Serenity: A Case Study in Developing Sustainable Information Systems

IT University of Göteborg
Software Engineering and Management
Göteborg, Sweden

Pontus Andersson
andpon@ituniv.se

David Birath
birath@ituniv.se

Patrik Willard
willard@ituniv.se

***Abstract*—The Internet has made a huge impact on the way we, as humans, communicate. During the last decade a series of new communication mediums have emerged and communication protocols have come and gone. This puts new requirements on the development process and architecture of the communication platforms, operated by communities, in order to pro-actively ensure support for future communication protocols.**

Even further, additional requirements are added when the software itself will be maintained by a community.

In this thesis we explore the world of Free/Libre and Open Source Software in a case study of the Serenity Information System. We present our suggestions of usable design-principles and our process in developing a sustainable information system.

I. Introduction

The proliferation of the Internet in the last two decades have given rise to several new communication protocols which supplement older ones, e.g. E-mail and Bulletin Board Systems. With this wide-spread availability and use of the Internet, the concept of *"virtual teams"* has emerged. Virtual teams can be defined as groups of collaborating individuals to whom geographical location and differences in time-zones are of little consequence, as there are communication mediums where time and synchronicity isn't an issue.

Without the boundaries of space and time there are some obvious benefits to virtual teams compared to "face-to-face" teams, but at the same time this necessitates a good communication infrastructure in order to function effectively.

Agarwal and Maruping use the media synchronicity theory to show how different communication mediums have differing sets of strengths and weaknesses[1]. This theory has come true on the Internet as well, due to the growing plethora of digital communication protocols. An optimized virtual team is thusly a team which has grasped the importance of using various communication protocols and created a framework supporting these.

This puts new requirements on the development process and architecture of modern web platforms. Fielding and Taylor state that *"Even if it were possible to build a software system that perfectly matches the requirements of its users, those requirements will change over time, just as society changes over time"*[2]. As the Internet will continue to evolve, new communication protocols are sure to be developed, and modern communication platforms should have the feature to adapt and include new protocols, incorporated in the core design.

For this thesis we approached the Free Software Foundation Europe (FSFE) as we knew, from a previous study[3], that the organization maintained a virtual community, and also that they were experiencing issues in communication which was beginning to diversely affect them.

The FSFE is an organization consisting of several virtual teams which produce a large amount of data using several different protocols. This in turn has resulted in an inconsistency in data storage, poor structure and an overload of information.

The current platform used by the FSFE was deemed too complex for the organization to extend, a new modern web platform needed to be developed. The spirit of those experiences was embodied by Henrik Sandklef (FSFE) stating: *"it is important that the software is structured and well written, modular since we need to maintain our installation and be able to add features later on."* In this thesis we describe the research and development that has gone into the creation of the Serenity Information System, a platform which could potentially evolve and become a sustainable replacement. In addition to this we also explore possible methods for developing sustainable software.

II. Literature Review

A. Sustainability

When discussing sustainability and IT, one often refers to the so called *"green IT"*. Going in to this thesis, however, our understanding of the term sustainability was that of a product's ability to present an evolving life-cycle in the face of changing requirements or technologies.

For the Free Software Foundation Europe, hosting several virtual teams, sustainable software add yet another aspect. The software in itself should help the community to stay active and attract new members.

Sustainability as we will approach it is therefore a mixture of properties ranging from communally revitalizing, to architecturally extendable. Based on the various definitions of the

term it can be difficult to find related research which directly fit our definition.

1) Architecture: From an architectural view, sustainability, as we define it, is not unheard of. An example is the case study of the Nightingale system conducted by Bass et al[4] who concluded that *"The system had to be maintainable, configurable and extensible to support new markets (e.g., managing doctors' offices), new customer requirements, changes in state laws and regulations, and the needs of the different regions and cultures."*

Several softwares handle this issue by offering a solid foundation which users then modify on a per installation basis. One such system is the Eclipse IDE[1] which has, since it was released under an open source license in 2001, adapted well to the introduction of new technologies and languages. This is due primarily to the architecture which allows Eclipse to be extended through plugins and thus handle new languages and development concepts, earning the reputation of being one of the best development tools on the market[5]. Although unproven, it stands to reason that this modularity gives Eclipse a huge competitive edge.

2) Free and Open Source Software: As the possibilities with, and popularity of, Eclipse increase, so does the community which supports it, making Eclipse a sustainable system, not only from a technical point of view but also from a communal view.

In 1999 Raymond wrote *"The Cathedral and the Bazaar"*[6] thereby defining the Bazaar model, which outlines ways for a community-driven development effort. A few years later, when it had gained real traction and come into interest, research related to the subject, such as community-based end-user involvement[7] began to spring up.

From a business point-of-view the decision to release Eclipse under an open source license goes hand-in-hand with what Holmström concludes: *"Traditionally, the conception of software development and maintenance is that of tedious and time-consuming processes. Thus, the opportunity to have parts of development outsourced to customers is indeed attempting[sic]"*[7].

More evident in organizations than companies, communities play an important role in free software and open source initiatives. Although it isn't unheard of for communities to spring up around companies (e.g. Blizzard Entertainment and the *MMORPG*[2] World of Warcraft, or SuSE and their community based variant OpenSuSE) the majority of communities are for the most part not customers, but volunteers contributing with their spare time.

B. Design Principles

Extensible platforms takes on a life of their own post-installation, as each individual installation will evolve alongside its own community and their specific requirements. This can easily be verified by observing the difference in features that organizations, which use the same technical solution, offer their members. Examples of such solutions would include *Drupal*[3] and *WordPress*[4].

In light of this, modularity and flexibility should be key features in the architecture, but important as well is to have a plan for how to handle such evolution, in order to not fragment the community (software users) by breaking compatibility between versions.

In the initial phase of development we identified five interesting principles that could serve us as developers by reducing the development time, and also make the product more maintainable:

- ORM *(Object-Relational Mapping)*
  is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a "virtual object database" which can be used from within the programming language[8]
- DRY *(Don't Repeat Yourself)*
  is a principle with clear benefits which ties well into maintainability, as there is less of a risk to introduce inconsistencies in the source code. The objective is to eliminate knowledge duplication, not by reuse, but by having only one source for each distinct piece of knowledge, and let all other components derive their knowledge from that one source[9]
- CBSE *(Component Based Software Engineering)*
  is a way to build systems by using functional or logical components. The components communicate through well-defined interfaces, and as long as these interfaces are honored, components may be replaced to provide additional or enhanced functionality without affecting the other components in the system[10]. This method increase reusability as you do not have to rewrite common functions which has already been developed in other systems. This method therefor shortens the development time and may reduce the overall cost normally associated with development. Weyuker highlights some potential risks which will be discussed further in section V
- 4GL *(Fourth Generation Programming Languages)*
  while not practices in themselves, can enhance and reinforce other practices. We also deemed it interesting to observe whether the use of a 4GL would provide any significant advantages, as compared to "the usual" web-programmings languages (e.g. PHP[5])
- Tracer Bullet development
  is a discipline, most commonly used in iterative development, that focus on writing small "proof-of-concept" features which can be refactored into a more functional state if the bullet "hits its mark". This practice can be extremely useful in the initial stages of development of a new component or feature

[1]http://www.eclipse.org/
[2]Massively multiplayer online role-playing game
[3]http://www.drupal.org/
[4]http://www.wordpress.org/
[5]http://www.php.net/

C. Communication

Sillencea and Baberb explores how the concept of *"Community Technology"*, when combined with other communication media, can be used to enhance the experience of the members in a community[11]. They conducted an experiment which incorporated text-messaging into the communication medium flora. The experiment resulted in increased activity within the community, which also increased the test group's interest in the soccer world cup of 2002, the event which the community centered around. The study mixed broadcast type mediums (e.g. web-sites) with what have traditionally been considered a point-to-point, individual and private medium (text-messages). One of the ways this was used bears a resemblance to how the US and Russia used to communicate with their missile submarines[12]. In practice a message is sent out through a medium which, with near total guarantee, will reach the recipient. This message cannot carry much information due to technical or physical constraints, so the purpose of this message is simply to alert the recipient that there is a real message awaiting transmission. It is then up to the recipient to get in contact with the sender.

It can be concluded that the principle behind this type of communication is applicable in several different fields, and probably several more, yet to have been discovered.

Though Sillencea and Baberb's conclusions only revolves around how traditional media could be improved if it was more integrated, it is likely that the same argument is true for modern communication forms as there exists a correlation between their research and the media synchronicity theory[1].

D. Information Overload

Information overload, though not a primary aspect of this thesis, has shown to be an important issue for the FSFE. It was thus a priority for us to research this field in order to, at the very least, provide a solution which does not increase information overload within the organization.

Information overload has been researched since 1970[13], [14] and even though several methods and techniques have been developed to mitigate the problem, the effects of the problem has increased. The research firm Basex[13] estimated the cost of unnecessary interruptions and related recovery time to a total of $650 billion in the U.S. alone.

Information overload affects not only businesses but also society as a whole, because with increasing overload comes stress, which not only leads to poor decision making but can also lead to mental breakdowns, with months of rehabilitation as a resulting factor[15], [16].

People become bogged down with information because:

- the sheer amount of information can be overwhelming
- the information is not always presented in an organized manner
- the information may not be relevant
- the information is in some cases given a false urgency

Speier and Price suggests that *"The use of aggregate data minimizes the effects of information overload on a decision maker"* but also warns that *"[aggregate data] may not provide a decision maker all the details he or she would like to have"*[17].

While their study focus on the temporal perspective of information (urgency or how fresh/stale the data is) which is an important component in both communication and decision making, the idea of aggregating information remains valid even without the time aspect, if the aggregation source can provide means to make the information clearer or more accessible.

Summarized data may reduce information overload, but may at the same time *"decrease overall decision quality as the information selected and processed is not as precise as the detailed information"*[17], which indicates that summarized information might not be the most advantageous approach.

These considerations proved to be important aspects to recognize and consider when deciding on the overall design and orientation of the system.

III. Research Approach

The aim of this paper was initially to study if centralized aggregation and data filtering could have a positive impact on information overload. Over the course of the study, however, the focus began to shift towards a more general study about how to develop sustainable community-driven web-applications.

As we had been involved in a previous study[3] regarding the Free Software Foundation Europe we felt a strong incentive to continue in the footsteps of that study, as we already possessed a great deal of domain-knowledge about the organization.

A. Subject Proposal Screening

Although we felt, as previously mentioned, a strong incentive to further study the FSFE the screening process was necessary in order to evaluate if there existed a research problem which could provide enough worth as a scientific and practical contribution, and which also affected the organization. Through the pre-study we had concluded that there was a gap in communication between the members, as the members tended to have a preferred communications protocol, rather than using the protocol best suited for any particular situation. In a perfect world the media synchronicity theory would reign supreme but that is almost never the case, and it would be presumptuous of us to try to change the members' opinions. The next best thing would be an attempt to develop a system which would bridge the gap.

A concern which arose was how to manage the risk of information overload as this system would have the potential to flood all the virtual teams of the foundation with all the information being sent, relevant or not. We then realized that this subject could be abstracted, to not focus solely on an application for the FSFE, but as a more general attempt at a practical solution. To solve the problem, our proposition was to aggregate the information at a central "hub" and categorize it. This would make it easy to search through the information,

filter it, and thus distribute it to members based on these categories.

B. Domain Knowledge Collection

With the subject of the thesis shaping up we set out to research the problem domain and visualize what had already been studied. The findings of this research are presented in greater detail in section II.

Knowing what our subject was, and what related research existed, we were almost ready to begin, there was just the question of what research methodology to use standing in our way. The iterative nature of the action research methodology, among other things, made it seem ideal for our purposes, as we would at the same time develop a new system. The four phases of action research (Plan, Act, Observe and Reflect) could be mapped to the iterative phases of development (Design, Implementation, (User) Testing and Feedback).

C. Design and Implementation

We decided that the most appropriate development process was to be an agile one. This was based on the fact that we did not have a clearly stated requirement specification from the FSFE and we felt the need to give the "customer" a central role in the development of the system.

1) Customer involvement: During the development, meetings were held on a weekly basis with representatives from the FSFE. During these meetings we assessed the overall project progress, demonstrated the current state of the system and discussed upcoming features. As this system will later be maintained by the community, it was important that we not only discussed functionality and usability but also covered the architecture from a maintenance point of view. We also gave their suggestions much consideration, drawing upon their previous experiences, to produce a better system.

In addition to these meetings, a live version based on a nightly build, which allowed end-users of the system to familiarize themselves with the user interface and the new concepts, was released.

2) Implementation: The development of Serenity followed, in most aspects, an iterative and incremental development process. The idea is to produce the system in minor iterations which is then released to the client. Usage of this method has decreased the resources spend in post-delivery maintenance significantly[10] and is often used in project, such as this, where a complete list of requirements are not known at the launch of the project.

The use of a incremental process served the project well as we had insight into the needs of the Free Software Foundation but there existed no formal requirements specification. We therefor decided to use the concept of tracer bullets[9] in conjunction with close "customer" involvement, in order to create proof-of-concept functionality which could then be demonstrated to the representatives. If they liked the feature, it was further evaluated and implemented. Features which turned out not to be in their interest was left out.

Serenity, as a system, is further described in section IV.

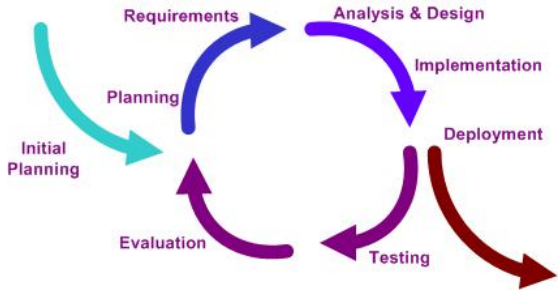

Fig. 1. Illustration of the iterative development process

D. Thesis Subject Revisited

Over the course of the study we began to ask ourselves questions more related to the design and development of systems maintained by, but also sustaining, a community of people. *What additional requirements are placed upon communal software? How do you develop software with communal qualities, such that it becomes a tool for sustaining the community?*

As Serenity had come to take shape we could focus fully on what qualities it needed, and already had, due to the feedback from the representatives from the FSFE, and how this could affect a community using it.

With the re-orientation of the subject we also came to realize that the research method we had selected did no longer serve its purpose, and in a meeting with a shadow consultant[6] it was suggested that the case study method might prove to be a better choice.

There are a lot of critics condemning the use of case studies, and even proponents of the method recognize that there are some potential difficulties which might not always be considered or appreciated by the users of the method[18]. There are however others, who feel that much of the criticism against case studies is unwarranted, or at least exaggerated[19], [20]. Nevertheless, the criticism has not gone unnoticed and has been applied to our best efforts in this study.

Due to the suggestion and after great thought, considering both drawbacks and advantages of the case study methodology it became evident that change was the only way forward.

We have during our study maintained a close communication with the FSFE representatives which has given us insight in their needs and hopes, and their reactions towards Serenity. From the pre-study[3] we had also a considerable amount of quantitative data in the form of answers from a questionnaire about the information infrastructure present during that study.

E. Evaluation and Reflections

As an ongoing process during this study, and the meetings held, we have devoted much thought to what the "customer" has given in terms if feedback and continuously weighed this into our efforts both to develop a better software, but to also

[6]a consultant who, at the request of a colleague, by means of a series of discussions, helps assess that colleague's diagnosis, tactics, or role in a specific assignment

analyze what was being said, and what the meaning behind it was. Sessions with the shadow consultant also gave rise to a third, neutral, perspective. We realized the importance of this action as Flyvbjerg concludes that there are critics claiming that case studies *"contains a bias toward verification, that is, a tendency to confirm the researcher's preconceived notions."*[19], and by enlisting a shadow consultant we could get an unbiased voice telling us if we had gone astray.

The quantitative data we had collected previously directed our actions as to what should be done, by giving us an understanding in how the members of the FSFE work, but also what they liked and disliked. Since the publication of our previous study, it has circulated through the FSFE and the general opinion about our conclusions are that it is a "disheartening read, as it is so true."

As there is no suitable way of measuring a user or client's "happiness" with a software, we observed the reactions of the representatives from the FSFE and conducted interviews to derive an interpretation about what their thoughts and feelings about the system were. This qualitative data was used to evaluate how well the system matched the "requirements" of the FSFE.

IV. The System

A. Problem Domain

From our previous study[3] concerning the Free Software Foundation Europe, it was concluded that the organization was having issues regarding communication. Information is distributed in an unstructured manner, which causes a general confusion within the community. Our study found the following points to be the primary factors for this:

- Various information is distributed through different communication mediums
- The website is unstructured, in terms of finding relevant information
- The lack of clear guidelines on how and where information should be distributed

As the study states, and which the FSFE agrees on, is that the organization needs to remove their current platform, since it was deemed too complex for the organization to extend and adapt to their specific needs. The FSFE want a new system to be developed, which is maintainable and sustainable, in order to "resurrect" the community, as well as to attract new members.

On the market today, there are a variety of information-scraping systems designed to increase the efficiency of communication by merging several different data sources into one, e.g., PlanetPlanet[7]. However, one shortcoming of the existing systems is that their primary focus is information gathering, thus offering limited or no functionality to differentiate information with respect to urgency or individual preference. Another common shortcoming prominent in these systems is that they are designed with a specific information distribution technology in mind, and are thus limited with respect to adaptability of, and interoperability with, other technologies. This presents a clear disadvantage from a sustainability and maintainability perspective.

[7]http://www.planetplanet.org/

B. Concept

To alleviate this problem, the authors propose a technical solution which would collect information from sources varying not only in location, but also in format and protocol. The information would then be organized, filtered, and possibly prioritized, before being presented to the user.

There are several ways of organizing information, the most common being to sort it based on existing metadata[21] such as publication date, topic and author. Neither of these sorting options provide the flexibility to allow users an intuitive way of selecting sets of information based on the information itself, i.e., the content.

The most promising solution to this seems to be adding more metadata, in the form of a keyword based summary. This solution too has its own set of problems as was concluded by Paulillo and Penumarthy[22].

The information passed through various protocols is already "earmarked" with various forms of metadata such as "recipient", "sender" etc. As most of this metadata is in a protocol-specific format, the system must be able to understand the various formats, and reformat information from one format into another.

C. Serenity Information System

As the goal of this study was to create a first version, we did not only design, but also implemented a system to test the proposition through empirical tests. This system, the Serenity Information System, is designed to extract data from several computer-mediated communication protocols. In order to achieve this, while still remaining flexible and "future-proof", all data are handled by custom built plugins each handling a specific format and or protocol. The benefits of this design is that all plugins are self-contained making debugging and development simpler.

When information is retrieved, the content is scanned for keywords from which metadata are derived and attached to the content. This technique is commonly known as *tagging*. Paolillo and Penumarthy describes tags as *"a form of metadata, or data which label other data for the purpose of organization and access"*[22]. This phase is handled automatically, but for convenience, this should also be possible to manage manually in the future, if the accuracy or relevance of the tags with respect to the tagged content is too low.

The decision to use tagging is based on the observed phenomenon and recent popularity in various prominent web-based applications such as StumbleUpon[8], WordPress[9] and YouTube[10]. It would seem that humans have a natural predisposition towards summarizing context into keywords, al-

[8]http://www.stumbleupon.com/
[9]http://www.wordpress.com/
[10]http://www.youtube.com/

though, each individual is likely to have differing definitions for each keyword.

There are a couple of drawbacks with tagging, *"neither users nor system can be sure in any specific instance if a particular tag refers to any specific type of information"* and *"because the tag vocabulary is not 'controlled' or standardized, the categorization produced is informal, and not guaranteed to be the same from one person to the next"*[22].

The second drawback is completely avoided in Serenity as only the system operators are able to add tags. This will likely inconvenience users who have their own definitions of what specific keywords mean, but this would be a temporary hurdle until the users have acquainted themselves with the "new" definitions. The benefits of this is however a homogeneous and universal definition, making all similar content related through the same tags.

Finally, the data can be individually distributed by matching the content to another set of tags chosen by the user from a list specified by the system operators. These tags are stored in a user's profile, which also, as a possible future feature, could be determined from a query string at the end of a URL requesting the service.

Having the ability to store settings in user profiles, creates a potential for the system to allow distribution of access-restricted information only to authorized members.

Naturally, since the system can gather information from different formats and protocols, Serenity also has the possibility to distribute information in various formats and protocols, e.g, through an RSS-feed, e-mail digest, or via the main web frontend.

In the distribution process, it is of great importance to help the user sort out information which is relevant for him or her. In order to achieve this, information entries which matches the tags specified in the user's profile gets prioritized, and also weighed in terms of number of "hits" in different tags (fig. 2).

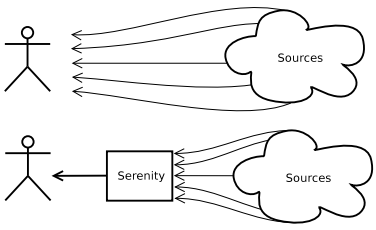

Fig. 2. Normal distribution vs. Serenity Information System

D. Serenity in Detail

As the following section tends to go into a deeper technical level, non-technical people can safely skip this.

The Serenity Information System consists of two major components: a backend and a frontend. The backend is meant to be run at all times collecting information from different sources. Consequently, it runs in the background, as a daemon[11], which means that it is independent of any direct interaction.

The backend works much like an operating system's kernel scheduler, allocating time and processing power for each plugin to execute their operations. The actual plugins does not necessarily need to be limited to a predefined set of tasks. The specification of what a plugin can do, and not do, is purposely "relaxed", in order to allow a flexible behaviour of the system, which will, first and foremost, make future requirements easy to adapt into the system, and secondly, it also lowers the complexity of the system.

The system is developed using Python[12] as programming language for both the frontend and backend. The frontend, i.e., the web frontend, which is the main interface for Serenity, is developed with Django[13], a web framework for Python. The reasons for the choice of Python and Django are many:

- Performance wise, compared to other similar platforms, like Ruby and Ruby on Rails, and PHP, Python and Django has an advantage
- Python is generally accepted, and there exist a multitude of resources to support the development process
- The people behind Django have a great developing philosophy (DRY)
- Both Python and Django have the characteristics to encourage one to develop using good practices and writing maintainable code

Django is an extremely versatile framework. It has an Object-Relational Mapper (ORM), which handles a relational database through data models. This enables the code to be very dynamic and intuitive. It also removes the need for any SQL to be written by hand, although, it is still possible for exceptional cases.

Since the Serenity frontend will always have a way to display information from various protocols and formats, it is of great convenience to let the frontend hold the one and only true representation of the data models. By doing this, and with the possibility to access Django's models outside of the project workspace, the backend receives the ability to utilize Django's ORM, which means that the *DRY* principle is followed. As a consequence, this creates an immediate dependency between the backend and the frontend, which from other design principles' point of views can be seen as a "defect". Although, for this case, with the advantages this choice gives to the system, we feel it is still motivated.

The currently existing plugins for the system, which handles a certain communication protocol, are RSS and IMAP. On the backend side those plugins' responsibility and purpose is to handle information gathering for their corresponding protocol and structure.

The engine which controls the plugins on the backend has fault tolerant characteristics regarding the execution of the plugins. This means that, when a plugin fails to operate, the

[11] a computer program which runs in the background
[12] http://www.python.org/
[13] http://www.djangoproject.com/

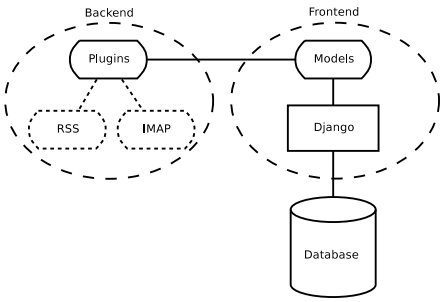

Fig. 3. Architechural overview of Serenity

engine will automatically disable the plugin from executing any further operations until the backend is reloaded. Consequently, an e-mail could be sent to notify the administrator, and the rest of the plugins can safely continue to operate normally without any disturbance.

The backend also provides automatic tagging functionality, which the information gathering plugins is intended to use in order to associate tags with the content retrieved. The tagging procedure works by scanning the provided content and matching the keywords that exist in the system, and then map the keywords to the associated tags.

In the frontend, plugins are, in Django terminology, "applications". Each plugin have their own model of an information entry that corresponds to its format's fields. There is also a way to map the plugin's entry model to a generic entry model. This model shares the most common fields amongst the different formats. By having a generic model, it is possible to mix all information entries without having the need to take knowing of which plugin they came from.

Fig. 4. Screenshot of Serenity's web frontend

V. Discussion

Communication using the Internet is increasing using more and more protocols. Organizations communicating primarily through the Internet run the risk of not reaching every member, as a consequence of everyone having their own favorite protocol. We approached the Free Software Foundation Europe as we in a previous study had learned that they where experiencing exactly these kind of problems. While easily solvable by setting up a well-defined communication strategy and more or less require members to follow it, in voluntary organizations such as the FSFE this may or may not be practical or applicable.

Through our early conversations with the FSFE we learned that one of their primary interests was of a system which could provide an improvement in the communication within the organization, but also, and maybe more importantly, spark a revitalization of the community by increasing the activity of their members.

Our approach to accommodating these aspects was to develop a platform which would utilize many of the present-day communication protocols, designing it to be easily adaptable to future changes in requirements. Further maintenance, i.e. post-deployment, would be left in the hands of the community as it would be impossible to foresee how the system will be utilized in the future.

A. Developing for a community

When developing for a community it can be important to let every member have his or her say in the project, which can quickly become very time-consuming.

We considered the use of tracer bullets to alleviate this problem, and found it extremely useful in the development of communal software, as the developer does not run the risk of spending countless hours developing a feature that won't make it into the system. By using the concept of tracer bullets, creating small proof-of-concepts, one can demonstrate these to the community, and then, if the feature is considered to benefit the system, be refactored and properly implemented.

Tracer bullets could be used in many aspects, not only code-wise but also in the appearance of paper mock-ups to demonstrate certain functionality. We also believe that this particular method should be used in any project where the requirements of the final product is unclear.

We believe that the use of an agile approach, which focused more on internal communication than extensive documentation, served this project well. However it is important to state that this methodology may not be suitable for all projects as there is a risk that this will decrease the quality and traceability in large scale projects.

During this study, while doing research on the communal aspects of development, we came to realize that there is at least one hurdle which may turn volunteers away from working on a project, namely the selection of tools and languages used in the development. The easiest way to lower this hurdle is to chose common and popular technologies which a majority of potential contributers are familiar with.

In the case of Serenity it would be prudent to have an ongoing discussion, with the members, about the definition and meaning of the keywords used by Serenity to classify and filter information, to ensure that the keywords stay relevant with respect to the content being organized.

Finally, for a sustainable system an aspect of high importance, is that the user interface is user-friendly, bordering

simplistic, so that even less technical members may take an active part in it, contributing in whatever means they can, such as feeding the Serenity with sources of information which add value to the community as a whole.

B. Design Principles

As we had not developed any similar software before, with requirements in sustainability or this level of flexibility, we recognized that it would be a good idea to look over our "tools" in order to have the best tools for the job.

The use of the web-framework Django, which has built-in support for object-relational mapping, has turned out to be a very successful choice. Django has proven to be a solid framework which in our case shortened the development time. Another benefit, as the DRY principle permeates through the entirety of the framework, is that the risk of duplication errors is dramatically reduced.

From a developer's perspective, Django is an extremely easy framework to work with. This is important since the code will be maintained by a community whoms knowledge in software engineering is unknown. Most of the common tasks, which usually needs to be handled manually, are done automatically, such as database queries, user authentication and the creation of an administrative interface. In fact, it was not until we reached the point of combining several different multi-relational entities that we actually had to write custom SQL queries ourselves. It is however unclear whether this is a hard limitation of Django, or of our understanding and expertise with the framework that forced us to do so.

One of the truly outstanding features of Django, is the thoroughly ingrained use of the DRY principle. Having "models" of the data structures which are to be used in the project, that Django then on its own converts to appropriate entities in the database is invaluable.

In total, 480 man hours was spent on the development of the system. What makes this extraordinary is that none of us had more than a sliver of basic knowledge in Python, and had done no development in Django at all, and Serenity, after roughly one month of coding, is a functional system.

The development time was kept short much due to Django and the thoroughly ingrained use of the DRY principle. Object-Relation Mapping is another incredibly powerful feature which simplified the interactions with the database to a degree where we seldom thought about it.

The general availability of already implemented functionality, and the ability to import any standard Python library goes to show just how well thought through Django is, and how well the DRY principle can work, when done properly.

We are particularly proud of the architecture of Serenity as we feel that is has the basic concepts for not only creating an active community around it but also the ability to overcome major changes in communication protocols and concepts through plugins. Henrik Sandklef states that the software meets the expectations of the FSFE and that Serenity has the potential to replace their current platform. The migration to Serenity will be discussed on the 29th of May 2008 in Brussels.

C. The Availability of Source Code

It is a common practice among modular systems to make use of, or offer the ability incorporate, third-party software as components in order to add needed functionality to the product. There are however problems to take into consideration before making the decision to use such a component. Weyuker observes that *"The first and most obvious problem is the almost certain lack of source code, precluding any modifications for either debugging or extensions"*[23]. This lack of source code make debugging cumbersome at best, and nearly impossible at worst, due to the lack of insight in the inner workings of the component. This has to be included in the risk-assessment for every project wishing to make use of such components.

As we discovered, a precondition to modifying a reused component, in the purpose of improving it, for use in community driven, sustainable, systems is access and right to carry out modifications on the source code. This in order to fit the component to the needs of the community. We did indeed find components which Serenity is based on, that we needed to modification in order to fulfill our requirements. Without the availability of the source code, this would not be possible and we would have to spend crucial time in developing our own components instead. We therefor strongly believe that the community must be in control of the software as their requirements change over time and the system is forced to evolve or be abandoned. This does not necessary mean that the code must be released under a free license but at least that the source is owned by the community and available for anyone, within the community, who wish to contribute.

D. Future Research and Applications

It would be most interesting to see how the tagging system could evolve using artificial intelligence to automatically add and adjust the tags for the content. Another study could be conducted about the effects of Serenity with respect to information overload. As Serenity will be released under a Free Software license, we welcome any further studies, and use, of Serenity.

We envision that Serenity could operate and expand into areas where this type of application might not dominate the market, such as company intranets.

For example, Serenity could be used as a perfect tool for stock exchange and business analysis. With the appropriate sources, such as from business institutes and stock exchanges, a user could be notified about changes in the market which are relevant to him, in other words, changes which match the tags he has subscribed to. Combined with the techniques used in the world cup study[11] he could also be alerted about severe swings where immediate actions are required, thus reducing the risk of loosing major investments. This presupposes that someone will write an appropriate plugin to do the job.

Another field could be news-serving or tracking several software development projects.

VI. Conclusion

In this study we set out to do a case study on how to develop a system which is maintainable and sustainable in a community. By reflecting on the process, and the system being designed we intended to draw conclusions of the important aspects which exists when developing such a system.

Our research is based on practical experiments including an iterative design process combined with continuous meetings with a representative from the Free Software Foundation Europe.

To avoid the possibility of using our reflections to prove our thesis we made use of a shadow consultant who ensured that we stayed on course throughout the study. Based on these sessions we discovered several existing design concepts which, when combined created a valuable method for developing web based information systems.

From an architectural point of view we have combined several known concepts and rapidly created a foundation which the community can easily build upon and extend as they see fit.

Django has proven to be an extremely useful and valuable framework for developing, maintaining web-based software, and incredibly easy to pick up without any prior knowledge of the framework. We feel that Django is an exceptional alternative for creating software destined to be maintained by a community. Paired with the tracer bullet technique, the two create a sum greater than its parts.

We believe that Django should be seen as a "role-model" for other frameworks and libraries, and thus be developed in the same "spirit". Much of the reason for Django's superiority lies in its use of powerful technical solutions and design principles such as ORM and DRY. We urge all who would consider themselves developers to study these in depth, and to try them out alongside other development techniques such as the tracer bullet.

We thus contribute by outlining development concepts and architectural design which has proven useful in the development of a sustainable community-driven system.

Acknowledgment

We would like to thank our supervisor, and primary contact with the Free Software Foundation Europe, Henrik Sandklef for all the help and support, and Carl Magnus Olsson at the IT-University of Göteborg for the feedback and insight during the writing of this thesis. We would also like to thank the Django development team for their exceptional work.

References

[1] Likoebe M. Maruping and Ritu Agarwal. Managing team interpersonal processes through technology: A task–technology fit perspective. *Journal of Applied Psychology*, 2004.

[2] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. 2002.

[3] Adriana Aires Rastén, Pontus Andersson, David Birath, Shahzeb Iqbal, Shane Kakau, Mattias Lingdell, and Patrik Willard. Suggestions for improving the Free Software Foundation Europe. 2007.

[4] Len Bass, Paul Clemens, and Rick Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley, November 2007.

[5] Peter Larsson. Eclipse vinner bland utvecklingsverktygen. http://www.idg.se/2.1085/1.163259, May 2008.

[6] Eric S. Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, 1999.

[7] Helena Holmström. *Community-based customer involvement for improving packaged software development*. Gothenburg Studies in Informatics, Göteborg University, November 2004.

[8] Wikipedia: Object-relational mapping. http://en.wikipedia.org/wiki/Object-relational_mapping, May 2008.

[9] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999.

[10] Ian Sommerville. *Software Engineering, Seventh Edition*. Addison-Wesley, Pearson, 2004.

[11] E. Sillencea and C. Baberb. Integrated digital communities: combining web-based interaction with text messaging to develop a system for encouraging group communication and competition. 2003.

[12] Wikipedia: Communication with submarines. http://en.wikipedia.org/wiki/Communication_with_submarines #Extremely_low_frequency, May 2008.

[13] Wikipedia: Information overload. http://en.wikipedia.org/wiki/Information_overload#Psychological_Effects, March 2008.

[14] Alvin Toffler. *Future Shock*. Random House, 1970.

[15] Francis Heylighen. Change and information overload: negative effects. http://pespmc1.vub.ac.be/CHINNEG.html, March 2008.

[16] Infogineering. Understanding information overload. http://www.infogineering.net/articles/understanding-information-overload.htm, March 2008.

[17] Cheri Speier and Michael F. Price. Using aggregated data under time pressure: A mechanism for coping with information overload. 1998.

[18] G Walsham. Interpretive case studies in is research: nature and method. 1995.

[19] Bent Flyvbjerg. Five misunderstandings about case-study research. *Qualitative Inquiry, Volume 12 Number 2*, April 2006.

[20] Robert K. Yin. *Case Study Research: Design and Methods, Third Edition, Applied Social Research Methods Series*. SAGE, 2002.

[21] Wikipedia: Metadata. http://en.wikipedia.org/wiki/Metadata, May 2008.

[22] John C. Paolillo and Shashikant Penumarthy. The social structure of tagging internet video on del.icio.us. 2007.

[23] Elaine J. Weyuker. Testing component-based software: A cautionary tale. 1998.