmers will go through an on-line tutorial, if one is provided as part of the application. Non-programmers will look at a 'Tips and Tricks' dialog box, if one is provided. Non-programmers ignore the printed manuals bundled with off-the-shelf software. Non-programmers would never buy a book about an application. They say technical books are for programmers.
- **Simple**: Non-programmers do not want detailed explanations. They want simple answers. Non-programmers hate too much detail. Non-programmers prefer short, systematic instructions. Non-programmers prefer information that answers the question 'How do I do X?' (where X is a common use of the application). Nonprogrammers do not want to see information about how a feature was implemented.
- **Complete, correct and up-to-date**: Non-programmers assume that on-line help will be updated in each new version of an application. If part of the on-line help is obsolete or missing, nonprogrammers will not use any of it. If a non-programmer cannot find an answer in the on-line help, they will either call tech support or use another application.

Good documentation can also compensate for a lack of access to informal knowledge about the domain and the product by new entrants to a project.

7.6.8 Bug Triaging

Involved Roles	Analyst, Developer, Maintainer, Supporter, Tester
Involved Tools	bug tracker

Economic principles apply to the open source world just as they do to other areas. It a weakness of many open source projects to assume that resources are unlimited and can therefore be spent at will. Most software has far more defects than can be corrected with reasonable amounts of time and resources. Bug triaging tries to identify the most critical bugs the fixes to which provide the most value.

Determining the most important bugs can be done in a variety of ways. One approach assesses the likelihood and assigns a severity to each bug. Bug fixing efforts can then start from the likely and critical bugs and fixes as many bugs as possible within the allotted time. A checklist of questions to ask about each bug during triage is useful. Such questions may include:

- Can this bug report easily be reproduced?
- Does the bug report identify one specific problem?
- Can an engineer actually resolve the issue?
- Does the bug look like a duplicate report of a known issue?

7.7 Project management

7.7.1 Schedule Development

Involved Roles	Developer, Manager
Involved Tools	asynchronous communication, calendar, issue tracker, product backlog

Well-managed, open-source projects use scheduling to raise the effectiveness of their collaborators. Roadmaps specify target dates for feature freezes and releases.

"Contributors to open-source projects are primarily motivated by seeing their code being included in a project at the earliest possible time. Imposed schedules conflict with this goal, and are thus strictly opposed by most contributors. It is the duty of the project manager to impose deadlines in the interest of the project, while still considering contributor biases. Interestingly, most work is performed as soon as feature freezes are announced, as everyone scrambles to include his favorite feature before the freeze is in effect." [Rothfuss]

7.7.2 Schedule Control

Involved Roles	Developer, Manager
Involved Tools	asynchronous communication, issue tracker, real-time communication, web content management system

Schedule control concerns itself with the oversight of ongoing and upcoming work to spot issues as early as possible, and introduce corrective measures. In open-source projects, schedule control is complicated by the distributed nature of the contributors, their voluntary and therefore unreliable commitment and by communications issues. A better label for the activity might be *schedule influence*. Schedule control is commonly attempted with periodic newsletters and online meetings.

7.7.3 Responsibility Assignment

Involved Roles	Manager
Involved Tools	asynchronous communication, calendar, web content management system

Most open-source projects face severe staff shortages during their lifecycle. Even successful, popular projects find it very hard to allocate resources to specific tasks. Reasons for these difficulties include:

- Inability to force assignments on volunteers

- Lack of knowledge by project leaders about the skills of contributors
- Inadequate oversight of assigned tasks (see "schedule control")
- Inadequate oversight of unassigned tasks
- Conflict for existing contributors between contributing to the project themselves and mentoring new contributors
- Weak commitment of new volunteers
- Strong project cultures that discourage new contributors
- Poor or nonexistent documentation, and weak project infrastructures, that prevent self-guided "play" with and study about the product
- Increasing difficulty for an individual to gain status within a growing project
- Inadequate opportunities for new contributors to make a meaningful, early contributions
- Competition for talent between open-source projects (a frustrated contributor may find it easier to go elsewhere)

Improvements in allocation hold the promise of much better utilization of available resources. The situation is improved by:

- Architecting the product to be buildable in small packages.
- Assigning specific tasks rather than vague goals. Specific tasks make it easier for contributors to "do the right thing" especially given that many contributors feel that they do not know enough about a project to contribute more substantially.
- Documenting the architecture, and making the infrastructure reliable, in order to reduce the handholding needed by new contributors.

8 Tools

A warning from [Rothfuss]:

> Tools provide valuable services to OSP, but should not be used for their own sake. Many OSP exhibit an over-fondness for tools and try to solve issues with tools that should be resolved with processes instead. Tools support processes, but cannot replace them. One pitfall is to set up dozens of data entry tools to gather user comments, bugs, status reports, summaries, documentation, and then wonder why no one finds anything anymore. It is often better to use fewer tools more extensively than many tools only superficially. The ease to setup new tools masks the real problems that only crop up later: duplicate information, and data silos that gobble up but never again release information.

The framework specifies classes of tools, leaving specific selections to each project.

8.1 Marketing

Tool Class	Description	Illustrative Implementations
Project homepage	• The site is may be hosted by a corporate or a nonprofit sponsor. • The homepage may be also be the start page for the project's collaborative development environment (CDE, like SourceForge) or Web content management system (Web CMS, like Wiki) • As a marketing tool, the homepage should be well designed; see [Nielsen] for guidelines. For optimal marketing, the homepage might be better separated from the project's CDE and Web CMS. • Abundantly list keywords in the page-header to attract web crawlers (spiders) and thus increase the number of hits from web searches. Keywords should encompass the domain, major technologies, and major contributors. • An easy way of awarding status to significant contributors is to list their names with the keywords of the project homepage and thus increase the contributors' visibility on the Web	❖ www.phoenix.tc-ieee.org – the Phoenix Project, sponsored by the Twin Cities chapter of the IEEE ❖ samples of interesting designs: ➢ www.snipsnap.org – both a homepage and a Wiki ➢ www.osafoundation.org – Open Source Foundation ➢ metaverse.sourceforge.net – Open Source Metaverse Project ➢ www.apache.org – The Apache Software Foundation ➢ www.croquet.org – Croquet Project ➢ www.objectweb.org
Online fundraising		❖

8.2 Human Resource Management

Tool Class	Description	Illustrative Implementations
Knowledge sharing weblogs	A weblog is a web application that contains periodic posts on a common webpage. The posts are usually in reverse chronological order. A weblog is often run through a WCM system, and a WCM system could be used in place of weblog software.	❖ Snipsnap (Java; is also a Web CMS) ❖ WordPress (PHP and MySQL; free and open source, but hosting must still be resolved)
Skills matrix	Allows contributors to classify themselves on a wide range of skills, and optionally allows other participants to rate their skills. Currently, skill matrices are not widely deployed.	❖ The collaborative development environment provided by SourceForge.net provides for a skills inventory for each project
Opportunity matrix	A table of tasks currently needing volunteers. Ideally describes the task, the desired skill set, and whether is appropriate for a new project member.	❖

8.3 Systems Management

Tool Class	Subclasses / Description	Illustrative Implementations
Asynchronous communication	❖ e-mail ➢ servers ➢ clients	❖
	❖ mailing-list server with archive	❖
	❖ Web site ➢ server (host + software) ➢ homepage (content)	❖
	❖ online discussion group ➢ news servers ➢ Web forums	❖
	❖ Web content management system[4] ➢ WCM system, Web CMS ➢ collaborative webpage editing	❖ Wiki ➢ TWiki (Perl) ➢ TikiWiki (PHP) ➢ SubWiki (Python + Subversion SCM) ➢ MediaWiki (used by Wikipedia) ❖ Snipsnap (Java; also provides project blog, though not personal blogs) ❖ JotSpot
Real-time communication	When used for conferencing, these tools are useful alternatives to for telephone conference calls and face-to-face meetings.	❖
	Instant messaging • must find public, hopefully free, server and then matching clients • multiple conversation-threads are inevitable, making participation difficult (like trying to converse in a room full of loud people) • important to maintain agenda	❖ Yahoo! Messenger
	Internet telephony • relatively new technology • limited by bandwidth • service may limit the number of participants in a conference call	❖ Skype
Authentication systems	Public Key Infrastructure (PKI)	❖ SSH
Backup systems		❖
Site-mirroring systems		❖

[4] Most Web content management systems archive every revision of each managed webpage. A project's software engineering repository is separate from the project's Web CMS repository, and the two repositories could be managed with different revision control tools.

8.4 Software Engineering

Tool Class	Description	Illustrative Implementations
Revision control system		❖ CVS ➢ WinCVS ➢ MacCVS ➢ TortoiseCVS ➢ CVSWeb ➢ WebCVS ❖ Subversion ➢ RapidSVN ➢ TortoiseSVN ➢ ViewCVS
Bug Tracker	Ticketing system	❖ RequestTracker ❖ Bugzilla ❖ Scarab
Design & code generation		❖ ArgoUML, Dia (UML) ❖ Torque, Castor (SQL) ❖ JavaDoc, Doxygen (API documentation) ❖ JUnitDoclet (unit-test skeleton)
Quality assurance tools	Unit testing frameworks	❖ xUnit ➢ jUnit if product source language is Java ➢ cppUnit if C++ ➢ implementations for other languages
	Code checkers	❖ Lint (C language) ❖ Checkstyle (Java and Python) ❖ RATS (security)
	Remote code review tools	❖ Codestriker
Collaborative development environment	Provides standard toolset: • web server for project content • mailing lists with archives • issue tracker • revision control system	❖ SourceForge ❖ SourceCast ❖ Go to readyset.tigris.org for ready-to-use, software engineering, website templates

8.5 Project Management

Tool Class	Description	Illustrative Implementations
Issue Tracker	Issue tracking is very similar to bug tracking, and can be accomplished with the same tools given that these tools allow the ability to prioritize issues, attach them to a due date, and send reminders emails to the assignees.	❖ RequestTracker ❖ Bugzilla ❖ Scarab
Calendar	Open-source participants need to juggle their project involvement with their day jobs and many other activities. Often, a bit of scheduling would actually help to remind participants to finish some work that others in the group are waiting for. Various Internet-enabled calendaring tools can help with this task. vCard and vCalendar are two Internet standards that allow the ability to share appointments ands schedules across programs and platforms.	
Product backlog	"An evolving, prioritized queue of business and technical functionality that needs to be developed into a system." [Schwaber]	

9 Framework Attributes Supporting Regulated Products

To be determined. This section will address:

- Two areas:
  - Quality assurance
  - Verification and validation
- Two roles:
  - Quality systems engineer
  - Verification and validation engineer

10 Appendices

10.1 Appendix – Potential Enhancements or Research Opportunities

1. Integrate the framework with the following, at least to the extent that terminology is consistent:
   - Software Engineering Body of Knowledge [SWEBOK]
   - Project Management Body of Knowledge [PMBOK]
   - OPEN Process Framework [Firesmith]
2. Rework the framework into an architecture, with architectural viewpoints, as described in [IEEE 1471].

10.2 Appendix – Factors Contributing to Success of Open-Source Projects

[Rothfuss] lists the following factors, based on surveys:

- Active marketing/promotion of project – an evangelist is helpful
- Nice website
- Willingness of founders to grant commit-access to others
- Acceptance of ideas and viewpoints of others – with willingness to incorporate those ideas
- Architecture of the project prevents coordination problems – plug-in architecture
- Leadership – technically proficient
- Leadership – personable
- Usefulness of software – fills a niche or is better than current
- Competent technical core of developers
- A sense of community
- Stability
- Documentation and support
- Glamour – allows creativity and growth, excitement about the technology and the end product
- Ownership by developer community
- Communication among developers
- Communication to user community
- Open development cycle
- Low barrier to entry for developers – easy to get in
- Redundant developer roles
- Clear dispute resolution mechanism – by democracy, not management
- Layered organization – hierarchy promotes delegation
- User base size

11 References

[Asklund]	Ulf Asklund and Lars Bendix. "Configuration Management for Open Source Software." Position Paper, First Workshop on Open-Source Software Engineering, 23rd International Conference on Software Engineering, Toronto, Canada, May 15, 2001.
[Firesmith]	Donald Firesmith. "Firesmith OPEN Process Framework (OPF) Website." www.donald-firesmith.com, 2005.
[IEEE 610]	IEEE Std. 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.
[IEEE 1471]	IEEE Std. 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.
[INCOSE]	International Council on Systems Engineering. *Systems Engineering Handbook: A "How To" Guide For All Engineers*. Version 2.0, July 2000.
[Nielsen]	Jakob Nielsen. "Top Ten Guidelines for Homepage Usability." www.useit.com/alertbox/20020512.html, May 12, 2002.
[PMBOK]	Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*. 2000.
[Robbins]	Jason E. Robbins. "Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools." http://www.ics.uci.edu/~jrobbins/papers/robbins-msotb.pdf. *Making Sense of the Bazaar: Perspectives on Open Source and Free Software*, J. Feller, B. Fitzgerald, S. Hissam, & K. Lakhani, Eds. Publication forthcoming.
[Rothfuss]	Gregor J. Rothfuss. "A Framework for Open Source Projects." Master's Thesis in Computer Science, Department of Information Technology, University of Zurich, November 12, 2002, Supervisor, Prof. K. Bauknecht. *A key source for the Phoenix Process Framework. Rothfuss presents the framework components and framework matrix, and integrates the important areas of marketing, human resources, and systems management with product development and project management. Unfortunately, the quality of the thesis's rhetoric does not match the quality of the underlying research. However, to his credit, the author guides academically inclined readers who want a historical perspective to the first four chapters, while readers with little time on their hands are directed straight to Chapter 5.*
[Sawyer]	Steve Sawyer. "Software Development Teams." *Communications of the ACM 47, 12* (December 2004), 95-99. *Describes three social structure archetypes of software development teams. Open source projects exemplify the Network Archetype.*
[Scacchi]	Walt Scacchi. "Understanding the Requirements for Developing Open Source Software Systems." *IEE Proceedings – Software*, 149(1), 24-39, February 2002.
[Schwaber]	Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2002.
[SWEBOK]	IEEE Computer Society and ACM. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. www.SWEBOK.org, 2004.