Team Security Crawler

Final Project Design

February 10, 2012

Team Members:
Alex AUDRETSCH
Michael EATON
Trevor KRENZ
Andrew SIEGLE

Client:
Alan WLASUK
WDDinc.

Advisor:
Steve CHENOWETH

Contents

1 Introduction

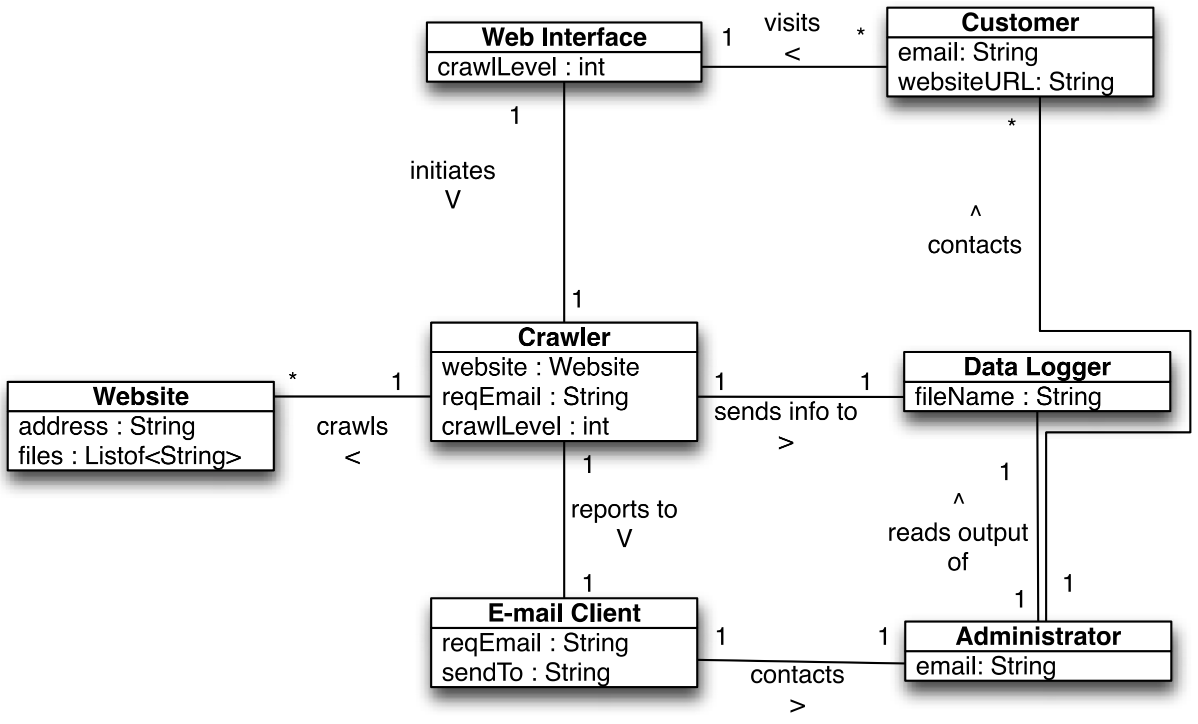
403 Security is a web security firm based out of Indianapolis that is looking to increase its client base by way of a free, automated security scan that a potential client can request by way of a web service. This service will save 403 employees valuable time and generate a sizeable number of clients for 403.

To this end, the Security Crawler system has been designed. Because it is an automated service that anyone can access, it is restricted to analysis of page source, and cannot perform in-depth penetration scans. However, a sizeable number of security vulnerabilites can be deduced from this data, such as out-of-date software and bad permissions in the html file tree.

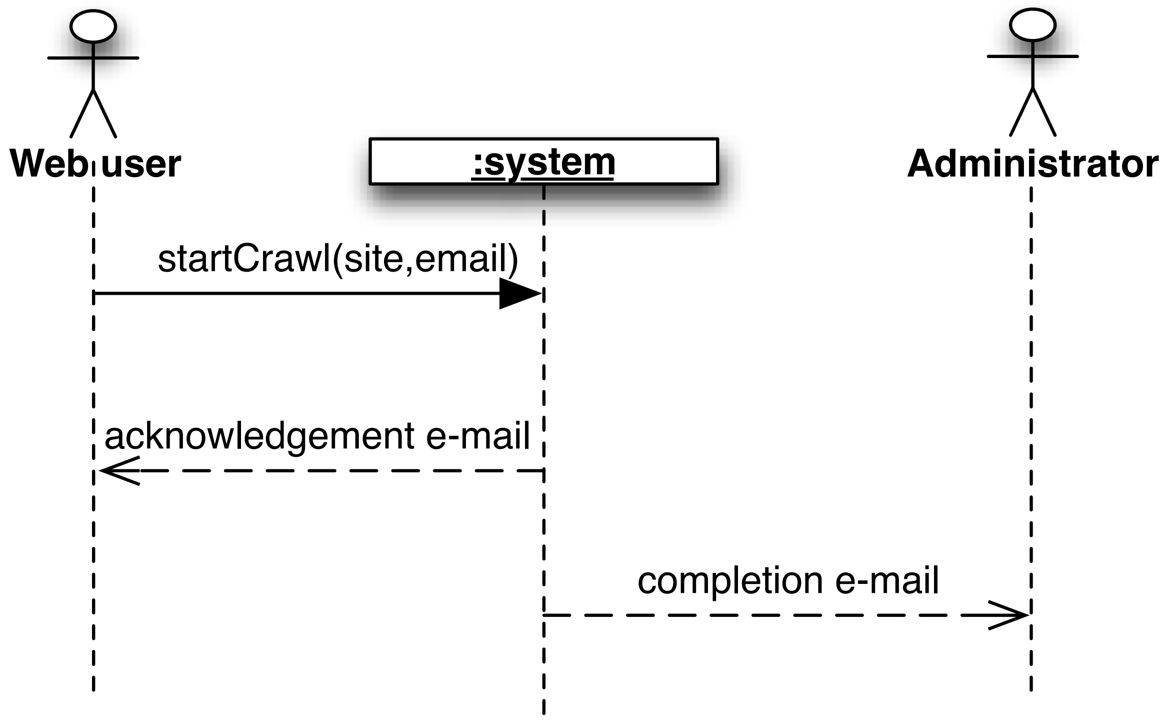
This document contains updated designs and architectural models for the security crawler system, as well as discussions of sysetm design choices in light of GRASP and GoF patterns. It includes the logical architecture of the system and finally an integration and acceptance testing plan for the system when it is deployed.

2 Analysis Models

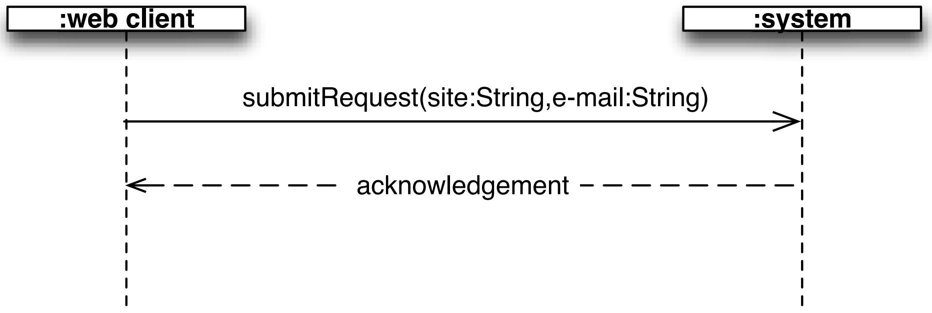
2.1 Domain Model

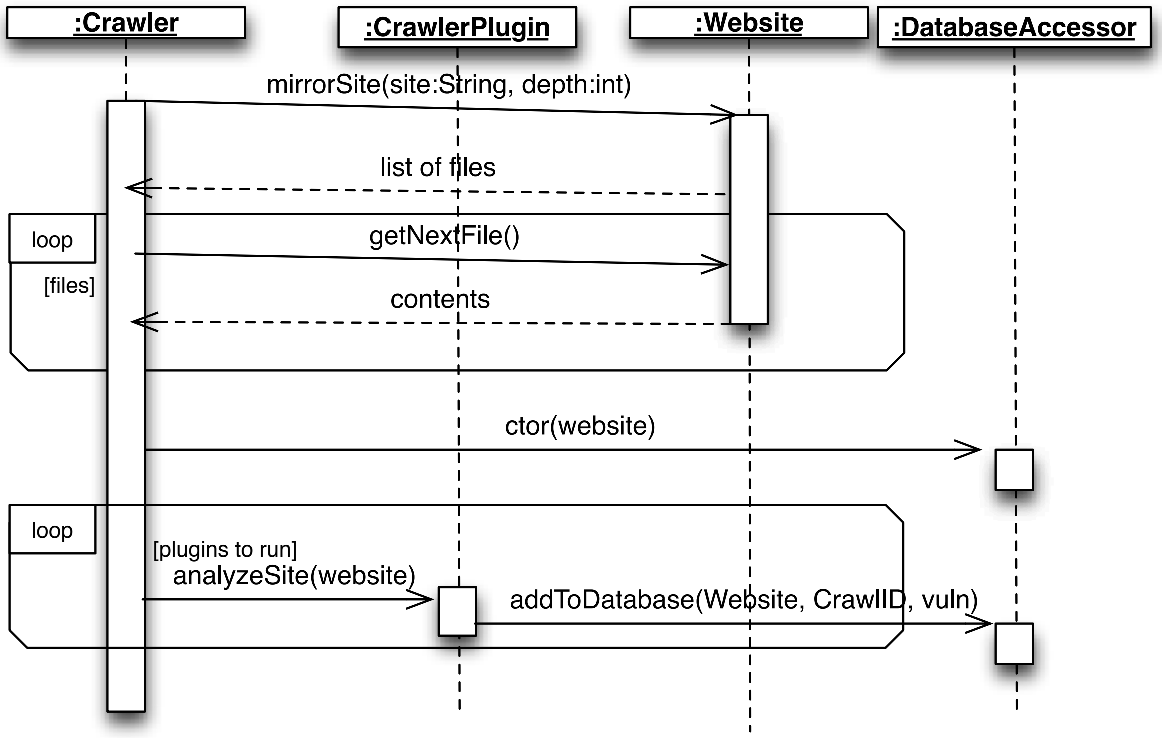


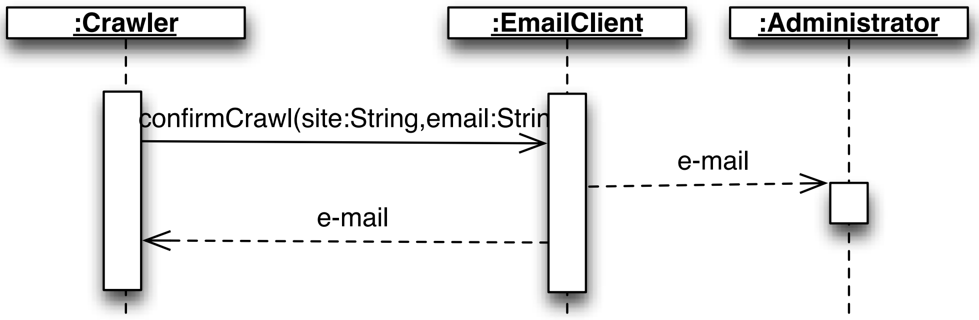
2.2 System Sequence Diagram



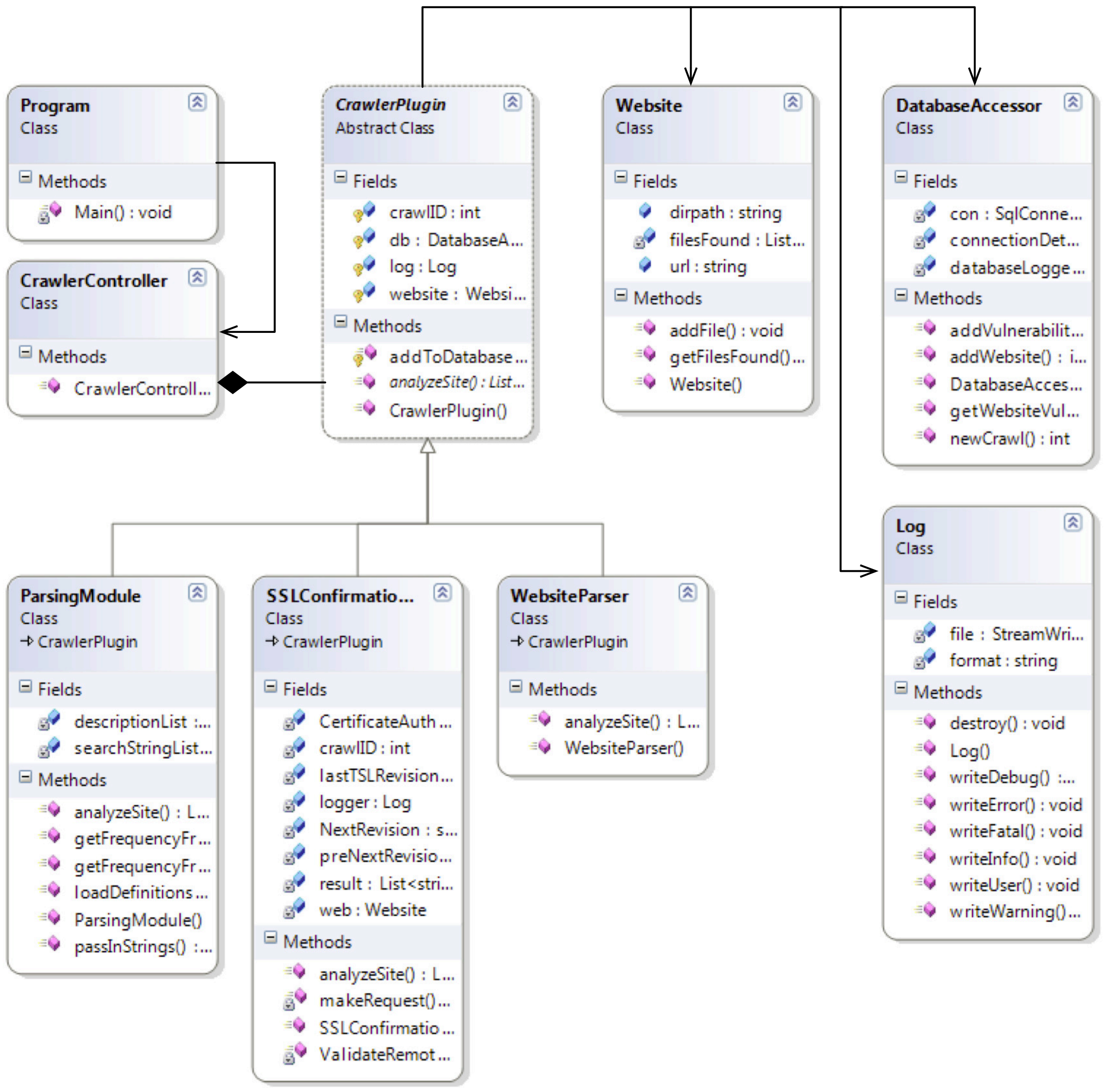
2.3 Sequence Diagrams







2.4 Design Class Diagrams



2.5 Operations Contracts

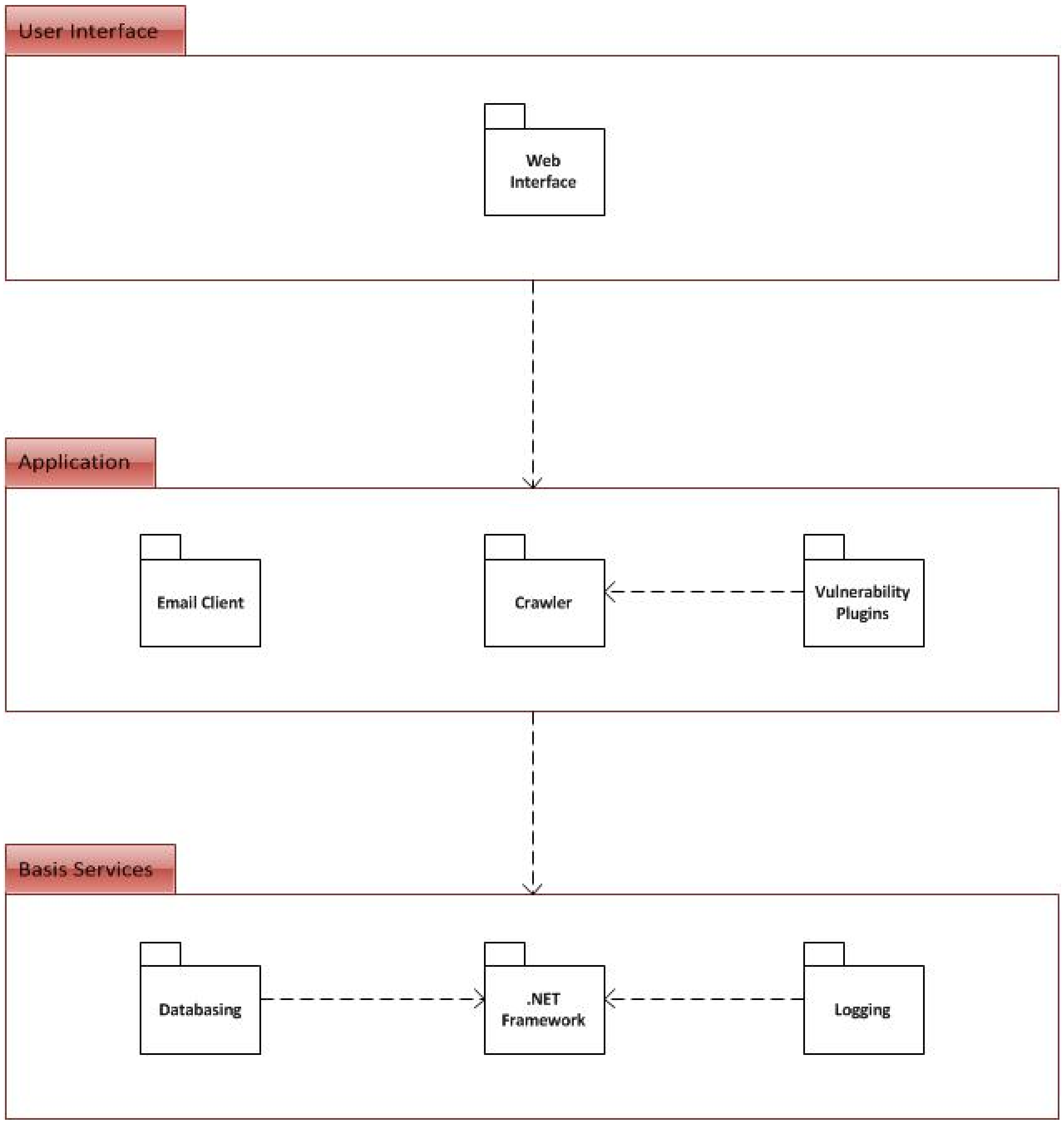
Contract 1: mirrorSite

Operation:	crawlSite(address :string, level int)
Cross References:	Use Case: Initiate Web Crawl
Preconditions:	• Site address is known • Site is valid • There is enough space on the hard disk to store the contents of the website
Postconditions:	• The website has been mirrored the hard drive • A list of all mirrored files is made • A list of all vulnerabilities found has been made

Contract 2: compileVulnList

Operation:	compileVulnList()
Cross Reference:	Use Case: Display a Trace Log
Preconditions:	• All files have been parsed • Certification and server information have been retrieved • There is a List that contains all of the possible vulnerabilities to scan for
Postconditions:	• A list of vulnerabilities is made • The list of vulnerabilities are written to the database

3 Logical Architecture



4 Design with discussion

4.1 GRASP

The *CrawlerEngine* class follows the **Controller** GRASP principle. This is advantageous because the CrawlerEngine has the information needed to instantiate other classes, such as the website that is to be crawled (used to create a Website instance) and email address (for a EmailClient instance).

The *CrawlerEngine* is also uses the **Creator** principle. This makes handling passing the Log instance to all of the other classes easier, since the Log can be passed to all objects when they are instantiated, as opposed to the CrawlerEngine class being visible to the other classes and having to use methods to pass the Log object. Because the Log always becomes visible via the constructor of classes, coupling is decreased because the classes created by the CrawlerEngine class do not need visibility of CrawlerEngine. This also prevents circular references, which increases cohesion and decreases coupling.

Also, we have added a *CrawlerPlugin* abstract class to our project. All plugins for the project will inherit from this class, and thus this is an example of **Polymorphism**. This increases cohesion because every class that inherits from the CrawlerPlugin class must override the methods in CrawlerPlugin, so it can be relied upon that all plugins can be interacted with via these methods. Additionally, the plugins for the crawler vary based on type, so it was a good idea to use polymorphism here.

The *AbstractInferenceModule* is an example of **Pure Fabrication**. This is a class that does not exist in our domain model, but having this class increases cohesion and lowers coupling. For example, the CrawlerEngine already has the responsibility of creating other instances, and because of the Single Responsibility Principle, the responsibility of interpreting the should not be within this class. Instead, it is the responsibility of the AbstractInferenceModule to interpret the parsed results of the website.

By having the *IInferenceModule* and the *IInterfaceResult*, a level of **Indirection** was achieved. This increases cohesion because it makes the classes more closely adhere to the Single Responsibility Principle. Anything that has to deal with the results is in the IInterfaceResult, while the actual processing of documents is done by the IInferenceModule.

The *server plugins* also demonstrate **Protected Variation**. Since this project is going to be taken over by another team once we are done, we know that it might change vastly, and the plugins for the crawler are points of instability. The class is then designed to handle plugins generally, which handles the instability.

The *CrawlerEngine* is also an **Information Expert**. This follows naturally from it being a controller because it gets all of its information from the GUI. Having all of the data in one place and having the CrawlerEngine be responsible for the data will increase cohesion because it allows the other class to focus one responsibility, and do that responsibility well.

4.2 Go4

Strategy Our JavascriptLibrary object will contain some sort of identification strategy which will be injected into each object. We expect that we will need to compose some Javascript identification functions out of others, so in order to reuse code well inject this behavior into each Javascript library were keeping track of. This contrasts to something like ServerSoftware, whose relevant unique information would be HTTPHeader-related.

Template Method We may, in the future, compose the JavascriptLibrary identifier by templating a solution from a string-recognition function, a Javascript VM execution function, and a .js file handler. C# is a functional language, so we can implement the functions directly, privately rather than making classes.

Singleton Our database adapter and email client classes are singletons. They network to outside sources, so we want to reduce the number of active connections as much as possible. In addition, our logger is currently only being instantiated once, but were injecting it into modules rather than making it static in case we want to refactor in the future and use several types of loggers.

Adapter Our two aforementioned singletons, database adapter and email client adapter, are adapters for .Net SQL and SMTP objects respectively. Its obviously much easier on our part to have one-stop adapters than to worry about making the SQL- and email-handling objects directly.

Iterator Our CrawlEngine is an iterator on files it needs to analyze, using the GetNextFile() method. It will contain logic on which files to ignore and which to analyze while iterating (something like robots.txt would be ignored.)

Template The plugins follow the Gang of Four pattern of a template. Each pluging implements its own version of analyzeSite which is then called from the CrawlerController class. Each new plugin will follow this template in order to make creating new plugins easier.

5 Integration and Acceptance Test Plan

5.1 Current Status

5.1.1 Completed

- Database functionality - the database has been created on the school-hosted server with appropriate accounts. Stored procedures are used to access the data. A DatabaseAccessor class is present in the system as a singleton.
- Website mirroring - the system uses the external program 'HTTrack' to access the website and copy all available pages into a local directory in order to parse those files. The files are saved in a directory named appropriatedly with the website and a timestamp.
- Basic parsing module - The basic parsing module reads a file of strings to look for in the files and scans them appropriately and records statistics according to frequency and also records vulnerabilities associated with the found contents.
- Framework - The system's basic framework and structure is laid down. The Crawler acts as a controller and a factory, creating the needed plugins and running the analyseSite() method to invoke their functionality.

5.1.2 In Progress

- Server Info - Obtains server information by analyzing HTML headers and packets received by the crawler.
- SSL Certificate checking - Compare received SSL certificates to a known certificate authority in order to confirm validity.
- Script parsing - the system will be able to check for php, javascript...etc. vulnerabilities.

5.1.3 To Do

- E-mail client - The system must send an e-mail to the administrator.

5.2 Acceptance Test Plan

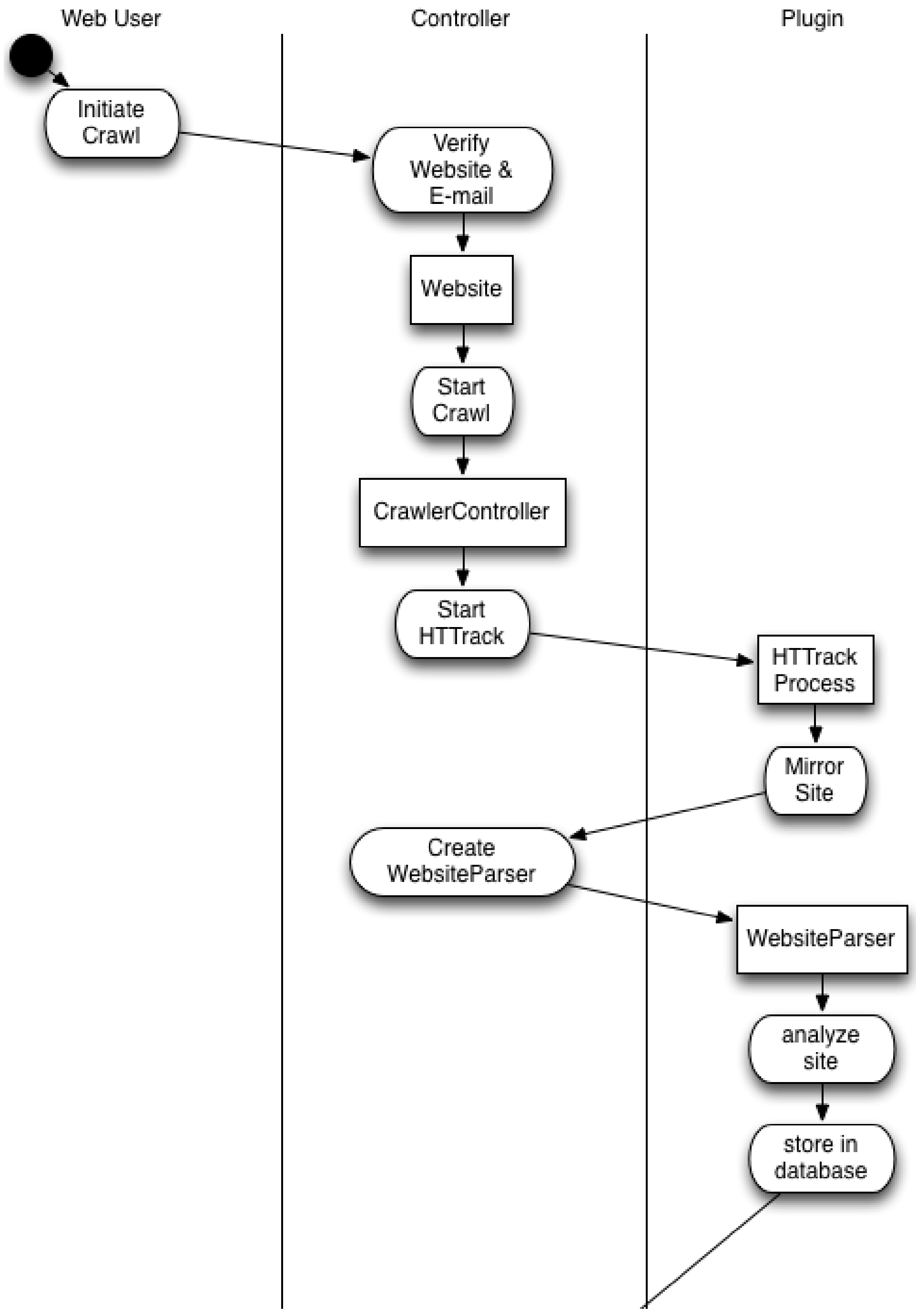
For this project to be an acceptable success, it must be able to crawl a site, write to a database, and notifiy the administrator. It must check SSL Certificates, retrieve server and website software packages and versions, and parse for basic javascript and php vulnerabilities. The system also must be able to accept additional plugins for scanning various vulnerabilities, allowing for future expansions.

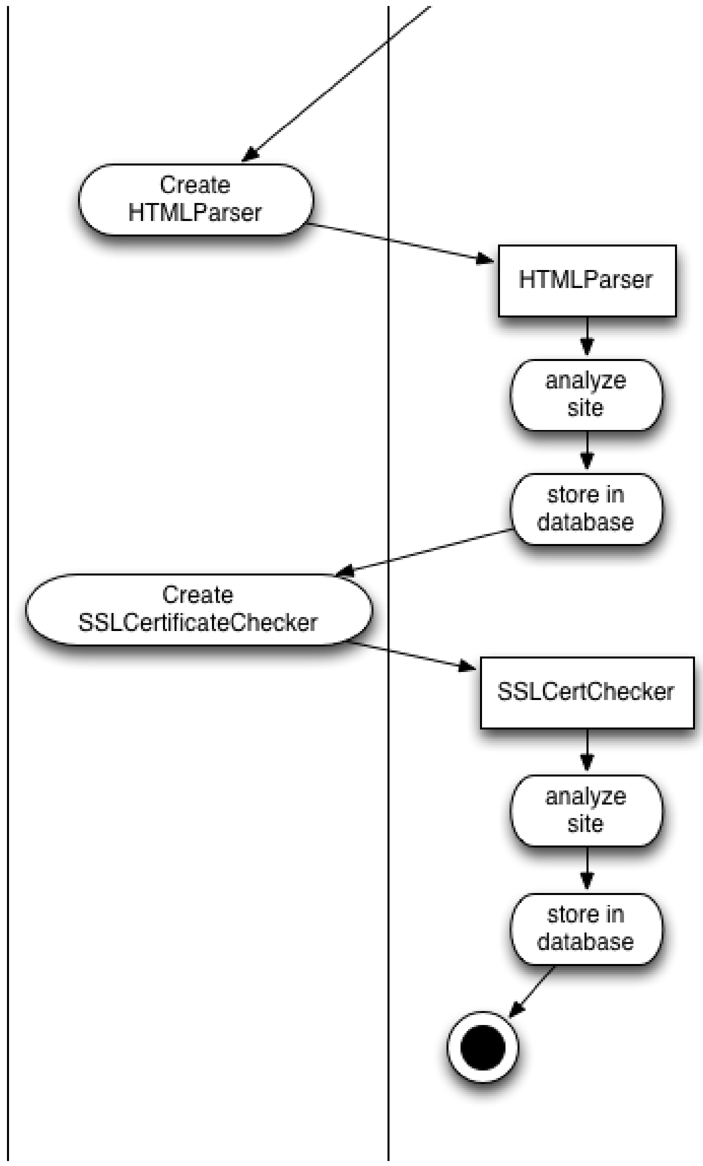
The system will be tested against a known security vulnerability scanner (Acunetix) in order to validate found vulnerabilities and functionality.

5.3 Required Features for Acceptance

Feature	Priority	Version	Reason	Accepted	Acceptance Criteria
Parse / analyze HTML header information	Critical	v1.0	Source of basic webpage and server data.	Yes	Output matches Acunetix output for at least 5 sites
Examine server file tree	Critical	v1.0	Potentially sensitive information may be hosted publicly.	Yes	Applicaiton mirrors complete site
Save crawl results to database	Critical	v2.0	Results must be saved for report generation and statistics tracking.	Yes	Vulnerability data saved successfuly to database
Report crawl results	Critical	v1.0	Results must be distributed to Alan so that he may report customized results to the user.	Yes	Email successfully delivered to admin-defined address
Parse and analyze Javascript files	Critical	v3.0	Javascript files are indicative of production framework.	Yes	Parse output matches basic Acunetix output
Parse / analyze HTML form information	Important	v1.5	Forms may indicate which framework generated them.	Yes	Output matches Acunetix output
Examine site security certificates	Important	v2.0	Check for expired certificates.	Yes	Expired and faulty certificates identified successfully
Maintain operations tracelog	Important	v1.0	Allows for traceback of operations for quality assurance and testing purposes.	Yes	All operations logged to acceptable detail level
Basic Web GUI	Useful	v3.0	Allows a user to enter information from client's web site.	No	User able to initiate a successful crawl via the web UI
Compare previous crawl information	Useful	v3.0	Allows the user to changes/improvements to website.	No	N/A
Save search engine optimization (SEO) data	Useful	v3.0	Retains information found during crawling for later statistical analysis on websites.	No	N/A
Statistical analysis of crawls	Useful	v3.0	Statistical tracking would give the client useful information about his or her website: how said site's vulnerabilities fit into the overall picture of web vulnerabilities, for example.	No	N/A

6 Activity Diagram




Create HTMLParser
HTMLParser
analyze site
store in database
Create SSLCertificateChecker
SSLCertChecker
analyze site
store in database

7 Who Done What Table

Team member	Tasks
Alex Audretsch	• Updated DCD • GoF Principles
Michael Eaton	• Introduction • Logical Architecture • Integration and Acceptance Plan
Trevor Krenz	• Design with explanation (GRASP and GoF Principles)
Andrew Siegle	• Document Design • Sequence Diagrams • Integration and Acceptance Plan