Free Open-Source Extensible Service Desk (FreeDESK)

David Cutting

A Dissertation submitted to

the School of Computing Sciences of the University of East Anglia

in partial fulfilment of the requirements for the degree of Master of Science

SUPERVISOR(S), MARKERS/CHECKER AND ORGANISER

The undersigned hereby certify that the markers have independently marked the dissertation entitled "**Free Open-Source Extensible Service Desk (FreeDESK)**" by **David Cutting**, and the external examiner has checked the marking, in accordance with the marking criteria and the requirements for the degree of **Master of Science**.

Supervisor:	XX-REDACTED-XX
Second Marker:	XX-REDACTED-XX
External Examiner:	
Moderator:	XX-REDACTED-XX

DISSERTATION INFORMATION AND STATEMENT

Dissertation Submission Date: **August, 2012**

Student:	**David Cutting**
Title:	**Free Open-Source Extensible Service Desk (FreeDESK)**
School:	**Computing Sciences**
Course:	**Advanced Computing Science**
Degree:	**M.Sc.**
Year:	**2012**
Organiser:	**XX-REDACTED-XX**

STATEMENT:

Unless otherwise noted or referenced in the text, the work described in this dissertation is, to the best of my knowledge and belief, my own work. It has not been submitted, either in whole or in part for any degree at this or any other academic or professional institution.

Permission is herewith granted to The University of East Anglia to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Student

Abstract

Service Desk systems, which are also known as Helpdesk or issue tracking systems, provide the ability to track “requests for service”. Such systems are in common use in many different organisations tracking requests raised by external customers, internally or automatically. For such a system to be usable in many different domains it must provide for a high level of customisation allowing it to be tailored to the specific requirements of each environment. Current solutions that exhibit this flexibility are commercial products whilst those that are freely available being either inflexible or highly domain-specific. The purpose of this project is to define, design, and implement an extensible service desk system which could be customised or extended as required to make it applicable cross-domain and release this as free open-source software.

A literature review was conducted covering aspects of service desks in general as well as techniques and methods used to build extensible systems and the common choices of implementation language. From an analysis of this review a set of high-level requirements and an overall system design were defined. An identified aspect of web-based multi user systems, ensuring updates to interface displays, was examined in detail. This included an experiment on the feasibility of using an alternative technique than regular polling for updates.

From the high-level design a detailed set of designs for the software were created and then implemented in an iterative and incremental manner. A website and documentation were built to support dissemination of the software for which a specific set of release pipeline tools were implemented enabling packaging into “release versions” for general use including an installer. Ongoing functional testing through a created unit test framework was performed on new features as they were added and on existing features as a form of integration testing.

An evaluation of the system in the form of an online questionnaire was completed and the results analysed. Feedback on the system was positive with a high proportion of respondents indicating they were satisfied with the system and would recommend it to others. A comparison of delivered features against the requirements analysis identified that all requirements of the system both for standard service desk function and relating to extensibility were met.

Acknowledgements

XX-REDACTED-XX

Contents

List of Tables

List of Figures

1 Introduction

1.1 What is a Service Desk/Help Desk?

Many organisations do, or could, make use of a computerised system to track “requests for service”. These are any requests raised by someone for action, for example incidents in an IT Helpdesk, queries for an HR department, or product fault reports for a retail chain. Such systems are known by different names including service desks, helpdesks, request trackers, or issue trackers.

In simple terms these systems are designed to log and manage requests raised internally or by an external customer, and for someone to do something to resolve the issue at hand either by performing an action or in many cases just providing the requested information. They are often also used to monitor and report on actions taken.

There are a wide range of systems available to support a Service Desk but generally these are either fully-featured and commercial (such as Hornbill Supportworks and Sunrise Sostenuto), freely available but closed source (including Sysaid and Hesk), or offering limited and/or domain-specific functionality (for example osTicket).

For a Service Desk system to be applicable cross-domain and in numerous possible types of deployment it must offer a form of customisation or extensibility, allowing the end-user to customise data, input format, and business logic to meet their intended purpose.

Such extensibility comes in numerous forms including a well defined applications programming interface (API), an architecture which allows for the possibility of future expansion (such as a plugin architecture), and/or by being an open-source free system allowing anyone to make any modifications they desire to the core system.

1.2 Problem Statement

Currently there is no fully-extensible free open-source Service Desk software solution capable of the customisation required for deployment in numerous different domains.

This leaves a choice of limited or highly domain-specific software available for free or expensive commercial software with closed source and ongoing licensing costs.

1.3 Aims and Objectives

This dissertation aims to define, design, and develop an extensible Service Desk software solution to be released as free open-source software (FOSS) under a suitable licence.

The specific objectives of the project are as follows:

1. Define and develop a suitable architecture allowing for easy extensibility through highly configurable components and plug-ins
2. Develop software to facilitate a Service Desk using this extensible architecture
3. Release developed software and associated documentation within the open-source community
4. Deliver a mature enough software system and documentation to allow third-parties to extend the base system (to replace certain core components with their own and to add entirely new functionality)

1.4 Expected Outcomes

A fully-featured[1] system made available for free use and distribution including full documentation.

[1]as identified in section 2.2 with additional functionality for extensibility discussed in section 2.3

2 Literature Review

The literature review for this system will cover a number of different areas including general requirements of a service desk system (section 2.2), available development environments (section 2.4), extensible software architectures (section 2.3), and PHP development (section 2.5).

2.1 Literature Search Methodology

In order to complete the literature review a systematic search was performed to find and consider different publications and other forms of information relevant to the project.

The search initially focused on three areas, the service desk domain in general, development environments and extensibility. Once the language for implementation had been chosen the PHP Hypertext Preprocessor (PHP) was added to the categories for search and review.

Searches were performed using a mixture of DBLP, Google Scholar and Google Web Search with a variety of identified key phrases. Further specific references were then taken from located relevant material and also included in the review.

2.2 Service Desk Domain Analysis

For a service desk to be useful it must provide features some of which are essential and some desirable. From Sinnett and Barr (2004a), Keller and Midboe (2010), Sinnett and Barr (2004b), Conlon (2007), and Black and Larsson (2004) the following list of these features can be defined:

- Essential
    1. Request tracking (creation, update, and recall) with unique reference
    2. Prioritisation (ability to view and sort requests in terms of their relative priorities)
    3. Clear update mechanism to update the original requester
    4. Performance monitoring (of the service desk performance)
- Desirable

1. Knowledgebase (addition of knowledge and search capabilities)
2. Method to avoid duplication of existing requests
3. Integration options with external systems (for example for authentication)
4. Automated escalation and/or closure of requests
5. Email integration
6. Web-based system (allowing use without specific software installation)
7. Customer portal (allowing self-service viewing, creation, and update of requests)
8. Customisation options
9. Selective issue titling (ability for requests when raised to be put into selective categories for analysis) through request profiles or similar
10. Grouping of requests and/or ability to assign requests to larger open issues

Implementing a service desk can require careful planning and the balancing of many considerations both of the initial functionality and any future stages. For example in an IT context, one question could be if features such as asset or change management be included at the outset or integrated later (Keller and Midboe, 2010).

For a service desk to be effective users must know about it, including self-service, and be able to access it and most importantly staff must be trained to use the system in an efficient and appropriate manner (Dooling and Bertrand, 2007).

2.3 Extensible Software Architectures

It is normal for software to evolve over time to meet new requirements, improve performance or to fix bugs. The traditional method for changes is to update the core system and then redeploy. This solution that is not ideal if the software is widely distributed or when small changes require a complete system rebuild (Chatley et al., 2004b). The term “software architecture” can be used variously to describe high-level structures, a design methodology, or specific development languages. In this analysis the term “software architecture” refers to the overall structure and high-level form of a system.

In the modern web-based context such pressures are increased as web-based application servers are becoming increasingly complex often with many individual components and parameters. Such servers operate in dynamic situations experiencing wide variation in demand or fault conditions (Gorton et al., 2008).

This can be addressed in some ways by clear separation of application logic modules and also through fault-handling capabilities (Gorton et al., 2008).

In their seminal work Gamma et al. (1995) identified that in software design many of the same problems occur again and again and that the underpinning structure that solves these problems can be reused in different cases. They defined 23 of these “design patterns” to solve the most commonly encountered problems they had experienced in development. Patterns fall into three broad categories of being either creational, structural, or behavioural.

Many of the patterns defined by Gamma et al. (1995) are applicable to use in an extensible architecture as they are concerned with defining the way in which objects interact rather than how they perform and operate, in many cases allowing (or indeed enabling) the use of different discrete software components to be used for a role as selected.

In the way of component-based development (CBD) focus is brought to the development of individual functional software components designed for independent reuse in more systems than the one for which they are originally implemented (Ampatzoglou et al., 2011). Szyperski et al. (2002) defines components as “units of independent production, acquisition, and deployment that interact to form a functioning system”. Components are, by definition, for composition; they are joined together to build a complete system, “composition enables pre-fabricated ’things’ to be reused by arranging them in ever-new composites” (Szyperski et al., 2002).

As any engineering discipline becomes mature components are seen as a logical step and software component-based development is a logical step as was the move to components within physical engineering, for example automotive manufacture which has become increasingly based around components (Szyperski et al., 2002). Developing individual components and then building an overall system from multiple components rather than as a single entity can allow for wide variation in reconfigurability and functionality combined with the ability to quickly respond to requirements changes in a highly efficient manner (Szyperski et al., 2002).

Object-oriented Application Frameworks (OOAFs) are an approach to aid the reuse of proven software technology (designs and implementation) with the goal to reduce cost and increase both quality and flexibility (Fayad and Schmidt, 1997). In this model a OOAF is a partially completed application which can then be specialised producing a final complete and specific application (Fayad and Schmidt, 1997).

Use of OOAFs encourage modularity by encapsulation of implementation details behind static interfaces, reuse by provision of generic modules and extensibility again through strong decoupled interfaces. OOAFs are best used "in conjunction with patterns, class libraries, and components", providing an overarching mechanism by which other technologies can be leveraged together (Fayad and Schmidt, 1997).

Software Product Lines (SPLs) can also offer a way to deliver flexible and reusable software through functionality planning and systematic development as a further evolution of CBD. SPL practice revolves around the separation of domain and application engineering as well as the separation of commonality and variability (Leitner and Kreiner, 2011).

In this model software products are seen as a collection of core components, that can be configured and arranged to form individual, but related, software products. SPLs accomplish this through shared reference models and identified variation points (ter Beek et al., 2012), with variability included specifically in the design and development process (Leitner and Kreiner, 2011).

2.3.1 Plugin Architecture

Modular (and later component-orientated) design have been in consistent use and widely held to deliver benefits (Chatley et al., 2004b). Plugins are another step, one in which components are not required but optional, can add additional functionality, and can be easily changed (Chatley et al., 2004b).

Adaptability requires a software system that is "adapt-ready", able to change behaviour either within the software architecture of through transparent modification by an external entity (Sadjadi et al., 2004). Generally these changes can be made at compile-time, alternative modules or code compiled (and/or linked) or at run-time, dynamically as the program is being executed (Sadjadi et al., 2004).

> “Plugins are optional components which can be used to enable the dynamic construction of flexible and complex systems” (Chatley et al., 2004b).

Such an architecture provides for run-time evolution an advantage of which is that systems can be updated without interruption (Oreizy et al., 1998).

Oreizy et al. (1998) state that use of modular or component architecture changes the focus from specific code to granular components and connection structures, abstracting away from tiny details to a wider view of the system as a whole, its structure and processing elements.

(Chatley et al., 2004b) identifies four key requirements a plugin architecture can address:

1. Extension of functionality
2. Modular seperation of large systems so only needed components are used
3. Upgrade of existing systems without full re-deployment
4. Ability of third-parties to make changes without making changes to or needing access to the core system

To ensure maximum functionality and minimum potential for side-effects of plugins a clear structure in which they operate, including interfaces and bindings, must be defined along with behavioural characteristics such as latency and error handling (Chatley et al., 2004a).

Further, a plugin architecture requires the ability for plugins to be registered by the system and the functionality or interfaces provided to be determined (Chatley et al., 2004a).

2.4 Development Environments

Each year new programming languages are created to fulfil changing paradigms and/or hardware. Many of these are domain-specific but some are designed as general-purpose languages (GPLs) and while some are widely used at all levels others occupy a niche or are predominantly used in academic research (Al-Qahtani et al., 2010).

The "comparison of programming languages is a common topic of discussion among software engineers" (Al-Qahtani et al., 2010).

High-level languages, those abstracted away from hardware specifics, provide programming benefits reducing many of the complexities found with low-level resource and memory management (Al-Qahtani et al., 2010).

In order to optimise the system architecture and function it is important to evaluate the level it is written in. Therefore to identify the best candidate language for this project it is necessary to compare the common widely-used high-level languages suitable for use in creation of a web-based system (as identified as a requirement in section 2.2) and their associated benefits and limitations.

2.4.1 Java

Java, designed by James Gosling of Sun Microsystems, is a portable object-orientated language (Al-Qahtani et al., 2010). Java source is compiled into "byte code" which is then executed by a Java interpreter known as a Java Virtual Machine (JVM) (Dwarampudi et al., 2010).

Al-Qahtani et al. (2010) identify the benefits of Java as offering increased memory management, cross-platform support, better inherent security (as memory is managed and unauthorised access impossible), and easier programming than C++.

In the context of web-based applications Java offers the ability to use servlets (pre-compiled Java classes), Java Server Pages (JSPs), or the creation of a standalone HTTP server (Al-Qahtani et al., 2010).

2.4.2 PHP

PHP is a widely-used scripting language primarily intended for web development with the ability for PHP code to be seamlessly included in with standard HTML. It is run through the PHP runtime which parses the source code along with any static content and then produces output (Al-Qahtani et al., 2010).

Because of its web focus Dwarampudi et al. (2010) recommend PHP over the other languages considered for the implementation of web applications or web services.

PHP 5 introduced abstract methods and classes to the existing object-oriented model (Al-Qahtani et al., 2010) and so PHP is now capable of implementing strong design patterns through composition, inheritance, interfaces, and delegation (Sanders, 2010).

2.4.3 ASP.NET

Visual Basic .NET (VB.NET) is Microsoft's implementation of their Visual Basic language within their .NET framework. The development of web applications in Active Server Pages .NET (ASP.NET) is a subset of VB.NET and uses the same coding styles and syntax.

.NET code is compiled into an intermediate common-language "byte code" which can then be executed by any system with the .NET Common Language Runtime (CLR) (Dwarampudi et al., 2010).

2.4.4 C++ CGI

C++ began as an extension to C to facilitate object-orientation by Bjarne Stroustrup. It is a statically typed language where code is compiled and linked to executable machine code for a specific hardware and operating system profile (Al-Qahtani et al., 2010).

C++ provides weak memory safety but this can be addressed through the implementation and use of "safe" memory management routines enforcing, for example, maximum data lengths in operations (Al-Qahtani et al., 2010).

2.4.5 Comparison

From Dwarampudi et al. (2010) the following comparison can be derived:

Java	**Security**: High **Availability**: Good (JRE installed widely) **Web Applications**: Requires extensive use of libraries **Performance**: Low **Reflection**: Good support **Ease of First Use**: Medium
PHP	**Security**: Medium (very application dependent and a lack of native input filtering) **Availability**: Good (installed on majority of web servers) **Web Applications**: Good, seamless integration and support for HTML and good support for web standards **Performance**: Medium (fast for page generation) **Reflection**: Possible but large performance impact **Ease of First Use**: High
ASP.NET	**Security**: High **Availability**: Medium (OS vendor limited) **Web Applications**: Good, web application framework available **Performance**: Low **Reflection**: Good, dynamic and static support **Ease of Use**: High
C++	**Security**: Low (requires extensive security control in the application) **Availability**: Good (native code execution) **Web Applications**: Medium, CGI possible but complex and limiting in nature **Performance**: High **Reflection**: Medium, some support **Ease of Use**: Low

Table 1: Language Comparison

2.5 PHP Development Aspects

2.5.1 PHP Language

PHP (short for PHP Hypertext Preprocessor) is a widely-used scripting language (as described in section 2.4.2). Because of its web application focus (section 2.4.2; Dwarampudi et al. (2010)) and the familiarity of the author with the language it has been chosen as the implementation language (the rationale is contained in section 3.3) so the literature review will consider key aspects of PHP development.

2.5.2 Security

PHP applications are often seen as highly insecure and bug-ridden owing to the large number of vulnerabilities found in these applications. In fact such vulnerabilities are often simply the result of inexperienced, possible even first-time, programmers using PHP because of its low entry bar and ubiquity. PHP can be secure if applications are developed in a security-oriented manner (Al-Qahtani et al., 2010).

As websites become more complex, evolving to meet new requirements, the scope of potential issues including security vulnerabilities grows (Letarte et al., 2011).

A commonly encountered security vulnerability in PHP applications is SQL injection which is the ability of a malicious user or script to insert additional database commands into an insecure input field then parsed into an SQL query by the application (Sadalkar et al., 2011).

Al-Qahtani et al. (2010) and Sadalkar et al. (2011) identify the most important steps for PHP security as:

- Validating using input
- Preventing SQL injection
- Preventing cross-site scripting (XSS)
- Preventing remote execution

- Enforcing of temporary file security
- Preventing session hijacking

2.5.3 Extensible Architecture

Although there is little academic work detailing implementation of extensible PHP architectures there are a number of projects already existing and widely used that support extensibility. For the purposes of this review three of the larger and more mature projects are considered, Wordpress (blog and online publishing software), phpBB (web forum software) and MediaWiki (online publishing and collaborative editing platform).

PHP does have support for reflection (section 2.4.5) but this has a significant performance impact for code analysis. None of the large projects providing extensibility analysed use this technique. This is unlike Java applications such as Eclipse which make wide use of reflection allowing automated identification of plugin methods and functionality rather than requiring specific registration of functionality (Amsden, 2003).

phpBB

phpBB uses a modification approach (MODs) where changes are made to core application code to support the desired functionality. These are then available for download, providing "patches" to existing installations, modifying a system to provide the new functionality (phpBB, 2011).

These patches are in the form of "diff" files containing individual byte or line changes (insertions, deletions or modifications) for system source code against a reference base version (phpBB, 2011).

In this way any or all features of the system can be updated or completely changed with the modifications being made directly to the source code of individual installations. After the MOD has been applied the target system source code should be an exact replica of the modified version from which the MOD was generated, assuming no other changes have been made and the MOD was applied to the correct version.

Although offering a powerful model for complete modification great care must be taken with this approach to ensure that functions do not overwrite, are applied to the correct version, different MODs do not interfere with each other, and when updating the core system to newer versions (phpBB, 2011).

Wordpress

Wordpress uses a plugin architecture where plugins, once installed, affect system behaviour using a clearly defined interface with the core system (Chan, 2011).

Plugins are created by inheriting a class from a base plugin class and then placed in a defined structure within the installation directory. When the plugin class is loaded it registers itself along with certain behavioural functionality with the core system, which then calls the appropriate plugin methods as and when required to perform the plugin functionality (Chan, 2011).

MediaWiki

MediaWiki uses an extension architecture (similar in many ways to the Wordpress plugin architecture) where extensions affect system behaviour through a defined interface for registration and execution (MediaWiki, 2012). Each extension, confirming to a strict naming and file convention, registers the features it provides with the core system, providing the names of methods or classes to be used in the execution of the registered functionality (MediaWiki, 2012).

Extensions are used to extend the functionality of specific areas of the MediaWiki software including markup, reporting, administration, automation, user interface and security (MediaWiki, 2012).

3 Analysis

3.1 End Users

FreeDESK (the system developed for this project) is developed to be released as free software and ultimately used by others. There are a number of different groups and types of end-user and in order to differentiate these it is important to identify and categorise them, providing each with a term that can be used throughout to refer to that specific group.

- **End-Administrator**: The end-administrator is the individual (or team such as an IT service team) who are responsible for the initial installation and ongoing configuration and management of the system.
- **Analyst**: FreeDESK is designed to be used by service-desk personnel to raise, manage and track requests to their service. Analyst is used as a generic term to describe these professional team members who, using the system, log and/or then perform the actions required by the requests which have been raised.
- **Customer**: Customers are the customers of the service desk which is supported by FreeDESK. They are not customers of the FreeDESK system itself but the customers who's requests are raised within the FreeDESK software.
- **Extension Developer**: Extension developers are third-parties who develop extensibility using the FreeDESK system through the available extensibility options.

3.2 System Form

A system can be developed as an application (either a native application compiled for and using the extensions of the target environment or as cross-platform for example in Java with Swing) or as web-based with the user interface being provided in a web browser. Section 2.2 identified a desirable feature of a service desk as being web based, to allow for ease of access from any browser-equipped client system and minimising the deployment requirements (no application-specific software needs to be deployed to each client).

Developing a system which is web based also provides the ability to offer alternative interfaces through the same channel while utilising the same core components, for example a mobile-optimised or customer interface.

For these reasons it has been decided that a web based system would be the optimal form for this service desk.

3.3 Development Language

Section 2.4 identified and analysed the key features of alternative development languages. For a web based system the choices are C++ CGI, Java, ASP.net and PHP. Each of these offers advantages and disadvantages which are identified, for example C++ offers high performance but is more complex than other languages analysed and requires careful attention during development to deliver a secure system.

Given the recommendation of Dwarampudi et al. (2010) combined with wide availability and the existing familiarity of the author with the language PHP, is the most appropriate choice and so has been chosen as the development language of the system.

3.4 Extensible Architecture

A core aim of this project is to develop an extensible system, one that can be extended by the end user to provide their own implementations of core features and to add completely new functionality. Section 2.3 of the literature review identified current key ideas with regard to extensibility including the use of design patterns, component-based development, application frameworks and software product lines (SPLs). In these techniques the overall system is made up from numerous individual components built together and interacting to provide the system as a whole. Section 2.3.1 dealt with a further step toward using plugins, an approach whereby new components are not required but rather optional, can easily be changed and can provide additional functionality (Chatley et al., 2004b).

Within a web-based context with PHP as the development language (as identified in sections

3.2 and 3.3 respectively) a range of prior art exists where systems are making use of some form of extensibility and/or a plugin architecture. Three of the most widely used systems with these features are identified and included in the literature review (section 2.5.3).

It can be seen that as well as individually offering powerful approaches to extensibility, these techniques can be used in combination. For example a system built using components in an application framework can also make use of design patterns in the way in which the components are defined and interact. A composite approach can harness the benefits from different approaches and provide for flexibility within a framework facilitating easy creation and compliance of new components.

In considering which specific approach or approaches to take with regard to the actual implementation of an extensible system in PHP the most applicable technique, and that offering the greatest comparability with design patterns and application frameworks is that of a plugin architecture. Use of code replacement (as with the phpBB MOD model) does offer the option for any level of change being made to the system by direct modification of primary system source code but has significant drawbacks with compatibility with different base versions and other different extensions.

For these reasons it has been decided that a plugin architecture utilising design patterns within an Object-Oriented Framework and incorporating one idea from SPLs (the concept of identifying static and variable functionality), is the best solution for the system. The system will be designed and implemented in a modular (component-based) fashion and areas of specific variability will be identified at each stage of development. These areas will provide the option for alternative concrete methods of implementation to be included (replacing primary system components). Additionally an open plugin framework will be designed and implemented allowing for dynamic runtime inclusion of plugin modules. These plugins can then add or replace functionality as well as registering for certain system hooks to be passed to them for event action handling.

This extensible functionality through plugins will take the form of run-time adaptability, with modules loaded into the codebase at runtime. This is something easily accomplished in PHP (code being dynamically loaded and executed) and in fact no compile-time adaptability is possible as there is no specific compilation step visible in the execution process. The nearest equivalent would be a form of static source linking where individual source code modules and files are statically set for inclusion in the project, a less flexible approach offering no performance benefits.

Although some languages, notably Java, make use of reflection for extensibility whereby new modules are dynamically examined and analysed for methods and names, use of reflection in PHP has severe performance impacts and is not widely used. To accomplish modular extension the FreeDESK system will therefore use a set interface format with registration capabilities, module source code files will conform to a standard naming convention (including class name). When the standard-named files are included and their classes instantiated, they will in turn call set registration interface methods within the core FreeDESK system which will register functionality provided and then call on the module classes as and when required.

4 System Design

4.1 Use Cases

In order to design a system it is necessary to first define exactly what the core uses and interactions with the system will be. To accomplish this a number of basic use cases have been defined for a service desk system covering interaction from a call logging/frontline support, analyst and customer perspective. The full textual use cases for these examples are contained in Appendix C. Although specific work practice will vary widely from service desk to service desk, something the extensibility of the system specifically addresses, in order to inform use cases and illustrate standard practice a generic workflow of a service desk (without any complex business functions such as a review function of resolved requests or specific third-party assignment functions) is shown in figure 1.

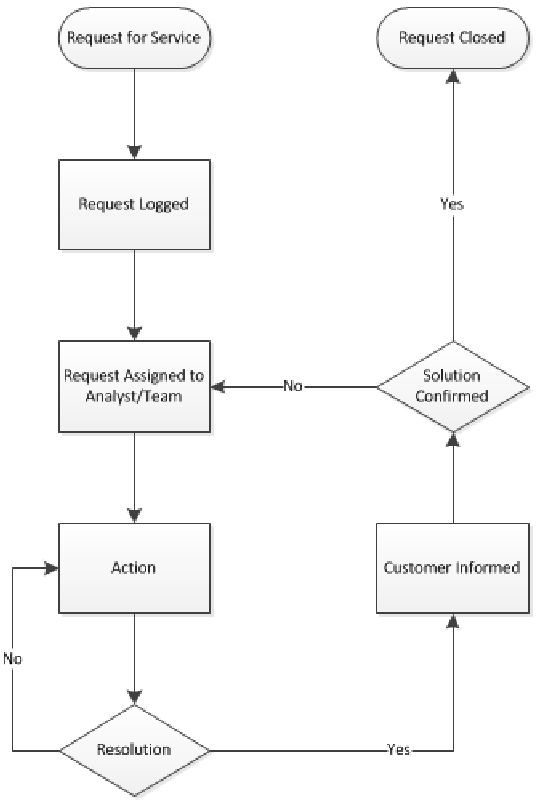


Figure 1: Generic Service Desk Workflow/Process Diagram

Figure 2 illustrates a use case from the perspective of a customer interacting with the system through a centralised call logging/frontline support service. In the first instance the customer contacts the call logger who takes the details and creates the request. The call logger will also handle ongoing communication with the customer, providing updates on requests and checking that the solution offered has indeed resolved their request.

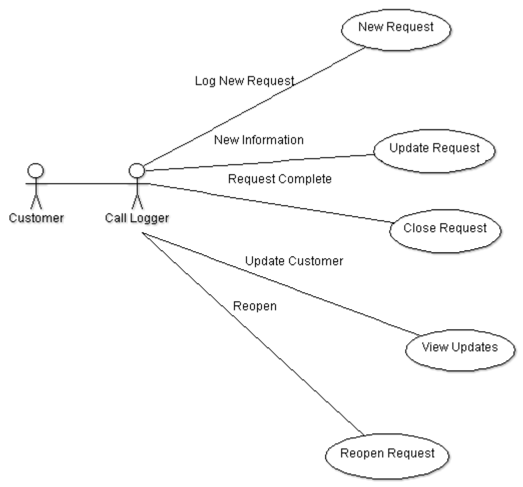


Figure 2: Use Case for Frontline/Call Logging

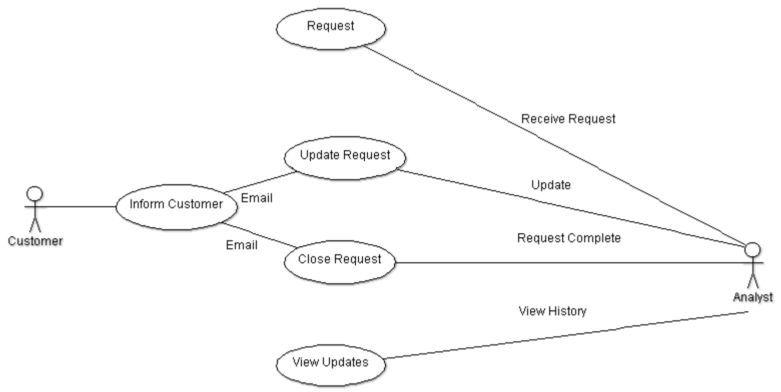


Figure 3: Use Case for Analyst/Specialist

An analyst/specialist interacting with the system is illustrated in figure 3. Once a request is logged (either by a call taker or directly by the customer) it has been assigned to a specialist or domain expert for their action and resolution. The analyst can view the history of the request and make updates to it, ultimately resolving the issue and then closing the open request.

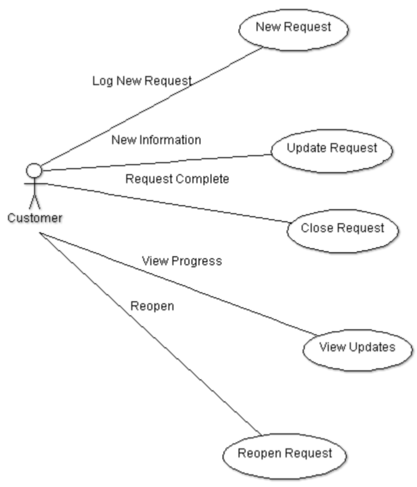


Figure 4: Use Case for Self Service

Customer self-service interaction is shown in figure 4 where the customer can, through a self-service portal, interact directly with the system to create requests, review existing requests, re-open and close requests as they wish.

4.2 Specific Requirements Analysis

Drawing from the service desk domain features identified in section 2.2 and the analysis of extensibility requirements (section 3.4) a range of system features can be identified. These features are then used to complete the following MoSCoW[2] analysis:

Must Have	Request tracking with unique reference Prioritisation (logging and view in relative priority) Customisation options (configuration) Security (see below) Extensibility options (plugins)

[2]The MoSCoW method of analysis is based around the concept of identifying what features a system *must* have, *should* have, *could* have, and *won't* have

Should Have	Performance monitoring Web-based system Screen update mechanism to show new updates through the analyst portal (section 4.3) Update mechanism to update customer (email)
Could Have	Customer portal (self-service) Mobile-accessible analyst portal Integration options with external systems
Won't Have	Visual business process management tools

Table 2: FreeDESK MoSCoW Method Analysis

The system must have robust security, meeting not only the requirements set out in section 2.5.2 to counter external threats but also providing for levels of internal security, controlling which authenticated users have access to what content and ability to change which settings. To enable maximum flexibility a granular security model will be implemented where specific restricted tasks have a particular permission. The security model will support groups for ease of administration and also an optional default permission flag to allow or deny all those permission roles not specifically defined for the user or group.

4.3 User Interface Update

An identified requirement is that of having an update mechanism to reflect new changes within the analyst portal (new requests assigned or updated) without the need for a manual refresh. The most common way to accomplish this is through either an automated page refresh (reloading the entire page contents from within the browser, equivalent to manually clicking the refresh button) or a periodic poll of the server to detect and display new updates.

Periodic polling is the most common and effective method as it avoids full page refreshes (with associated delays and overheads of the entire page being rebuilt) but an alternative method may be to use so-called "Long-Held HTTP Polling". This is an alternative polling/refresh architecture designed around the concept of connections being held open between the client and server allowing for faster updates.

To consider the feasibility of this type of architecture an investigation and experiment should be made using the FreeDESK technologies of PHP, an SQL server and Javascript/XML. The detail of this experiment, results and conclusions are contained in section 5.

4.4 Extensible Architecture Definition

4.4.1 Plugin Architecture

The plugin architecture for the system will offer two distinct types of interface: replaceable components for specifically identified areas of variability (as in a SPL) and a generic plugin module interface for general unspecified extensions to the system (providing new additional features within and using the provided interfaces rather than simply replacing or changing the method of implementation of pre-existing function). The areas of variability for which components can be substituted to handle the implementation are as follows:

Functionality	**Description**
Data Storage	The underlying relational SQL connection (to change the target database software)
Authentication	User authentication process (to allow alternative authentication methods rather than purely the inbuilt user table)
Appearance (Skin)	The visual display of the system as it appears to the end user
Logging	The method(s) used to log system events (allowing logging to an external mechanism than the inbuilt syslog table)
Requests	The concrete entity class used to represent a request and deal with changes such as updates
Entity	The concrete entity class used to represent a "generic entity" (a stored entity other than a system entity)

Table 3: Areas of Variability in the System

Areas of the system which plugin modules can modify to allow for their provision of additional functionality are as follows:

System Area	Purpose of Modification
Menu Structure	To allow for the creation of new menu options in the user interface
Pages	To allow for the display of module-generated content within the standard interface
Appearance (limited)	To allow the inclusion of module-specific CSS information within the standard interface
Scripts	To allow the inclusion of module-specific Javascript to the end-user browser within the standard interface to support client-side module functionality
Security	The creation of new security permissions to allow granular security of module functionality

Table 4: Areas of Plugin Module Modification

By providing a mechanism for plugin modules to modify the areas above a module can provide user-accessible custom content and options, controlled and secured through the main standard interface.

4.4.2 Application Programming Interface (API)

The system will provide an API in two ways, through direct code interface (any external script running locally on the same system may interface directly with the system code) and through an external HTTP-XML API. The direct code interface will consist of all public methods and members in system classes, each class being fully documented both in the code with phpdoc comments[3] and in the technical system documentation (section A).

The HTTP-XML API will provide an external HTTP based protocol (with XML returned data)

[3]A system similar to javadoc of defining a standard comment structure detailing, for example, the purpose, parameters and return values of a method

for the majority of system functionality. All common and core functions of the system will be provided by the HTTP-XML API and these will be commonly used by all the core interfaces of the system. By handling the majority of events and actions through the HTTP-XML API it will be certain during the development of the system that a fully usable and stable API is exposed that can also be used by external systems and other developers. Provision of this HTTP-XML API will also allow for the development of user interfaces in a "web 2.0" manner[4].

The HTTP-XML API will be constructed in such a way as to allow for interaction purely through it's exposed methods, allowing the full cycle of exchange from authentication to action and disconnection of a session. All HTTP-XML API calls and return data formats will be fully documented for use by other developers (section A). A full list of desired functionality and protocols will be defined with the software design (section 6).

4.4.3 Skinning

Skinning provides a mechanism by which the visual display is abstracted away from core data; rather than output being directly generated containing all the relevant HTML markup a set of files are used (which make up the "skin") controlling the visual output. In this way the visual appearance of the system can be easily changed with an alternative skin providing a different set of base markup within which data is displayed.

To implement skinning as an area of variability within the system a display engine must be created to include and output the relevant display files from the configured skin to the user. Skin files will be both static (for example containing CSS) and also dynamic, able to take outside data provided by the core system and render it for output in a chosen manner. Using this method the creation of a list becomes, rather than a simple set of HTML output from the core system, the inclusion of multiple skin elements for example a list start, each individual content item (as a dynamic skin item with the item contents passed in a predefined manner) and a list end.

Wherever possible during system development use of individual skin elements should be used for any user interface items. In the process of development a "default" skin will be created

[4]Using Javascript asynchronous requests (AJAX) for data exchange rather than standard POST/GET and page reloads

with in-code documentation (as well as detailed technical documentation for skinning with the overall system - section A) acting not only as the standard interface for the system in basic configuration but also as a reference for any developers wishing to create their own skin.

An installed skin can provide any or all of the skin items with a fallback provision to use the item within the default skin. This allows for easy changing of just some individual skin items without having to copy or re-create the entire set. The choice of skin for display interface can be made on a system-wide or user-specific basis (if the user has permission to change their display options).

5 Evaluation of Long-Held HTTP Polling for Client Update

5.1 Overview

In section 4.3 two alternative methods to meet the requirement for client updates were identified, typical interval based polling and long-held HTTP requests. To establish the feasibility of the less-common long-held method and evaluate relative performance an experiment was proposed. The design, implementation and results of this experiment are detailed in this section.

The Hypertext Transfer Protocol (HTTP) is a protocol built around the request/response paradigm whereby a client establishes a connection to a server, makes a request, receives a response and the connection is then closed (Loreto et al., 2011).

As the server cannot therefore make connections to the client, for example to notify it of events, the client must repeatedly poll the server for updates, an inefficient method known as "short polling" involving significant overhead as new connections are set up then torn down for each poll and client updates can only be received once per polling interval (Loreto et al., 2011).

Short polling occurs "blindly" irrespective of a data update being present and in order for low latency (high data/display accuracy) the polling interval must be low meaning a high use of resources (Bozdag et al., 2007).

One approach to resolve this issue is the use of long-held HTTP polling ("long polling") a technique where the connection between the client and server is held open until data is available and sent (Loreto et al., 2011). Current examples of this technology in use include Google products such as gMail and gTalk (Russell, 2006).

As the FreeDESK system will make use of asynchronous Javascript XML requests (AJAX) and will need a mechanism to ensure screen updates are picked up and reflected within the user interface consideration must be given as to how this is accomplished, through traditional short polling or using a held open connection with long polling. To examine the difference in latencies between these techniques an experiment was performed.

Although Bozdag et al. (2007) presents a similar experiment this was constructed using Java as a base technology and also a messaging bus. The architecture of the FreeDESK system being PHP and a relational SQL database has a different structure so this experiment is necessary to check performance against the specific architecture of the FreeDESK system.

Long polling offers potential benefits with latency but also has inherent issues including the server overhead from connections being held open, timeout problems where the connection is timed out by the client and/or server and caching where request responses are cached in an intermediate proxy or web cache and then returned time and again (Loreto et al., 2011).

There can be no doubt that the issue of server resources is serious especially in large or shared deployments where many clients are using the same HTTP server. Owing to a lack of resources and in order to keep the experiment focused (also because of the existing work Bozdag et al. (2007) has performed to quantify this issue) consideration of server resources are outside the scope of this experiment.

In this experiment timeout problems were mitigated with a built-in drop and reconnect period whereby, after a specified period (25 seconds) before the browser request or PHP script will themselves timeout, the connection is closed with no data, causing the client to perform an immediate reconnect. To avoid caching conflicts all data will be sent and received using HTTP POST rather than HTTP GET which should not be cached by intermediate proxies. As an additional safeguard a “nocache” field will also be passed containing a randomised (and therefore different) string for each request which, though ignored by the server, will mean the POST request contains different data each time and so should definitely not be cached. Data returned from the server will have headers marked to show it should not be cached and has “expired” already.

5.2 Hypothesis

Before conducting the experiment the following hypothesis were made both the expected outcome (H_1) and also a null hypothesis (H_0).

H_1: Long polling will offer significant reductions in latency bring average update response time to 0.5s or less

H_0: Long polling will show no significant reductions in latency over short polling

5.3 Experiment Design

To replicate the architecture of the FreeDESK system as much as possible the experiment was designed with a PHP front-end and an underlying SQL database. When a test session was in progress it would move through stages of trying short polling (with 10 and 5 second polling intervals) and long polling. A series of messages would be generated at randomised intervals in the database and then provided to the test client which would then acknowledge their receipt allowing the server to calculate a total "acknowledgement time" (the time from the creation of the message to the acknowledgement being received from the client).

Figure 5 illustrates a timeline using short polling to a system with an underlying SQL database and figure 6 shows the timeline in a long polling context.

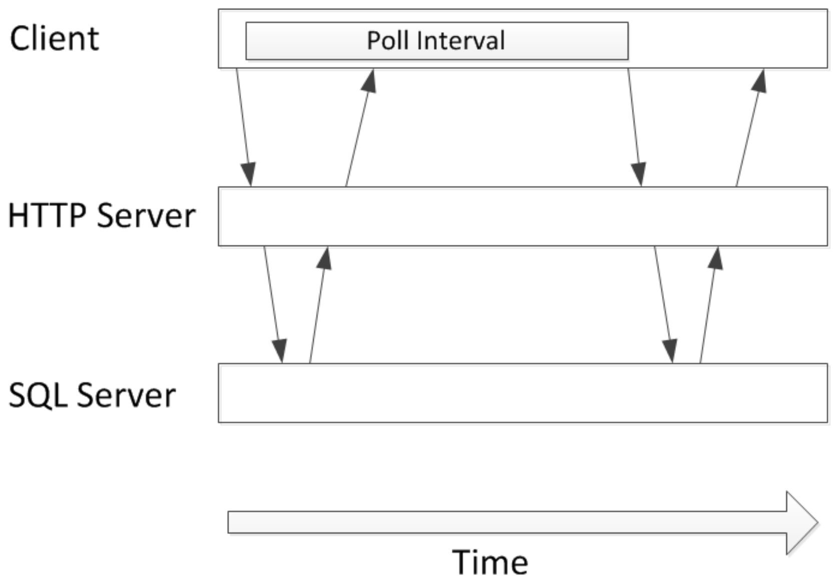


Figure 5: Short Polling to HTTP Server with Underlying SQL Database (not to scale)

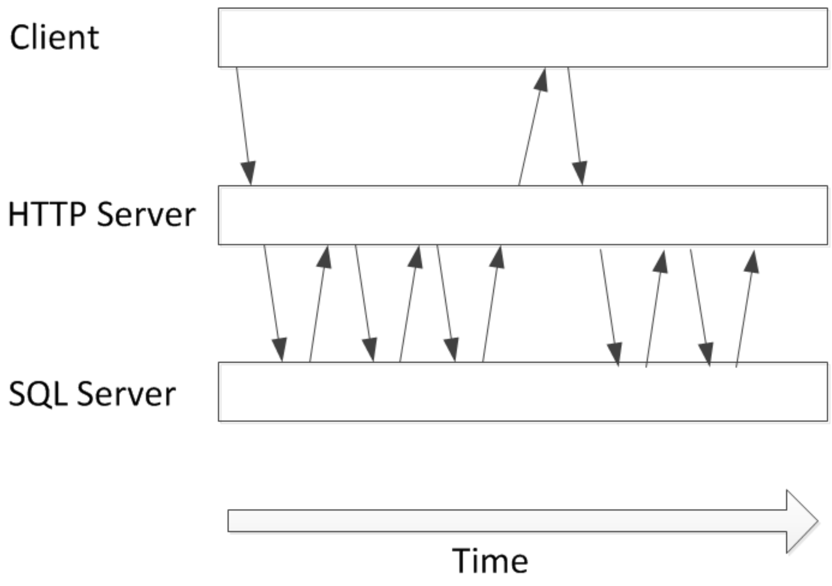


Figure 6: Long Polling to HTTP Server with Underlying SQL Database (not to scale)

In order to detect the effects of general network latency intermediate stages on the client performed a “static fetch” where a small (65 byte) static XML file was be fetched from the server with the client recording the time taken from request to response and then passing this information to the server for inclusion in the overall dataset. This allowed for identification of sessions with unusually high general latencies that could either be removed from the dataset or adjusted accordingly giving a truer estimate of latencies in the fetching only.

Stage	**Client**	**Server**
0 Registration	Generate randomised session ID and register with server	Register session and spawn message generation thread
1 Static Fetch	Perform three fetches of static content and record times	
2 Short Poll (10s)	Perform periodic (10s) poll of server for new messages, acknowledge receipt immediately	Check for new messages in SQL database and provide them, record acknowledgement times
3 Static Fetch	As 1	
4 Short Poll (5s)	As 2 with 5s period	As 2
5 Static Fetch	As 1	

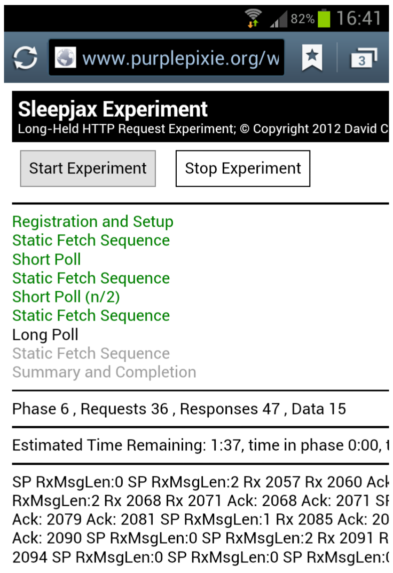
6 Long Poll	Perform long polling, recycling a new connection on every response with acknowledgement of any data received	Keep connection alive and periodically (500ms) poll the SQL database for new messages, dropping the connection after 25 seconds if no data received, record acknowledgement times
7 Static Fetch	As 1	
8 Complete	Send completion message including static fetch times	Mark session as complete (so message thread can exit), move message data into completed table, record static fetch times

Table 5: Long-Held HTTP Experiment Stages

The experiment was written in PHP and SQL for the server-side components and HTML with Javascript for the client-side. The full experiment source code can be downloaded from `http://www.purplepixie.org/davestuff/sleepjax.zip`.

5.4 Experiment

The experiment was conducted over two weeks in early July 2012. The server installation was on a Linux based server running Apache 2 located in a Tier-1 data centre in Manchester, England. Clients connected from numerous locations in the United Kingdom using a number of different web browsers, platforms and connections. This ranged from Internet Explorer under Windows on a fixed broadband line to Android Browser on a mobile phone using GPRS/3G. The experiment running on a Samsung Galaxy SIII phone can be seen in figure 7.


82% 16:41

www.purplepixie.org/w

Sleepjax Experiment

Long-Held HTTP Request Experiment; © Copyright 2012 David C

Start Experiment Stop Experiment

Registration and Setup
Static Fetch Sequence
Short Poll
Static Fetch Sequence
Short Poll (n/2)
Static Fetch Sequence
Long Poll
Static Fetch Sequence
Summary and Completion

Phase 6 , Requests 36 , Responses 47 , Data 15

Estimated Time Remaining: 1:37, time in phase 0:00, t

```
SP RxMsgLen:0 SP RxMsgLen:2 Rx 2057 Rx 2060 Ack
RxMsgLen:2 Rx 2068 Rx 2071 Ack: 2068 Ack: 2071 SF
Ack: 2079 Ack: 2081 SP RxMsgLen:1 Rx 2085 Ack: 20
Ack: 2090 SP RxMsgLen:0 SP RxMsgLen:2 Rx 2091 R
2094 SP RxMsgLen:0 SP RxMsgLen:0 SP RxMsgLen:(
```

Figure 7: Experiment Running in Android Browser on a Phone using WiFi

5.5 Results

Test Sessions	101
Messages Passed	2238
10s Short Poll Acknowledgement	4.8691s Mean σ = 2.997
5s Short Poll Acknowledgement	2.3915s Mean σ = 1.429
Long Poll Acknowledgement	0.3812s Mean σ = 1.429
Static Fetch	0.0563s Mean σ = 0.065

Table 6: Long-Held HTTP Request Polling Experiment Headline Results

Figure 8 shows the distribution of response times using the different polling methods to the nearest 0.1s.

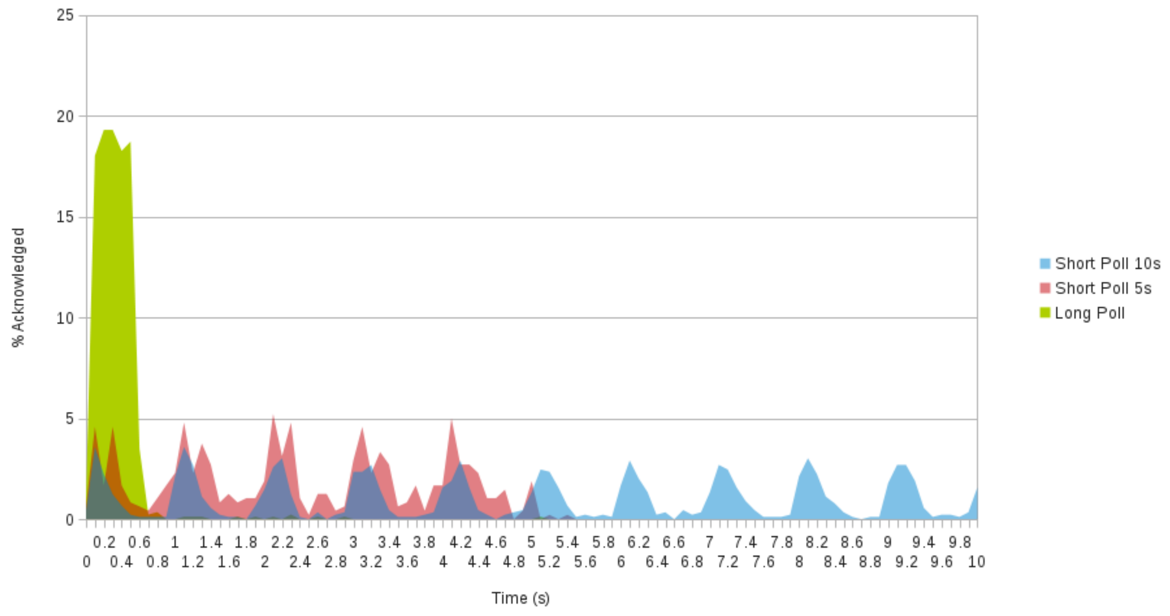


Figure 8: Experiment Acknowledgement Times to Nearest 0.1s

5.6 Long-Held HTTP Request Conclusions

From the results it can clearly be seen that use of long polling provides a significant reduction in latency with the mean acknowledgement time for a data message dropping from a mean of 4.7s and 2.4s respectively for 10 and 5 second interval short polling to a mean of 0.4s for long polling.

Not only was the long poll mean under 0.5s but 85.27% (n=735) of long poll messages were acknowledged in 0.5s or under. This data therefore strongly supports the hypothesis (H_1) and the null hypothesis (H_0) can be discarded.

A portion of the experiment was to repeat a series of "static fetches" and record results to allow for the identification of test sessions where high overall network latency was a factor. It was found that, even with a large range of connection methods ranging from corporate leased lines to GRPS, the static times were consistent with a mean of 0.07s and a standard deviation of only 0.065. For this reason analysis of results against and considering static transfer times was not completed.

An area of consideration not covered in this experiment but analysed in Bozdag et al. (2007) was server resource utilisation. An additional resource constraint in the FreeDESK context is that of repeated SQL database polling by the HTTP server to find new data. Bozdag et al. (2007) showed that although in their experiment use of long polling also offered a significant reduction in latency times server load was greatly increased especially when handling a large number of clients.

Unfortunately given resource constraints this experiment could not replicate the effects of large numbers of clients connecting but it is plausible that server load would increase inline or even ahead of the Bozdag et al. (2007) findings.

Additionally the architecture of a system to support long polling must be necessarily more complex than one for short polling. Rather than allowing a simple "refresh" of data after a polling intervals changes must be detected and then passed to the client as and when they occur. This requires specific handling at all levels of the system and the ability to record and identify changes to data against an existing view of that data, potentially a complex and resource intensive task.

For these reasons although it is concluded that long polling does indeed offer good potential for a near real-time communication mechanism to reflect changes given the tight timeframe for the FreeDESK system, a need for the system to scale and the lack of a suitable testing infrastructure to assure this using long-polling it has been decided not to implement long polling within the current design and build cycle of the FreeDESK system. Consideration will be given for a later version offering this functionality but on a per-user or per-connection configurable basis rather than as the standard mechanism and only after suitable scalability testing has been conducted.

6 Software Design

6.1 Development Model

Development was performed in two distinct phases; the initial design and then ongoing incremental and iterative development (including design refinement and testing as part of the iterative process). The high-level overview of this method is shown in figure 9.

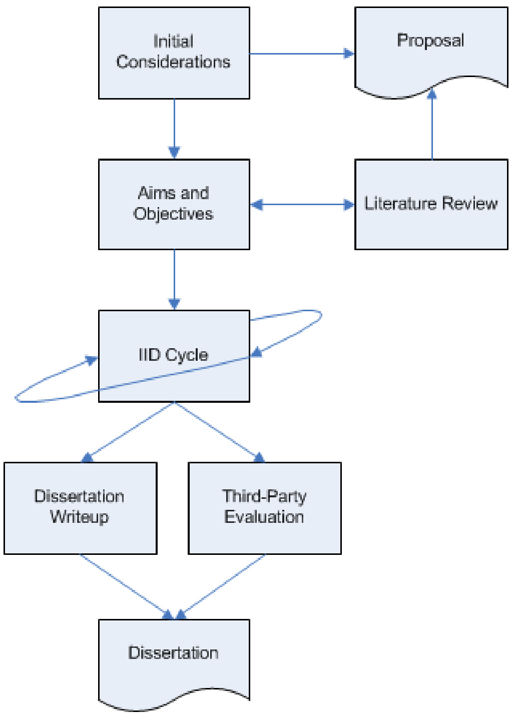


Figure 9: Project Method

The initial design phase defined a series of high-level general use cases for the system (section 4.1) which informed the overall system design (section 4) which was then turned into high-level structural UML (section 6.2).

Development occurred in a series of iterative stages, as illustrated in figure 10, each consisting broadly of the following steps:

- Define requirements to be implemented in this stage
- Plan implementation including any stage-specific detailed design required
- Implement functionality
- Incorporate new functionality into general test base and test system (section 9)

- Resolve any errors identified in testing (returning to the implementation of, if required, to the stage design)

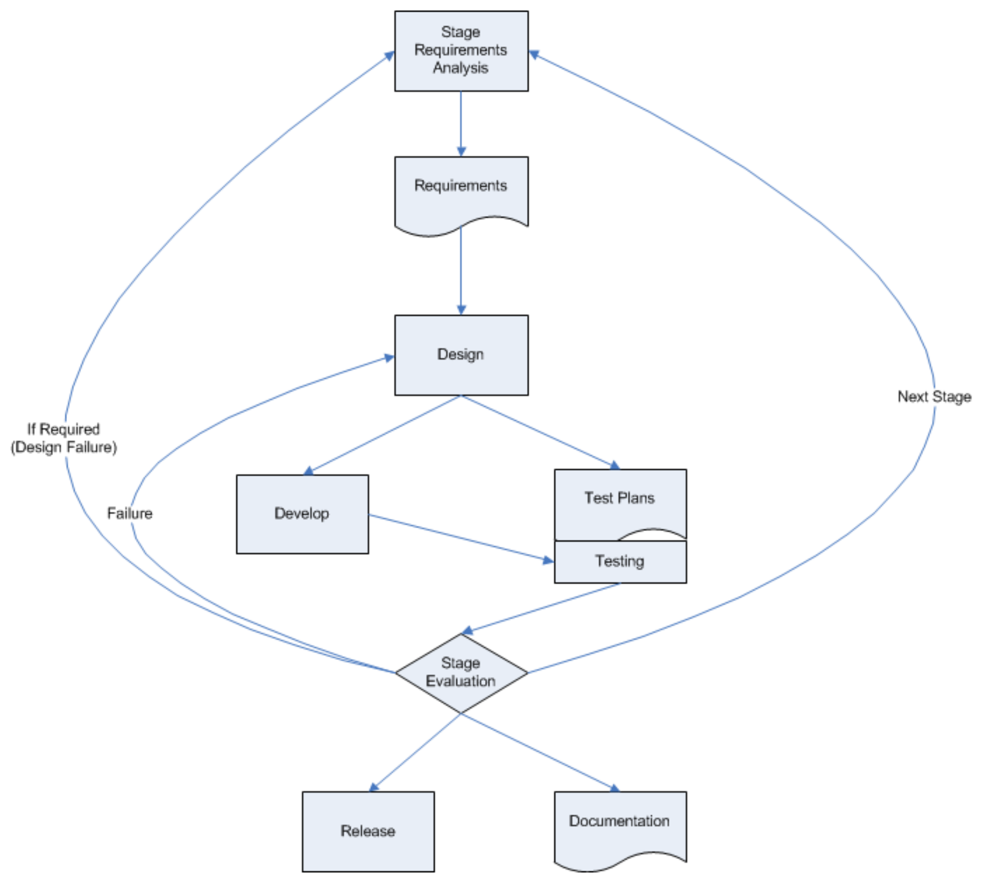


Figure 10: Incremental and Iterative Development Cycle

6.1.1 Model View Controller Pattern

The Model/View/Controller (MVC) as originally used to build interfaces in Smalltalk-80 is a way to separate data storage, application logic and user interfaces (Gamma et al., 1995). This pattern is now widely used in web-based applications providing as it does the ability to decouple layers of the application (Robles et al., 2012).

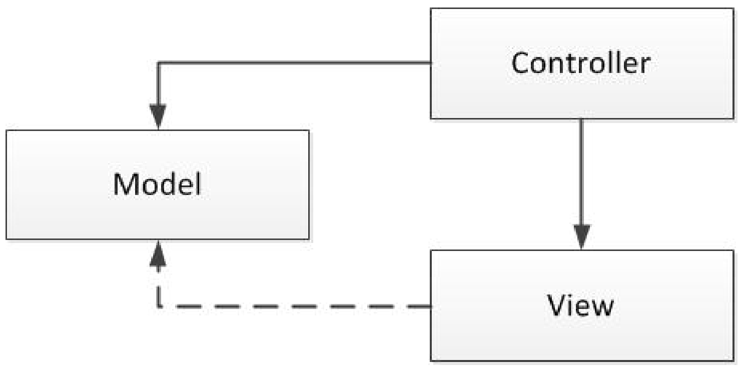


Figure 11: Basic MVC Structure

Figure 11 shows the simplest form of an MVC pattern. The model is the underlying application primarily storing and dealing with state data, the controller deals with how the application responds to input requests and the view is the display of the data to the end user (Gamma et al., 1995). This is more clearly illustrated in figure 12 where annotated data flows are shown detailing the interaction between the components.

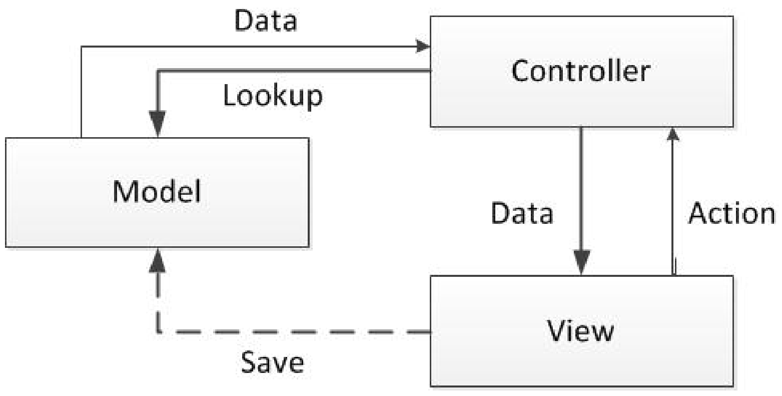


Figure 12: More Detailed MVC Overview

The MVC approach offers a powerful model for the development of an interface driven application (Gamma et al., 1995). In the context of the FreeDESK system where requests may come not only from the standard user interface but also directly to the API it offers considerable benefit and so the system architecture will incorporate this pattern.

Figure 13 shows how the MVC pattern will be implemented in FreeDESK illustrating specifically the standard user interface interactions with the system. Some components span two areas of the MVC pattern for example the page system which performs some validation and data collation (controller) and also generates initial HTML displays for default forms (view).

6.2 High-Level Design

Figure 14 shows a simplified high-level UML class diagram for the FreeDESK system. The core system class (FreeDESK) is responsible for the inclusion and loading of all the main sub-classes and contains public references to their instances. Classes can then make use of each others public methods and members, accessing classes through the reference contained in the FreeDESK class. The FreeDESK class itself contains little other than initialisation control code and then exists purely to composite all the main sub-classes.

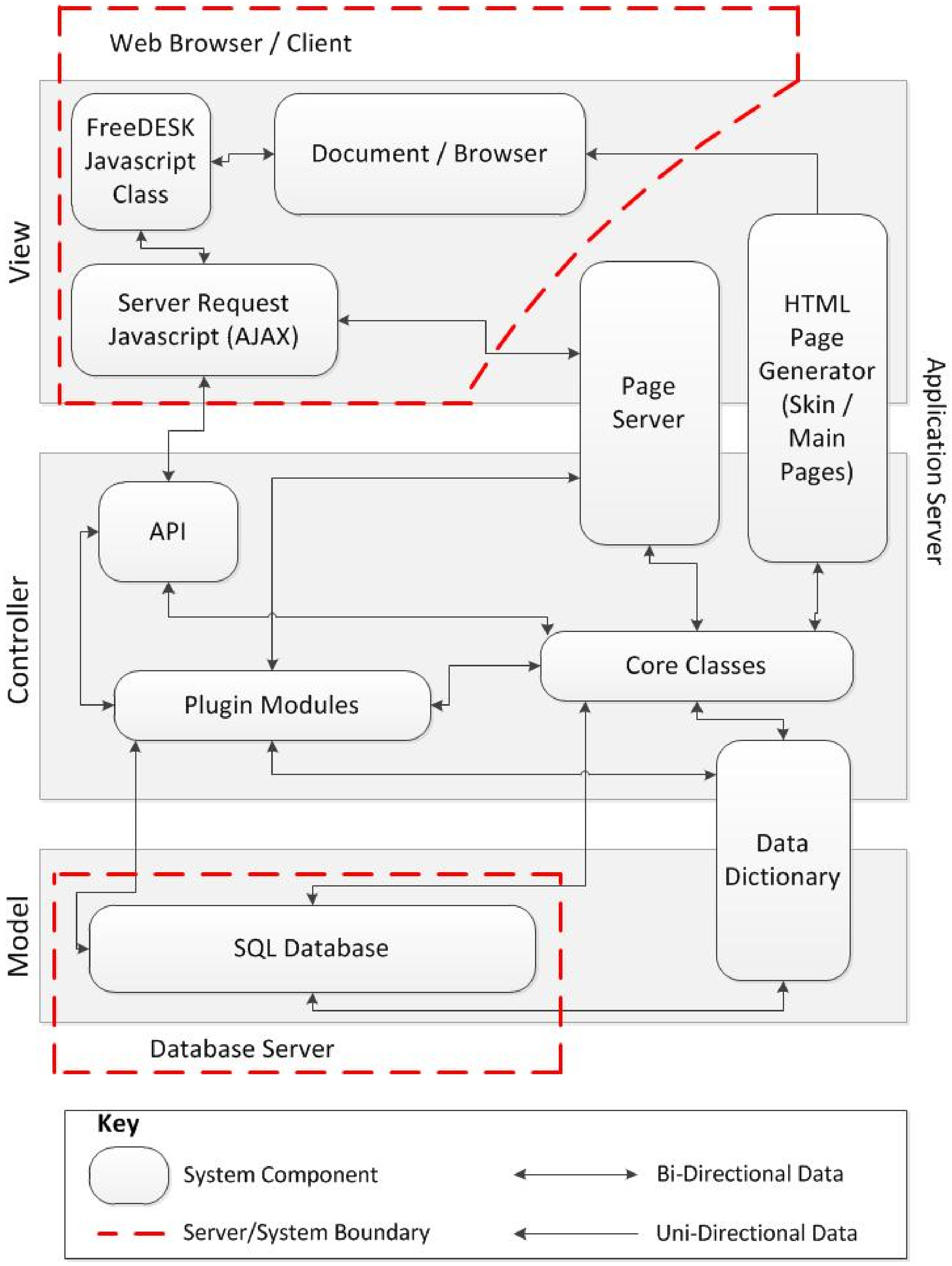


Figure 13: FreeDESK MVC Overview

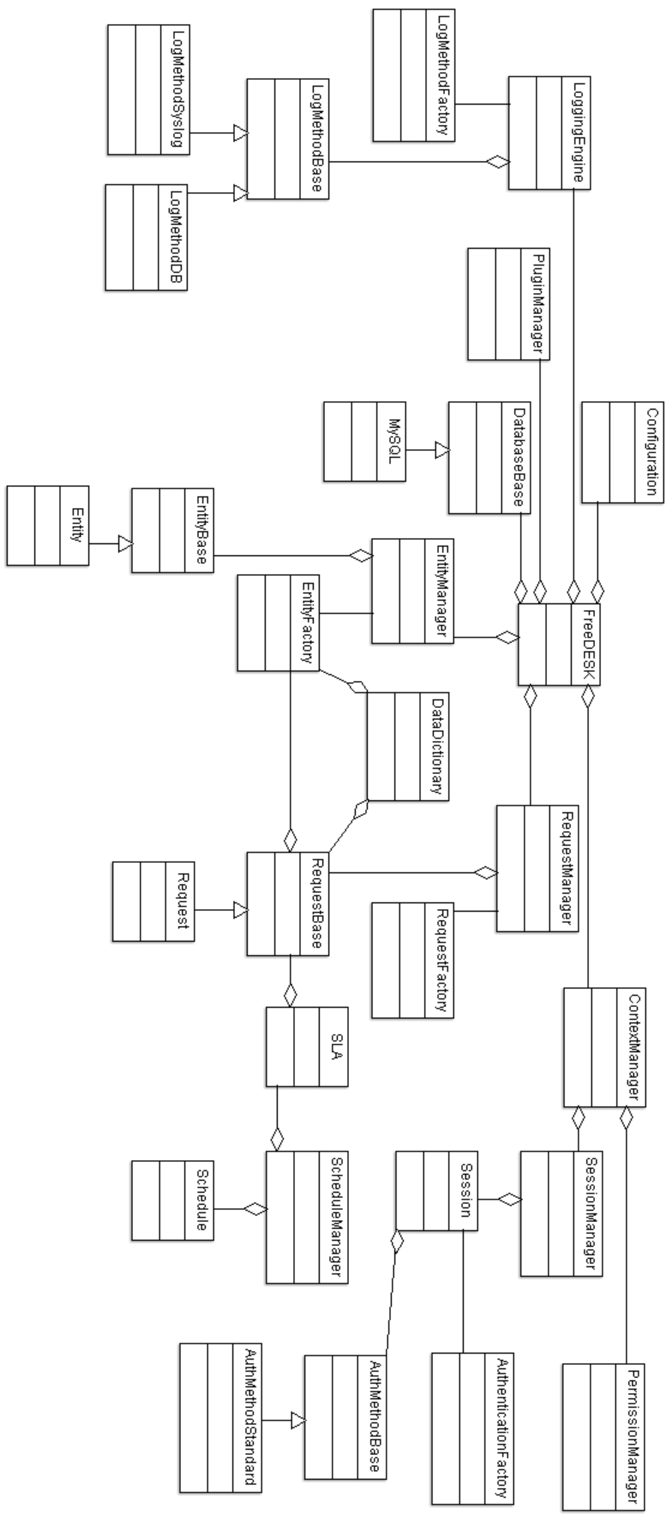


Figure 14: FreeDESK High-Level UML

The core classes of the FreeDESK system are request management (RequestManager and associated sub-classes, section 6.3.1), extensibility through the plugin management system (section 6.3.2), visual customisation through skinning (section 6.3.4) and authentication, access control and security through the ContextManager and associated sub-classes (section 6.4).

6.3 Design of Specific Components

To allow component classes to access the main FreeDESK object (and through that each other), instances of these classes must have a reference available to the FreeDESK instance. This can be accomplished through the use of global variables ensuring the FreeDESK instance is available externally in scope at least during startup (where a reference could be copied to an internal variable). However using globals in this way and in general is frowned upon in development because of security and coordination problems and also in this case it may limit future functionality, for example having a script which has two current instances of a FreeDESK object to two different databases for copying/backup or analysis. The solution employed within FreeDESK is for component classes to each contain a private reference to the FreeDESK instance which created them, a reference that is passed when the class is created.

For example to implement the component SomeComponent with this model a private variable is declared to point to the instance of FreeDESK and then the constructor receives the instance by reference (rather than value which would just create a copy of the FreeDESK instance) and assigns that reference to the internal variable. A code example showing SomeComponent and the code used within the core FreeDESK class itself is as follows:

```
class SomeComponent
{
  private $DESK = null;

  function SomeComponent(&$freeDESK)
  {
    $this->DESK = &$freeDESK;
    // Call AnotherComponent::SomeMethod via FreeDESK:
    $this->DESK->AnotherComponent->SomeMethod();
    }
}
... in FreeDESK class:
$this->someComponent = new SomeComponent($this);
```

6.3.1 Requests

The entire concept of FreeDESK revolves around requests, these are the requests for service or action that are the primary purpose of a service desk system to log, update and manage. To incorporate flexibility and extensibility requests can be internally represented by instances of different classes derived from the RequestBase abstract class created by the RequestFactory. As with other factories within the system the RequestFactory is “dynamic”, it can create instances of different concrete classes by name based upon system configuration through the PluginManager. Request classes contain specific methods to deal with updates and changes as well as an instance of a basic Entity class containing all fields and data.

The RequestManager class manages interactions with requests, returning new request instances (created by RequestFactory) when specific requests are fetched. RequestManager also handles security and permissions for requests and teams determining which users can view and update which requests based upon their assignment and the user permissions. The relationship between RequestManager, RequestFactory, RequestBase and the default Request class can be seen in figure 15.

6.3.2 Plugin Management

As discussed in section 4.4.1 (page 22) the main form of extensibility within FreeDESK is through a comprehensive plugin architecture which supports the registration and operation of functionally-specific plugins (such as authentication methods or request classes) as well as providing for generic plugin modules which affect system behaviour in a wider sense. These operations revolve around the PluginManager class which creates instances of active modules, registers functionality and then calls specific methods as and when required. PluginManager also provides a mechanism by which interface components such as menu options or output

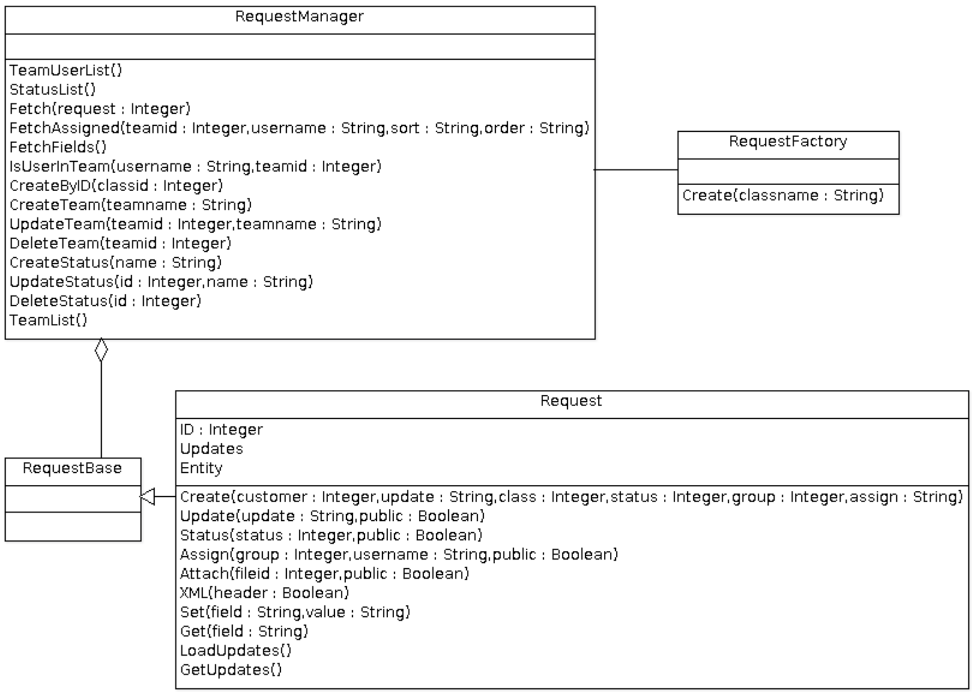

RequestManager
TeamUserList()
StatusList()
Fetch(request : Integer)
FetchAssigned(teamid : Integer,username : String,sort : String,order : String)
FetchFields()
IsUserInTeam(username : String,teamid : Integer)
CreateByID(classid : Integer)
CreateTeam(teamname : String)
UpdateTeam(teamid : Integer,teamname : String)
DeleteTeam(teamid : Integer)
CreateStatus(name : String)
UpdateStatus(id : Integer,name : String)
DeleteStatus(id : Integer)
TeamList()
RequestFactory
Create(classname : String)
RequestBase
Request
ID : Integer
Updates
Entity
Create(customer : Integer,update : String,class : Integer,status : Integer,group : Integer,assign : String)
Update(update : String,public : Boolean)
Status(status : Integer,public : Boolean)
Assign(group : Integer,username : String,public : Boolean)
Attach(fileid : Integer,public : Boolean)
XML(header : Boolean)
Set(field : String,value : String)
Get(field : String)
LoadUpdates()
GetUpdates()

Figure 15: Request Management and Classes UML

pages can be registered (and then displayed when requested from the UI). The overall structure of PluginManager and the associated registration classes are shown in figure 16.

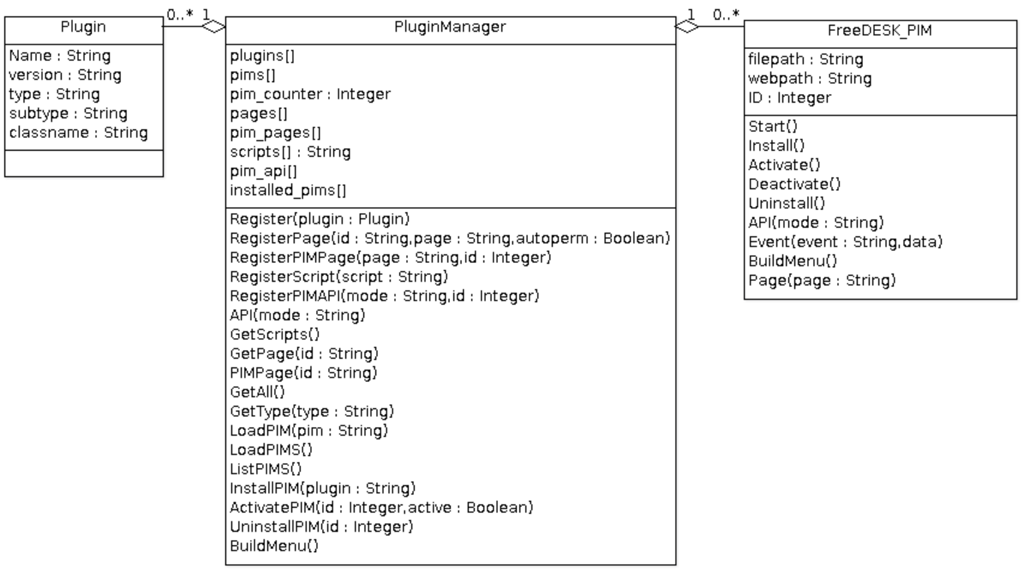


Figure 16: Plugin Management and Modules UML

Functionally-specific plugins are themselves classes derived from the base class relevant to their function, for example AuthBase is the base class for authentication methods. These plugins are then registered by creating a Plugin class with pertinent details which is passed to the PluginManager::Register method. Once registered these concrete classes can then immediately be created by the relevant factory class when that type of concrete implementation is requested.

Plugin modules (PIMs) are derived from the more generic FreeDESK_PIM class and are loaded automatically on system startup if they are set as active (through the system administration interface) and assigned an internal ID value within PluginManager which is provided to the PIM. PIMs then register functionality they provide or call other methods through the PluginManager interface to make changes to system operation. An example of a PIM providing a user interface page accessed through a menu option would therefore be as follows:

```
class SomePlugin extends FreeDESK_PIM
{
  function Start() // on startup
  {
    // Register a page
    $this->DESK->PluginManager->RegisterPIMPage("example",
      $this->ID);
  }

  function BuildMenu() // when UI menu is built
  {
    $mymenu = new MenuItem(); // create
    $mymenu->tag="mymenu";
    $mymenu->display="My␣PIM␣Menu";
    // Submenu item for mymenu
    $mitem = new MenuItem();
    $mitem->tag="mypage";
    $mitem->display="Display␣Example␣Page";
    $mitem->onclick="DESK.loadSubpage('example');";
    // Add to our menu
    $mymenu->submenu[]=$mitem;
    // Register Menu
    $this->DESK->ContextManager->AddMenuItem("mymenu",
      $mymenu);
  }

  function Page($page) // display a page
  {
    if ($page == "example") // example page
    {
      echo "Hello␣World!";
    }
  }
}
```

To further illustrate the use of plugins Appendix D contains a commented example functional plugin. As with other technical aspects of the system the plugin architecture is fully documented with example code in the system documentation (Appendix A).

6.3.3 Data Dictionary

To allow extensibility with regard to the underlying data stored in an SQL database by the system a "dynamic data dictionary" will be used. This will contain a codified representation of the underlying database entity (table) structure including field types and also the relationships between entities. This information, held by the DataDictionary class will consist of a number of DD_Table objects each comprised of a number of DD_Field objects as well as optional DD_Relationship classes defining the relationships. The UML structure of the data dictionary is shown in figure E.

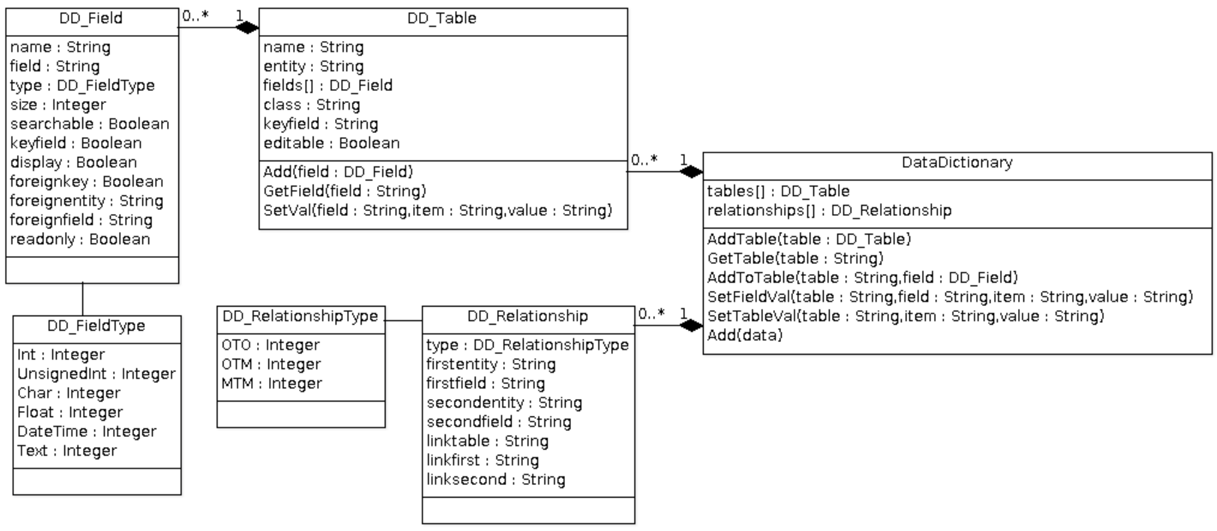


Figure 17: Data Dictionary and Entity Classes UML

Codifying this structure and allowing it to be modified provides powerful support for extensibility as code can easily read and analyse structures as well as dynamically update and change them. Provision of a standardised description of structures also allows items such as search forms and output data to be easily created and formatted in the correct manner for the underlying data. A generic entity search function within FreeDESK allows the lookup, view and modification of any "editable" (flag within DD_Table meaning the entity is directly human editable) entities which are described by the data dictionary.

Although more advanced functionality such as the mapping of many-to-many relationships through a third link entity is not currently included in the implementation plans the options to identify and map such relationships exists for future use (or the use of any plugin components).

6.3.4 Skinning

Section 4.4.3 describes the need for and aim of a skinning system, allowing the decoupling of logic and appearance so providing for the complete customisation of the interface "look and feel".

In the implementation of FreeDESK this functionality will be provided by the Skin class which provides an interface through which Skin items are loaded into data or displayed to the end user. A default skin is defined and shipped with the system providing all required skin items for display. The Skin class can be then configured to use an alternative skin and, when each item is requested, it will be taken from the configured skin and fallback to the item in the default skin if not found.

Items are located in the skin-specific directory of the system (e.g. a skin called "myskin" would be located in freedesk/skin/myskin). Primary skin items are as follows:

Item	**Description**
header	Main full page header including any references to stylesheets or display-related javascript
footer	Main full page footer
min_header	Minimalistic page header (for popup windows) including any references to stylesheets or display-related javascript

min_footer	Minimalistic page footer
login	Login box for credentials
menu	User interface menu display
main_start, main_left_start, main_right_start, main_left_finish, main_right_finish, main_finish	Visual control components for the main screen including the team and user list area (left) and the request list display area (right)

Table 7: Primary Skin Items

6.3.5 HTTP-XML API

The HTTP-XML API is the primary mechanism through which remote communication[5] is made to FreeDESK. All system functionality is provided through this API through the means of an incoming HTTP request (GET or POST) specifying the "mode" of operation (the specific API call) and response being returned in valid XML form for processing by the originator of the call.

The HTTP-XML API will contain built-in functionality for core features but also be extensible through the plugin module interface whereby PIMs can register API modes and then handle them directly doing any actions required and returning their own XML as a response (see section 6.3.2). The full range and calls provided by the API are documented in the system documentation (appendix A) but the main in-built functionality to be provided is listed as follows:

Mode	**Description**
login	Attempt to authenticate a session to the system for given username/password. Return error on failure or session ID on success.
logout	Destroy current session

[5]from a source external to the server or not using native code such as the end-user interface on a remote browser or a third-party application local or external to the server

requests_assigned	Return list of requests assigned to specified team/user
entity_search	Perform a generic entity search for specified entity and search fields (section 6.3.3)
entity_save	Update an entity record
entity_create	Create an entity record
user_edit, create_user, delete_user	Manage system users
request_create, request_update	Create and update requests (include mechanism for status and assignment changes in the update function)
plugin_install, plugin_activate, plugin_deactivate, plugin_uninstall	Control PIMs

Table 8: HTTP-XML API In-Built Functionality

In addition to the modes shown above functionality will be provided to access system administration features for example updating possible request statuses, changing and saving permissions, assigning users to groups and any other general administration tasks performed through the user interface. This functionality will be added during development as each section of the user interface is created.

6.4 Granular Most-Specific Permission Model

To allow for a flexible permission structure where individual functions or actions can be granted or denied to individual users a permission management strategy must be defined.

This strategy will be based on a granular most-specific model in which permissions can be assigned down to individual actions (or even sub-actions). In this model each individual permission is tested for the user session to find the most specific grant or deny setting for that session. At each level permissions can be defined (allow or deny) or undefined in which case the next level of specificity is tested.

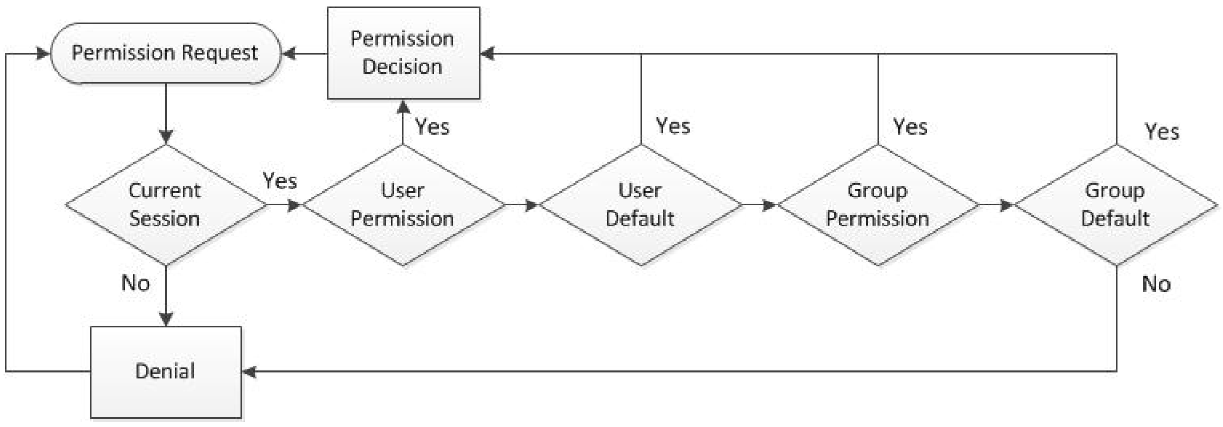


Figure 18: Granular Most-Specific Permission Flowchart

Figure 18 shows a flowchart illustrating how such a model will be implemented. Permissions can be defined and stored for specific users or for a security group which a user can be a member of. Users and groups may also define a default permission to be used if no specific permission has been defined.

In this way when a permission is requested for a user it is first checked if that permission has been defined for the specific user and used if it has. If no specific permission for the user is defined the default permission for the user is checked. If no matching permission, either specific or default, can be found for the user the user's security group is tested, again first for the specific permission and then for the default permission.

With this model the most specifically defined permission is used, a permission set specifically against a user "trumps" one set against the user's group and a specific permission for a level always "trumps" the default for that level.

If no matching permission can be found then the permission to perform the action is denied.

6.5 Client Server Communications

Communication between the end-user web browser (client) and the FreeDESK installation (server) will occur via standard HTTP requests returning pages and also through Asynchronous Javascript requests for XML (AJAX). The basic makeup of pages will initially be created by the server in response to a request from the client and output for display in HTML (with associated in-line or linked CSS and Javascript). Once displayed in the browser most user actions (or automated actions such as refreshes) will then communicate using AJAX to the server and process the results.

For example the standard main page will have its initial structure directly created in HTML but on the initial data load (and on refresh or user input) an AJAX request will be made to the server, returning XML request lists (for the main list display) or HTML to be displayed as a sub-page (where the main list display is hidden until called back into view by the user). This relationship can be seen in figure 12 as section 6.1.1 shows how components can act to generate HTML and also calls are made to the API from the browser.

7 Implementation

Following from the detailed software design (section 6) code implementation was performed in an incremental and iterative manner (section 6.1) and individual sections and modules of software were created for integration. The implementation followed the designs with a few issues identified and leading to design refinement or tool script creation which are detailed in this section.

7.1 Source Code Management

Development of FreeDESK used the Source Code Management (SCM) package git with repository hosting on github.com. This provided a centralised development repository, secure backup of code and a historic change view.

As individual source files are created they must be added to the git repository manually and are then copied to github.com when a code commit is made. As the addition of files is a manual process it is one prone to human error with files being omitted from the repository (and hence not being stored and backed up) or deleted files still included. In the early stages of development it was clear from a manual audit of the repository some key files had been missed for addition.

To resolve this issue a "gitcheck" script[6] was created which compared the local filesystem of the development machine with the file list contents of the git repository within certain parameters (for example ignoring temporary gEdit files and release directories). Any discrepancies found such as missing files (not included in the repository) or orphaned files (in the repository but no longer present on the filesystem) were then clearly displayed for action. Running of this script became a regular part of the daily final commit process and, since implementation, has identified files for inclusion which have then immediately been added and committed to the repository.

[6]Source code available in the main FreeDESK repository and more details along with standalone code at `http://blog.freedesk.purplepixie.org/?p=26`

7.2 Core Components

During development of the core components a number of areas were found for specific state settings that would be best served by use of enumerations (enums) as an alternative to "magic numbers" (examples are the open context mode being an analyst or customer connection). PHP does not provide native support for enums and so a workaround utilising abstract classes with static member variables was identified, tested and then used in the final solution.

For example to create an enum called SomeEnumeration the following class code would be created:

```
// abstract, no direct initialisation
abstract class SomeEnumeration
{
   const ValueA = 1;
   const ValueB = 2;
   const ValueC = 3;
}
```

Once implemented the SomeEnumeration type could be initialised and tested as follows:

```
$myEnum = SomeEnumeration::ValueB;
if (SomeEnumeration::ValueB == $myEnum)
{
   // do some action
}
```

As PHP is a weakly typed language (as with C++) then any native enums would be comparable to integers which is consistent with the operation of the enum workaround.

7.3 Security

Security within the system is assured through session authentication with a granular most-specific security model (section 6.4) ensuring that actions can only be performed by authenticated sessions allowed to perform the action. Each request to FreeDESK, for a page to be displayed or through an XML-HTTP API call, has a session ID (SID) passed as part of the request which is then validated against current sessions.

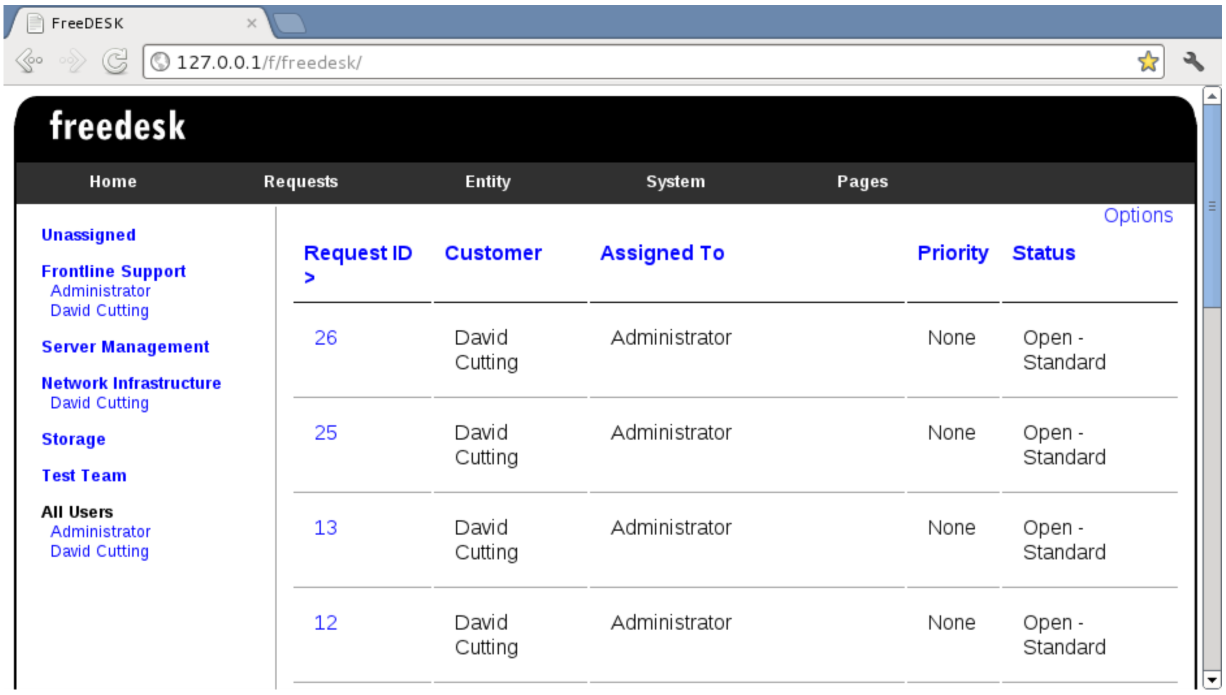
The analyst portal has an integrated but usually invisible login form. In the event that no SID exists (or if the system responds to a request with an error message indicating the SID is invalid, for example if the session has been idle for long enough for the session to time out), the login form is displayed. Once credentials are entered a session is attempted to be authenticated through the XML-HTTP API. If the session is successfully authenticated then the client-side SID is updated and the original request can proceed or an error message is displayed if authentication fails.

As a final protection against unauthorised access all database access is escaped to prevent an attack called SQL injection. Without this protection a validated session (or an as-yet unauthenticated login request) could be formatted in such a way as to allow for arbitrary SQL execution. The SQL connection component provides methods to do escaping of both textual and numeric data which ensures correct encapsulation meaning that a query containing an SQL injection attempt would most likely just fail and certainly not allow for arbitrary execution.

Although not implemented in the current version support is included within FreeDESK for a third type of authenticated session (currently analyst and customer are implemented), the API mode. In API mode it is forseen that any call can be directly made providing a valid API key (an alphanumeric string configured on a per-API-user basis) which avoids the need to always open an authenticated (username and password) session, allowing a single HTTP call to be made rather than two (the initial authentication to get the SID and then the action call using the SID).

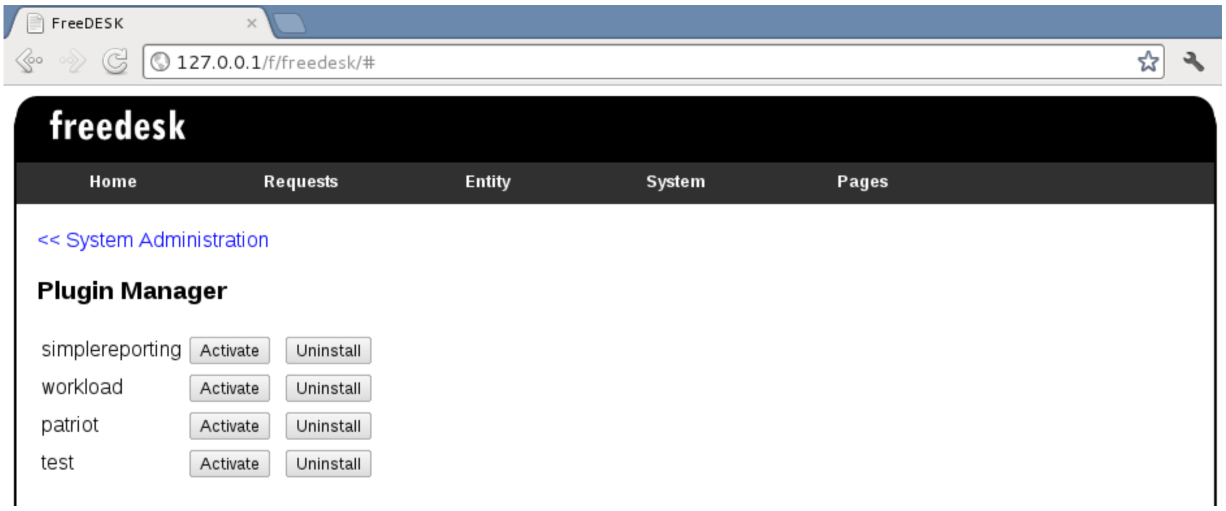
7.4 Plugin Framework

The plugin framework was developed as specified (section 4.4.1) and designed (section 6.3.2) with each of the areas of variability supported through the registration of concrete classes with associated dynamic factories to create the needed instances and additional generic support for pages and modules. The designed mechanism allowed plugin modules to register areas of specific or generic functionality which then could be used by the system. A simple example of changes made by plugins can be seen in figure 19 showing the analyst portal in "default" mode, with plugins being activated (figure 20) resulting in a change to the main menu bar (figure 21) providing access to reporting functions provided by a plugin.



Request ID >	Customer	Assigned To	Priority	Status
26	David Cutting	Administrator	None	Open - Standard
25	David Cutting	Administrator	None	Open - Standard
13	David Cutting	Administrator	None	Open - Standard
12	David Cutting	Administrator	None	Open - Standard

Figure 19: The Main Analyst Portal Interface



Plugin Manager

simplereporting	Activate	Uninstall
workload	Activate	Uninstall
patriot	Activate	Uninstall
test	Activate	Uninstall

Figure 20: FreeDESK Plugin Management Interface

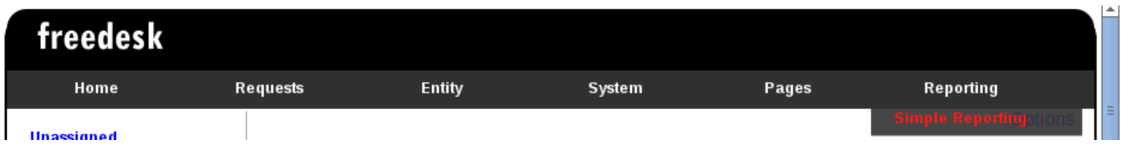


Figure 21: Analyst Portal Menu Bar with Reporting Plugin

As the core XML-HTTP API calls were implemented within the *api.php* script an additional "catch all" was implemented whereby, if no specifically programmed call was found, the call was passed to the plugin manager to call any plugin module that had registered that call.

7.5 Client-Side Controls and View

The client-side analyst portal was implemented in HTML 5 using the designed framework, with initial HTML framework pages being generated server-side in PHP and then modified as required by client-side Javascript.

Development proceeded smoothly with just one major issue found when using the XML-HTTP API from within the Internet Explorer browser. Once a call had been made through the AJAX client-side module (ServerRequest) the response was received but could not be parsed or used as XML. This was eventually traced to a MIME-type issue, although the version of Internet Explorer was using and supported the XMLHttpRequest object (unlike previous versions of Internet Explorer which are supported through ActiveX objects) it did not support an overrideMimeType setting for the object, though this failed silently. To overcome this problem a check was implemented in ServerRequest only setting the MIME type if the override option existed and an explicit header was put into the server-side XML-HTTP API script to set the MIME type to text/xml.

7.6 Customer and Mobile Interfaces

To meet two requirements (customer portal and mobile-accessible analyst portal) identified as "could have" in the MoSCoW analysis (section 4.2) a customer and specifically mobile-accessible analyst portal were developed. These were built around the core system and used the same interfaces as the main analyst interface, just providing a limited (in scope and operation) and customised (in HTML and screen format) interface.

Although it is possible through use of CSS media tags to use the exact same interface for desktop and mobile browsing given the relatively high complexity of the analyst portal along with no clear need to provide all but core functionality to the mobile browser the approach of a separate skin set of HTML hooking to the core interface was chosen. Mobile browser detection was used with display defaulting to the mobile interface for login through the session could be switched to the full desktop version (and vice-versa) at any point.

The customer interface was designed to make less use of Javascript or any form of “exotic” HTML/CSS to allow use from a wider set of browsers including older versions of Internet Explorer, facilitating clear and easy access to pre-existing requests for the customer along with the option to log new requests.

8 Release Pipeline

For FreeDESK to be usable by end administrators, without them having to manually download the source code then create the database structure and configuration by hand, release packages must be built and made available. These will contain the current most stable version of the code along with scripts to facilitate installation or upgrade, including the database structure and system configuration options.

This process takes place in three stages, on the Linux development platform for the software where the release is built and packaged, the Linux web server for FreeDESK where the release is uploaded to, recorded and made available for download and finally on the end-users choice of installation platform where the install scripts are run.

8.1 Build Process and Tools

A release consists of the source code for the system along with install and setup scripts built into a single archive by tool scripts. The process is as follows:

Stage	**Description**
Final commit (optional)	The current version of source code committed to the SCM before build, this step will also refresh the SQL scripts to the current database schema[7]
Directory created	A version specific directory for the release build is created
Source code and SQL copied	The current source code and SQL scripts are copied to the build directory
Configuration and development files removed	The development configuration and any other development files (such as temp files from gEdit) are removed
Setup script renamed	The installation script is renamed from it's development name to be automatically run with the system startup

[7]SQL scripts are created both through the use of mysqldump creating a set of SQL to create a fresh schema from scratch and also by myrug a GPL tool by the same author to create SQL instructions which will update an existing schema of any previous version to the current revision

Generate tarball (optional)	The release directory is created into a tar archive which is then gzipped (to create a .tar.gz tarball file)

Table 9: Release Build Process

At this point if the options have been chosen a release tarball of the system (in the form freedesk-x.yy.zz.tar.gz where x, yy and zz are major, minor and patch versions respectively) has been created release for release.

8.2 Release Process

Once the release tarball has been created it must be put online and made available to download through the FreeDESK website. This process is as follows:

Stage	**Description**
Check if release exists	Check that the release version is not already held by the webserver, error and exit if it is
Get change log information	Prompt for information to be included in the change log for the released version and also if it is to be marked as the current/recommended release
Get MD5	Generate an MD5 hash of the tarball
Upload	Upload the release tarball to the server with additional information such as the MD5 hash and change log
Validate	On the server validate the upload against the provided MD5 hash, error and stop if mis-match
Generate ZIP	On the server, unpack the tarball and create a ZIP file of the release also
Move for download	On the server move the tarball and new ZIP file to the downloads directory

Record release	On the server record the release into the database which will make it appear in version lists and become automatically available for download, update the current version information if required (if the version is marked as current)

Table 10: Release Process

8.3 Install Process

Once a release has been downloaded and the contents of the archive copied onto a webserver the user will navigate their browser to the directory. After an install (or following an upgrade) the file setup.php will exist. If this is found the user is immediately redirected to it.

The setup file runs the installation or upgrade process for FreeDESK.

First the install process checks for the existence of core system configuration (config/Config.php) which indicates if the install is fresh or an upgrade. If an upgrade the user is asked if they wish to rewrite their configuration or just move on, a fresh install will create a configuration file.

Basic configuration values are entered and from these a basic configuration file created (if access allows this can be written straight to the config directory otherwise the contents are displayed for the user to manually copy and paste into the appropriate file).

SQL files are then read and executed against the underlying database either to create a fresh schema or to update an existing one. As the system provides for the addition of table prefixes to an installation all table names are prefixed with %%$%% in SQL files and this is replaced with the configured prefix as the files are read prior to execution against the database.

For fresh installations the user is now asked to input a password for the admin user which will be hashed and stored in the database.

The user is then instructed to delete the setup.php file as the system will not function with it in place as a security measure to avoid unauthorised re-installation of the system (if the file is in place any attempt to access FreeDESK will be automatically redirected to the setup file).

8.4 Extension Development Process

In order for an extension developer to incorporate additional functionality in FreeDESK they need to create PHP code which provides the functionality and registers with the provided plugin interface, as documented in the technical system documentation (see Appendix A). To use the extension within the system it must be placed in a correctly-named structure within the plugin folder structure at which point it can be first installed and then activated through the analyst portal system administration features.

This process is shown in detail with an example plugin module in Appendix D.

9 Testing

Testing was carried out iteratively during development as part of each development stage (often multiple times per stage). Testing was performed in two distinct ways through unit and functional testing.

9.1 Unit Testing

To facilitate unit testing a unit test framework (UTF) was developed and then individual unit tests were incorporated into the UTF as additional functionality was added to FreeDESK. These unit tests were predominantly black-box tests (using standard external interfaces and just considering input and validating output or results) with some white-box tests (using internal functionality or written to test the internal function of specific components) implemented when required.

The UTF was managed and launched through the *unittest.php* script which ran the tests in sequence keeping a tally of tests run and passed along with any errors or warnings generated. The script also allowed very quick identification of any PHP syntax errors contained in any included code as the script would immediately fail with a textual error if any syntax error was found during system startup.

A specific unit test creates an instance of the DESKTest class, executes the test, sets the result in the DESKTest instance and then records this result in the UTF results class (DESKTester, instanced as $desktest). An example unit test using the UTF is as follows:

```
// Set the type of unit test
$type="sometype";
// Set an incremental ID for the type
$id=0;
// Create a DESKTest
$t=new DESKTest($type, $id++, "Description");
// Now perform the test...
// And test the result (result)
```

```
if ($result)
    $t->passed=true;
else
    $t->passed=false;
// Record the result into the UTF
$desktest->Add($t);
```

9.2 Functional Testing

As features were implemented within FreeDESK these were then tested within web browsers. To ensure maximum cross-browser (and cross-platform) support a number of test environments were used including:

- Mozilla Firefox (Windows and Linux)
- Google Chrome (Windows and Linux)
- Internet Explorer (Windows)
- Safari (MacOS X)

At each stage of integration of new features into the overall system not only were the newly added features tested, with errors feeding back into the development cycle, but pre-existing core features were also re-assured in a limited form of integration testing. Each major completion of features and prior to any build and release all the primary features of the system were tested for functionality in all the above browsers.

10 Evaluation

To evaluate the FreeDESK system two approaches were taken, consideration of the system delivered against the goals (section 10.1) and the results of an evaluation questionnaire made available on the FreeDESK website to anyone using the system (section 10.2).

10.1 Completion of Goals

The overarching goals of the system were identified in the MoSCoW analysis (section 4.2) quantifying what features the system *must have*, *should have* and *must have*. These goals can now be reviewed against what has been implemented in the system to evaluate how well the identified goals have been delivered.

Must Have

- Request tracking with unique reference: Yes, requests assigned a unique reference and can be tracked within the system
- Prioritisation (logging and view in relative priority): Yes, prioritisation of requests and view sorting by priority
- Customisation options (configuration): Yes, all system features are configurable through the analyst portal system administration interface
- Extensibility options (plugins): Yes, extensibility is supported for key areas of variability and generic extensions

Should Have

- Performance monitoring: Yes, requests can be monitored for performance against defined service levels and targets
- Web-based system: Yes, interface is fully web-based

- Screen update mechanism to show new updates through the analyst portal (section 4.3): Partial, updates to the main request list are provided through a polling mechanism
- Update mechanism to update customer (email): Yes, emails can be generated and sent within the system when requests are updated using set auto-completed templates

Could Have

- Customer portal (self-service): Yes, a customer portal is provided allowing review, update and creation of customer requests
- Mobile-accessible analyst portal: Yes, a mobile-accessible version of the analyst portal is provided allowing for key functionality (request management)
- Integration options with external systems: Yes, a featured and documented XML-HTTP API is provided

10.2 Questionnaire Evaluation

In August 2012 a survey was carried out to obtain views and comments on experiences when using FreeDESK. People using the system were invited to fill in an evaluation questionnaire on the main FreeDESK website. The questionnaire, which is contained along with the full results for options selected in Appendix E, was web-based and given approval by the UEA School of Computing Science Ethics Officer. Once completed all individual session data was deleted and only summary data remains. There were twenty-seven responses.

Divided into five sections questions were intended to gain a profile of the respondents, their overall opinion of the FreeDESK system, which features they have used, their opinion of the extensibility features (if indicated they had used these) and for any other information or suggestions they wished to provide. Of twenty-seven respondents to the questionnaire six indicated they has used extensibility features and so were asked questions about these.

Figure 22 shows the profile of respondents which indicates a wide variety in the types of sizes of organizations with a larger proportion coming from smaller organizations with over 50% from

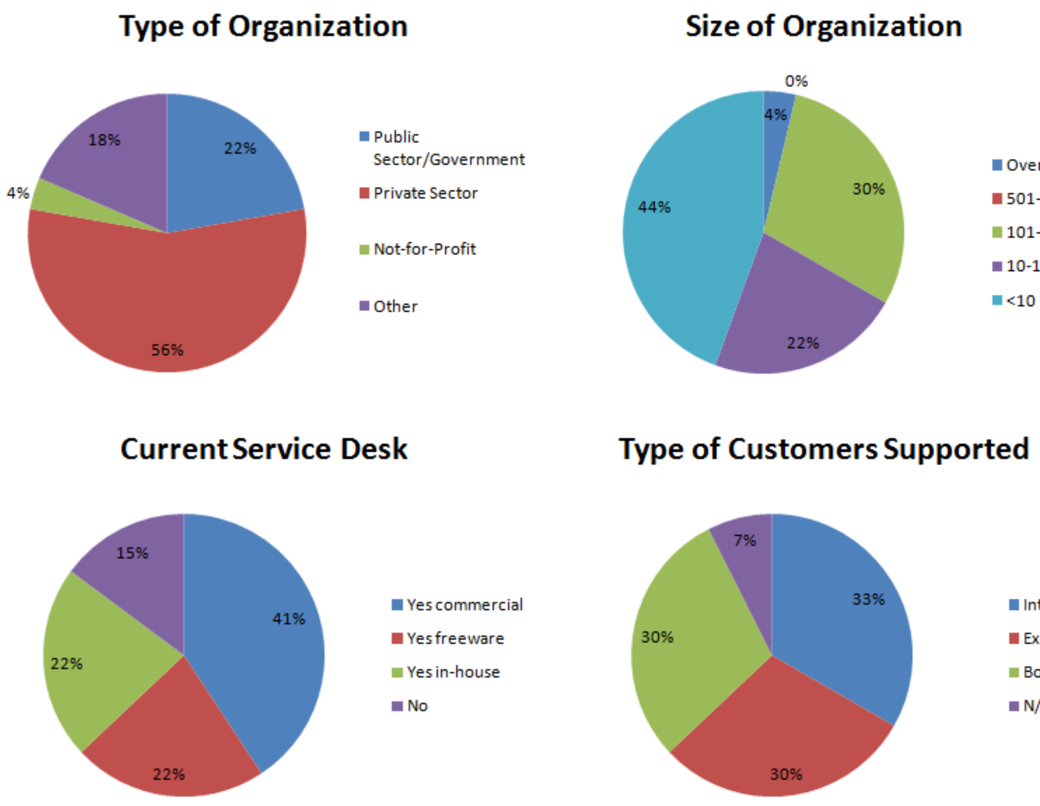


Figure 22: Profile of respondents to the questionnaire

the private sector. The majority of users already have a service desk solution (85.19%, n=23) of some form, of which the most common was a commercial solution (40.74% of all respondents, 47.83% of respondents with an existing service desk, n=11). Of customers supported 59.25% (n=16) indicated they supported purely external or a mix of both internal and external customers with a third of respondents (33.33%, n=9) supporting purely internal customers.

88.89% (n=24) of respondents indicated they agreed with the statement that "FreeDESK is a useful system" with 85.19% (n=23) and 85.19% (n=23) agreeing that they would use FreeDESK in the future and recommend FreeDESK respectively. Documentation has the lowest approval score with 59.25% (n=16) agreeing that the "documentation and information provided about FreeDESK is good" however owing to time constraints the evaluation was proceeding while the documentation was not complete, an unfortunate situation that may explain the relatively low approval of documentation.

Availability and installation ("FreeDESK was easy to download and install") gained 100% (n=27) approval. A confidence statement ("I feel confident in the FreeDESK system") had 74.07% (n=30) approval and 66.67% (n=18) agreed that "the user interface is clear and easy to use". To gauge the overall effectiveness of the system respondents were asked if they agreed with the statement "FreeDESK provides the features I need in a Service Desk", a statement that 77.78% (n=21) agreed with.

These scores are very positive with the lowest approval for documentation, something which unfortunately was not complete at the point of evaluation. The respondents cannot however be truly seen as a fully representative random sample of people who have downloaded and used the system. As the system is provided free, with no entry cost or "buy-in", it can be argued that someone who did download the system but then did not like or use it would be unlikely to take the time to fill in an evaluation questionnaire for the system.

With respect to features used within the system the large majority of users had used the request creation (96.30%, n=26) and update (74.07%, n=20) features with only a minority using the extensibility features such as the development/plugin framework (18.52%, n=27), API (3.70%, n=1) and event handling framework (7.41%, n=2). This distribution of feature use can be expected as it was always intended that many people would simply download and use the system as a standalone Service Desk solution. Thus a much smaller proportion of these people going on to use the extensibility features to develop new functionality and it is also very early in the lifetime of FreeDESK with no third-party plugin examples or documentation and so it is currently not possible to properly quantify cost and ease of development.

Of the six respondents who completed the extensibility section the response to the development options was positive with 83.33% (n=5) indicating they agreed that "the provided interfaces are sufficient for my needs", "the development framework is well documented" (technical documentation and in-code documentation was mostly complete by the evaluation unlike general system documentation)and "FreeDESK is developed in an open and easy extensible manner". The ease of development is also ranked positively with 66.67% (n=4) agreeing that "developing extensibility in FreeDESK is easy".

Only 50% (n=3) of respondents who are undertaking development agreed that they "hope to incorporate extensions into the general release for others to use", presumably developing extensions for purely their own use or independent release (something allowed and encouraged by FreeDESK licensing).

11 Conclusion

The aim of this project was to deliver a free open-source extensible service desk which was broken down into an initial literature review (section 2), high-level overall system design including requirements analysis (section 4), an investigation into update mechanisms (section 5), detailed software design (section 6), implementation in an iterative fashion (section 7) and finally an evaluation (section 10).

Service desk systems are used in a wide variety of environments, tracking and managing "requests for service" from customers. There are already in existence a wide variety of such systems but it was identified that no fully-extensible system is available for free and under an open-source licence. To make a system extensible numerous approaches, techniques and architectures are commonly used in software development which were identified and analysed before a solution employing design patterns within an Object-oriented Application Framework utilizing a concept from Software Product Lines was selected for this project.

Three common use cases were identified and developed to inform the high-level system design, the basis of the system revolving around the idea of requests and the assignment, management and update of these. As a multi-user web based system, one in which changes can be made simultaneously to the same common data by different individuals, some method of interface update must be provided allowing these changes to be reflected. To investigate a possible solution offering low-latency updates an experiment was designed and performed to consider long-held HTTP polling as an update method. Though this experiment proved the feasibility of this technique with the system architecture lack of resources allowing load analysis and the inherent complexity of the method led to a decision to use a poll-based refresh technique with the possibility of future incorporation of the long-held technique.

To provide for future extensibility areas of variability within the system were identified (where concrete functionality can be interchanged), along with a "skinning" provision for the interface, an XML-HTTP API for third-party external accessibility and a modular extensibility architecture allowing for additional functionality to be added utilizing the existing interfaces.

Implementation occurred in an iterative and incremental manner, individual features being designed and then implemented in steps, building an overall functional system in a series of stages. At each stage of the development functional test cases were also implemented allowing ongoing testing of the new functionality and also integration testing (with previous features also being tested to ensure they continued to operate correctly). A website was built with a release pipeline allowing the code, along with supporting files such as SQL, to be built into a single archive and made available for general download. To support the system a large documentation archive was created using a "wiki" web-based system, including both technical and user-guide documentation.

The system as delivered meets all the identified goals providing a usable Service Desk system incorporating extensible options through an extensive plugin mechanism and API support. In this way it can be seen as a success, providing all the required functionality identified and defined in the design.

Once a release was available and being downloaded people were invited on the website to complete an evaluation questionnaire. 27 respondents filled out this questionnaire, the results of which were very positive. For example 85.19% of respondents agreed the system was useful with over 80% of respondents agreeing that they would use the system in the future and recommend it to others. Of the 27 respondents only six had made use of the extensibility features but again felt positively with 83.33% agreeing the extensibility features met their needs. Although this was probably not a fully representative sample of people who had downloaded the system it is nonetheless a strongly positive finding.

One aspect of the evaluation scoring a lower mark was the documentation although this is most likely a result of the documentation being incomplete at the time of evaluation owing to time constraints though even in this situation 59.25% of respondents agreeing the documentation was good.

In conclusion it can be seen that the system as specified has been delivered, meeting all the design goals and providing a complete Service Desk system with a high ability for extensibility. Evaluation by respondents has been very positive with a vast majority finding the system useful, something they would use in the future, and something they would recommend to others. It can therefore be said that the implementation of the system has been a success, providing a useful and flexible solution which will hopefully continue to be used by greater numbers for years to come.

References

Al-Qahtani, S. S., Pietrzynski, P., Guzman, L. F., Arif, R., and Tevoedjre, A. (2010). Comparing selected criteria of programming languages java, php, c++, perl, haskell, aspectj, ruby, cobol, bash scripts and scheme revision 1.0 - a team cplgroup comp6411-s10 term report. *CoRR*, abs/1008.3434.

Ampatzoglou, A., Kritikos, A., Kakarontzas, G., and Stamelos, I. (2011). An empirical investigation on the reusability of design patterns and software packages. *Journal of Systems and Software*, 84(12):2265–2283.

Amsden, J. (2003). First eclipse plugin. `http://www.eclipse.org/articles/Article-Your-First-Plug-in/YourFirstPlugin.html`. [Online; accessed July 2012].

Ashworth, J., Lacy, L., Tanger-Brown, L., and Rhodes, C., editors (2007). *Proceedings of the 35th Annual ACM SIGUCCS Conference on User Services 2007, Orlando, Florida, USA, October 7-10, 2007*. ACM.

Black, B. and Larsson, E. A. (2004). A case study: implementing supportworks professional helpdesk at drew university. In Whiting et al. (2004), pages 26–29.

Bozdag, E., Mesbah, A., and Van Deursen, A. (2007). A comparison of push and pull techniques for ajax. In *Web Site Evolution, 2007. WSE 2007. 9th IEEE International Workshop on*, pages 15–22. IEEE.

Chan, L. (2011). Wordpress plugin framework. `http://code.google.com/p/wordpress-plugin-framework/`. [Online; accessed March 2012].

Chatley, R., Eisenbach, S., Kramer, J., Magee, J., and Uchitel, S. (2004a). Predictable dynamic plugin systems. *Fundamental Approaches to Software Engineering*, pages 129–143.

Chatley, R., Eisenbach, S., and Magee, J. (2004b). Magicbeans: A platform for deploying plugin components. *Component Deployment*, pages 97–112.

Conlon, M. J. (2007). Overhaul your helpdesk ticketing system. In Ashworth et al. (2007), pages 37–40.

Dooling, C. and Bertrand, R. (2007). It professional development and helpdesk support on a shoestring budget or renewable resources. In Ashworth et al. (2007), pages 60–64.

Dwarampudi, V., Dhillon, S. S., Shah, J., Sebastian, N. J., and Kanigicharla, N. (2010). Comparative study of the pros and cons of programming languages java, scala, c++, haskell, vb .net, aspectj, perl, ruby, php & scheme - a team 11 comp6411-s10 term report. *CoRR*, abs/1008.3431.

Fayad, M. and Schmidt, D. (1997). Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional.

Gorton, I., Liu, Y., and Trivedi, N. (2008). An extensible and lightweight architecture for adaptive server applications. *Softw., Pract. Exper.*, 38(8):853–883.

Keller, A. and Midboe, T. (2010). Implementing a service desk: A practitioner's perspective. In *NOMS*, pages 685–696. IEEE.

Leitner, A. and Kreiner, C. (2011). Software product lines - an agile success factor? In O'Connor, R., Pries-Heje, J., and Messnarz, R., editors, *EuroSPI*, volume 172 of *Communications in Computer and Information Science*, pages 203–214. Springer.

Letarte, D., Gauthier, F., and Merlo, E. (2011). Security model evolution of php web applications. In *ICST*, pages 289–298. IEEE Computer Society.

Loreto, S., Saint-Andre, P., Salsano, S., and Wilkins, G. (2011). Rfc 6202: Known issues and best practices for the use of long polling and streaming in bidrectional http. `http://www.ietf.org/rfc/rfc6202.txt`. [Online; accessed July 2012].

MediaWiki (2012). Developing extensions. `http://www.mediawiki.org/wiki/Manual:Developing_extensions`. [Online; accessed June 2012].

Oreizy, P., Medvidovic, N., and Taylor, R. (1998). Architecture-based runtime software evolution. In *Software Engineering, 1998. Proceedings of the 1998 International Conference on*, pages 177 –186.

phpBB (2011). Introduction to writing mods. `http://www.phpbb.com/mods/author-introduction/`. [Online; accessed March 2012].

Robles, S., Fernández, J. L., Fortier, A., Rossi, G., and Gordillo, S. E. (2012). Improving the model view controller paradigm in the web. *Int. J. Web Eng. Technol.*, 7(1):22–44.

Russell, A. (2006). Comet: Low latency data for browsers. *alex. dojotoolkit. org*.

Sadalkar, K., Mohandas, R., and Pais, A. R. (2011). Model based hybrid approach to prevent sql injection attacks in php. In Joye, M., Mukhopadhyay, D., and Tunstall, M., editors, *InfoSecHiComNet*, volume 7011 of *Lecture Notes in Computer Science*, pages 3–15. Springer.

Sadjadi, S., McKinley, P., Cheng, B., and Stirewalt, R. (2004). Trap/j: Transparent generation of adaptable java programs. *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, pages 1243–1261.

Sanders, W. B. (2010). Learning oop with weakly typed web programming languages: adding concrete strategies to a php strategy design pattern. In Cook, W. R., Clarke, S., and Rinard, M. C., editors, *SPLASH/OOPSLA Companion*, pages 189–192. ACM.

Sinnett, C. J. and Barr, T. (2004a). Building a champagne helpdesk on a beer budget. In Whiting et al. (2004), pages 351–353.

Sinnett, C. J. and Barr, T. (2004b). Osu helpdesk: a cost-effective helpdesk solution for everyone. In Whiting et al. (2004), pages 209–216.

Szyperski, C., Gruntz, D., and Murer, S. (2002). *Component software: beyond object-oriented programming*. Addison-Wesley Professional.

ter Beek, M. H., Muccini, H., and Pelliccione, P. (2012). Guaranteeing correct evolution of software product lines. *ERCIM News*, 2012(88).

Whiting, J. S., Ashworth, J., and Mateik, D., editors (2004). *Proceedings of the 32nd Annual ACM SIGUCCS Conference on User Services 2004, Baltimore, MD, USA, October 10-13, 2004*. ACM.

Appendices

A Documentation

Full system documentation including installation and user guides as well as technical documentation is available online in the FreeDESK wiki.

The URL for the wiki is: `http://wiki.freedesk.purplepixie.org/`

B System Source Code

The FreeDESK software is licensed under the GNU General Public Licence version 3 (or later if a user so chooses) and as such all the source code is freely available online to view or download.

At the time of submission the current version of FreeDESK is 0.01.0 and this version should be used as the reference version in assessment of this work (as later versions may be released after submission but before final assessment).

Source Code Repository

The full system source code including the release, unit testing and utility scripts is available on the GitHub repository. Changes made include date of submission so can be used to view only code completed prior to the submission date (23rd August 2012).

The URL to browse the code repository is: `https://github.com/purplepixie/FreeDESK`

Release Versions

Release versions of the software can be downloaded directly from or browsed online at the FreeDESK website.

The URL to download is: `http://freedesk.purplepixie.org/changelog.php` from where individual versions can be selected.

To browse source code online by release version the URL is: `http://freedesk.purplepixie.org/codexplore/`

C Use Case Descriptions

Use Case	**Request Logging**
CHARACTERISTIC INFORMATION	
Goal in Context	Request successfully logged and assigned identifier
Level	Primary Task
Success End Condition	Request logged
Failure End Condition	Request logging failed
Primary Actor(s)	Call logger (analyst) or customer
Trigger	Customer wishes request raised
MAIN SUCCESS SCENARIO	
1. Customer contacts service desk call logger or uses self-service 2. Customer details established (automatically on self-service) 3. Request details recorded 4. Request raised	
FAILURE PATHS	
1. Software failure (error in syntax or logic causes request to be lost) 2. Database connectivity or write failure (request not recorded or only partially) 3. Client/Server connectivity failure (request not recorded)	
RELATED INFORMATION	
Frequency	Often (depending on Service Desk load)
Channel to Primary Actor	Duplex, web interface
Secondary Actors	Call logger, System
Channel to Secondary Actors	Telephone or email, system through interface

Use Case	**Request Assignment**
CHARACTERISTIC INFORMATION	
Goal in Context	Request successfully assigned as specified
Level	Primary Task
Success End Condition	Request assigned
Failure End Condition	Request assignment failed
Primary Actor(s)	Analyst
Trigger	Requests requires action by someone else
MAIN SUCCESS SCENARIO	
1. Request is opened for display 2. Assignment option chosen from those available 3. Request assigned	
FAILURE PATHS	
1. Software failure (error in syntax or logic assignment change to be lost) 2. Database connectivity or write failure (assignment not recorded or partial) 3. Client/Server connectivity failure (assignment not recorded)	
RELATED INFORMATION	
Frequency	Often (depending on Service Desk load)
Channel to Primary Actor	Duplex, web interface
Secondary Actors	System
Channel to Secondary Actors	System through interface

Use Case	**Request Update**
CHARACTERISTIC INFORMATION	
Goal in Context	Request updated
Level	Primary Task
Success End Condition	Request updated
Failure End Condition	Request updated failed to record
Primary Actor(s)	Analyst or customer
Trigger	Update to request (or request status such as closed)
MAIN SUCCESS SCENARIO	
1. Request opened (in analyst or customer portal) 2. Update entered (textual data) along with any additional changes such as status 3. Update recorded 4. Update events called	
FAILURE PATHS	
1. Software failure (error in syntax or logic causes changes to be lost) 2. Database connectivity or write failure (changes not recorded or partial) 3. Client/Server connectivity failure (changes not recorded)	
RELATED INFORMATION	
Frequency	Often (depending on Service Desk load)
Channel to Primary Actor	Duplex, web interface
Secondary Actors	System, Plugins
Channel to Secondary Actors	System through interface, Plugins through event framework

D Full Plugin Example

To demonstrate some of the plugin functionality in more detail this Appendix will show a fully working plugin with commented code. The plugin will provide a "workload" report with which an overview can be gained of the number (and priority) of open requests assigned by team and individual analyst.

The plugin will register a permission for all its functionality (*workload*), a page for display (*workload_page*) which will show summary information with detail options and an API mode (*workload_api*) providing requested details to the page. It will consist of three files, the main PHP code (*workload.php*), a brief CSS for visual elements (*workload.css*) and a Javascript file (*workload.js*) which will call the API for details and display the results on the page.

workload.php

```
<?php
// A new class for the PIM extending FreeDESK_PIM,
// named the same as the directory and filename
class workload extends FreeDESK_PIM
{
// The startup method - register functionalityh
function Start()
{
  // Register ourselves as a plugin
  $this->DESK->PluginManager->Register(new Plugin(
    "Workload", "0.01", "Reporting" ));
  // Register a page which will be displayed
  $this->DESK->PluginManager->RegisterPIMPage("workload_page",
    $this->ID);
  // Register an API call for details
  $this->DESK->PluginManager->RegisterPIMAPI("workload_api",
    $this->ID);
    // Add a permission for this action
    $this->DESK->PermissionManager->Register("workload", false);
    /* Register a JS script (workload.js) in our directory
       for inclusion.
    Note the file can be called anything we like or even not
    in the directory (unlike this PHP file) */
    $jspath = $this->webpath . "workload.js";
    $this->DESK->PluginManager->RegisterScript($jspath);
    // Register a CSS file (workload.css)
    $csspath = $this->webpath . "workload.css";
    $this->DESK->PluginManager->RegisterCSS($csspath);
  }
  // The method to add menu items
  function BuildMenu()
  {
    // Check if the permission is allowed or don't show the menu
    if ($this->DESK->ContextManager->Permission("workload"))
    {
      // Check if the reporting menu already exists, add if not
      $currentItems = $this->DESK->ContextManager->MenuItems();
      if (!isset($currentItems['reporting']))
      {
        $repmenu = new MenuItem();
        $repmenu->tag = "reporting";
        $repmenu->display = "Reporting";
        $this->DESK->ContextManager->AddMenuItem
          ("reporting", $repmenu);
      }

      // Built the menu item for this plugin
      $sReport = new MenuItem();
      $sReport->tag="workload";
      $sReport->display="Workload";
      // Set the action for the click
      // (show the page 'workload' registered above)
      $sReport->onclick = "DESK.loadSubpage('workload_page');";
      // Add the menu option to the reporting menu
      $this->DESK->ContextManager->AddMenuItem
        ("reporting",$sReport);
    }
  }
  // Handle page requests (specifically the workload page)
  function Page($page)
  {
    if ($page == "workload_page" // page is the workload page
           && // and permission for this page
            $this->DESK->ContextManager->Permission("workload"))
    {
      // Page title
      echo "<h3>Workload</h3>\n";
      // Team and user list
      $teamuser = $this->DESK->RequestManager->TeamUserList();
      // Iterate through this list and display workload info
      echo "<table>\n";
      foreach($teamuser as $teamid => $teamdata)
      {
        // Counter for the team
        $teamcount=0;
        echo "<tr>\n";
        echo "<td>\n";
        echo "<b>".$teamdata['name']."</b>\n";
        echo "</td>\n";
        echo "<td><b>\n";
        // Get the requests assigned to just the team
        $requests =
          $this->DESK->RequestManager->FetchAssigned($teamid, "");
        $reqcount = sizeof($requests);
        echo $reqcount;
        // Increment team counter
        $teamcount += $reqcount;
        echo "</b></td>";
        // Detail link for the team
        $js="Workload.Detail(".$teamid.",'');";
        echo "<td>\n";
        echo "<a␣href=\"#\"␣onclick=\"".$js."\">Detail</a>\n";
        echo "</td>";
        echo "</tr>";
        // Detail row for the team
        echo "<tr><td␣colspan=\"3\"
␣␣␣␣␣␣␣␣id=\"detail_".$teamid."_\"></td></tr>\n";
        // Iterate through the users in the team
        foreach($teamdata['items'] as $user => $userdata)
        {
          echo "<tr>\n";
          echo "<td>\n";
          echo $userdata['realname'];
          echo "</td>\n";
          echo "<td>\n";
          // Requests for this user and team
          $requests = $this->DESK->RequestManager->
            FetchAssigned($teamid, $user);
          $reqcount = sizeof($requests);
          echo $reqcount;
          // Increment team counter
          $teamcount+=$reqcount;
          echo "</td>\n";
        // Detail link for the team and user
        $js="Workload.Detail(".$teamid.",'".$user."');";
        echo "<td>\n";
        echo "<a␣href=\"#\"␣onclick=\"".$js."\">Detail</a>\n";
        echo "</td>";
        echo "</tr>\n";
        // Detail row for the user
        echo "<tr><td␣colspan=\"3\"
␣␣␣␣␣␣␣␣␣␣id=\"detail_".$teamid."_".$user."\">";
        echo "</td></tr>\n";
      }
    // The team total
    echo "<tr><td><b>Total</b></td>";
    echo "<td><b>".$teamcount."</b></td></tr>\n";
    // A spacer
    echo "<tr><td␣colspan=\"3\">";
    echo "<hr␣class=\"workload\"></td></tr>\n";
    }
    echo "</table>\n";
  }
}
// API Handler
function API($mode)
{
  if ($mode == "workload_api" // Correct API Mode
    && // and permission for workload
    $this->DESK->ContextManager->Permission("workload"))
  {
    // Get the team and user requested
    $teamid =
      isset($_REQUEST['teamid']) ? $_REQUEST['teamid'] : 0;
    $username =
  isset($_REQUEST['username']) ?
  $_REQUEST['username'] : "";
// Get the assigned requests
$reqs = $this->DESK->RequestManager->FetchAssigned
  ($teamid,$username);
// Get the list of priorities
$pris = $this->DESK->RequestManager->GetPriorityList();
// Our output array with unknown default
$out = array(0=>array("name"=>"Unknown","total"=>0));
// Iterate through the requests
foreach($reqs as $req)
{
  // Get the priority
  $pri = $req->Get("priority");
  // Check if exists and add to totals
  if (isset($pris[$pri])) // valid
  {
    if (isset($out[$pri]))
      $out[$pri]['total']++;
    else
      $out[$pri]=array(
        "name"=>$pris[$pri]['priorityname'],
        "total"=>1 );
  }
  else
    $out[0]["total"]++;
}
// Build the HTML
$html = "";
// For each detail line
foreach($out as $line)
{
        $html.=$line["name"].":␣".$line["total"]."<br␣/>";
      }
      // Create the XML output
      // XML creation object
      $xml = new xmlCreate();
      // Add a char element with CDATA encoding
      $xml->charElement(
        "data",
        $html,
        0,
        false,
        true );
      // Output the XML with the header
      $xml->echoXML(true);
      // Exit (ensure no other output)
      exit();
    }
  }



}
?>
```

workload.js

```
// The PIM Function
function workloadPIM()
{
  // Last detail items asked for
  var teamid = 0;
  var username = "";
```

```
// Get detail for the given team and user
this.Detail = function(teamid, username)
{
  // Set last requested items
  this.teamid = teamid;
  this.username = username;

  // Create a ServerRequest object
  var sr = new ServerRequest();
  // Build the URL to the API
  var url = "api.php?";
  // Add the mode
  url += "mode=workload_api";
  // And the team and user detail
  url += "&teamid="+teamid+"&username="+username;
  // And the SID from FreeDESK
  url += "&sid="+DESK.sid;
  // Define the callback
  sr.callback = Workload.Callback;
  // Set the URL
  sr.url = url;
  // Set to XML
  sr.xmlrequest = true;
  // Call the API
  sr.Post();
}

// The callback method (API response)
this.Callback = function(xml, additional)
{
  // Check if the API has returned an error
  if (DESK.isError(xml))
```

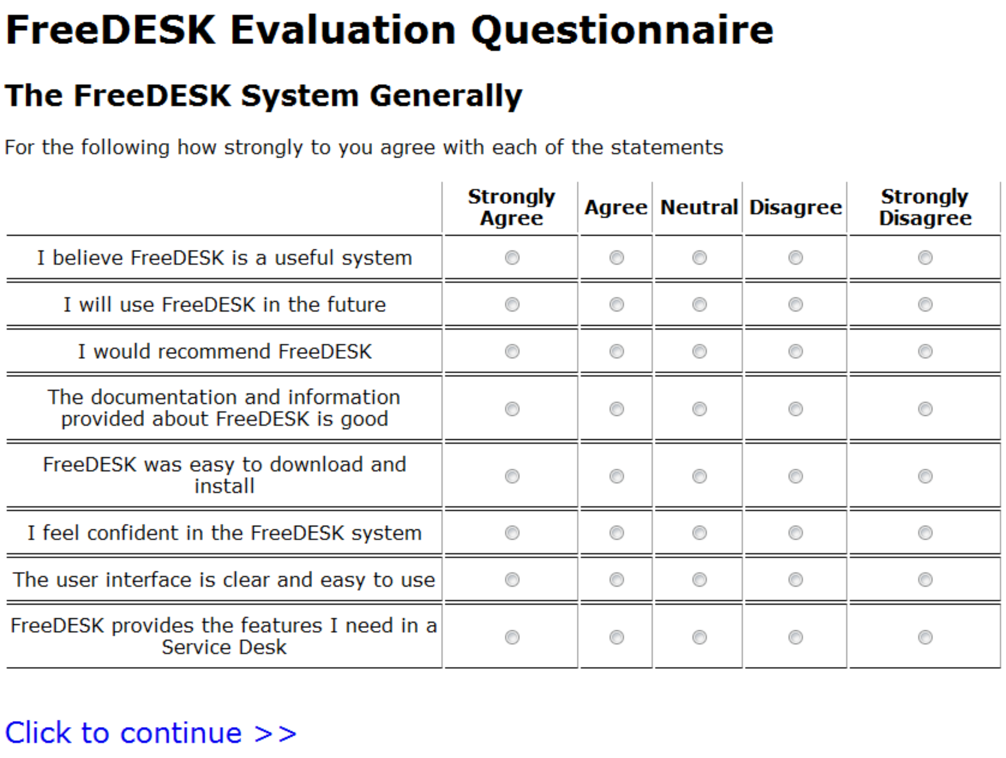
```
    {
      // Display error
      Alerts.add(DESK.getError(xml), 2, 10);
      // Exit
      return;
    } // no error so continue
    // Strip the response from the server
    var data = xml.getElementsByTagName("data")[0]
      .firstChild.nodeValue;
    // Get the element to display it in
    var eleID = 'detail_'
     +Workload.teamid
     +"_"+Workload.username;
    var ele = document.getElementById(eleID);
    // And display the data
    ele.innerHTML = data;
  }
}
// Instantiate the workloadPIM object
var Workload = new workloadPIM();
```

workload.css

```
hr.workload
{
  color: green;
  background-color: green;
  border: 0;
}
```

E Evaluation Questionnaire

The evaluation questionnaire was available through an online form during early August 2012 and had 27 respondents. There follows an image of one of the questionnaire pages, a full list of questions, answer options along with the number of and percentage for each option.



FreeDESK Evaluation Questionnaire

The FreeDESK System Generally

For the following how strongly to you agree with each of the statements

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
I believe FreeDESK is a useful system	○	○	○	○	○
I will use FreeDESK in the future	○	○	○	○	○
I would recommend FreeDESK	○	○	○	○	○
The documentation and information provided about FreeDESK is good	○	○	○	○	○
FreeDESK was easy to download and install	○	○	○	○	○
I feel confident in the FreeDESK system	○	○	○	○	○
The user interface is clear and easy to use	○	○	○	○	○
FreeDESK provides the features I need in a Service Desk	○	○	○	○	○

Click to continue >>

Section 1: General Information

1. Is your organization?		
Public Sector	6	22.22%
Private Sector	15	55.56%
Not-for-Profit	1	3.70%
Other	5	18.52%
2. The size of your organization (employees)?		
Over 1000	1	3.70%
501-1000	0	0%
101-500	8	29.63%
10-100	6	22.22%
¡10	12	44.44%

3. Do you have a current service desk system?		
Yes - commercial	11	40.74%
Yes - freeware	6	22.22%
Yes - in-house	6	22.22%
No	4	14.81%
4. What type of customers do you support?		
Internal	9	33.33%
External	8	29.63%
Both	8	29.63%
N/A	2	7.41%

Section 2: The FreeDESK System Generally

A series of statements which respondents are asked to indicate how strongly they agree with or not.

5. I believe FreeDESK is a useful system		
Strongly Agree	11	40.74%
Agree	13	48.15%
Neutral	1	3.70%
Disagree	2	7.41%
Strongly Disagree	0	0%
6. I will use FreeDESK in the future		
Strongly Agree	12	44.44%
Agree	11	40.74%
Neutral	1	3.70%
Disagree	2	7.41%
Strongly Disagree	1	3.70%
7. I would recommend FreeDESK		
Strongly Agree	11	40.74%
Agree	12	44.44%

Neutral	1	3.70%
Disagree	3	11.11%
Strongly Disagree	0	0%
8. The documentation and information provided about FreeDESK is good		
Strongly Agree	7	25.93%
Agree	9	33.33%
Neutral	9	33.33%
Disagree	1	3.70%
Strongly Disagree	1	3.70%
9. The FreeDESK system was easy to download and install		
Strongly Agree	19	70.37%
Agree	8	29.63%
Neutral	0	0%
Disagree	0	0%
Strongly Disagree	0	0%
10. I feel confident in the FreeDESK system		
Strongly Agree	8	29.63%
Agree	12	44.44%
Neutral	4	14.81%
Disagree	1	3.70%
Strongly Disagree	2	7.41%
11. The user interface is clear and easy to use		
Strongly Agree	6	22.22%
Agree	12	44.44%
Neutral	6	22.22%
Disagree	3	11.11%
Strongly Disagree	0	0%
12. FreeDESK provides the features I need in a Service Desk		
Strongly Agree	13	48.15%
Agree	8	29.63%

Neutral	4	14.81%
Disagree	1	3.70%
Strongly Disagree	1	3.70%

Section 3: Specific FreeDESK Features

Respondents are asked to indicate if they have used the features listed.

13. Installation Wizard		
Yes	25	92.59%
No	1	3.70%
Unsure	1	3.70%
14. Request Creation		
Yes	26	96.30%
No	0	0%
Unsure	1	3.70%
15. Request Update		
Yes	20	74.07%
No	4	14.81%
Unsure	3	11.11%
16. Development/Plugin Framework		
Yes	5	18.52%
No	20	74.07%
Unsure	2	7.41%
17. API		
Yes	1	3.70%
No	24	88.89%
Unsure	2	7.41%
18. Event Handling Framework		
Yes	2	7.41%
No	22	81.48%

Unsure	3	11.11%

Section 4: Development Features

The questions on development features were only asked if yes was answered to question 16 or 17 (6 respondents answered yes to one or both of these questions). For each of the statements the respondent was asked to indicate how strongly they agreed or not.

19. The development framework is well documented		
Strongly Agree	3	50.00%
Agree	2	33.33%
Neutral	1	16.67%
Disagree	0	0%
Strongly Disagree	0	0%
20. The provided interfaces are sufficient for my needs		
Strongly Agree	4	66.67%
Agree	1	16.67%
Neutral	1	16.67%
Disagree	0	0%
Strongly Disagree	0	0%
21. FreeDESK is developed in an open and easily extensible manner		
Strongly Agree	5	83.33%
Agree	0	0%
Neutral	1	16.67%
Disagree	0	0%
Strongly Disagree	0	0%
22. Developing extensibility in FreeDESK is easy		
Strongly Agree	1	16.67%
Agree	3	50.00%
Neutral	2	33.33%
Disagree	0	0%

Strongly Disagree	0	0%
23. I hope to incorporate extensions into the general release for others to use		
Strongly Agree	2	33.33%
Agree	1	16.67%
Neutral	1	16.67%
Disagree	2	33.33%
Strongly Disagree	0	0%

Section 5: Further Comments and Suggestions

A final set of questions were asked with "freeform" textbox input options.

- **24. Any overall comments about FreeDESK?**
- **25. Any suggestions for future development or additional features you would like to see?**
- **26. Any other comment you would like to make?**

Individual comments to section 5 questions are not included in this data.