Proceedings of the 6th Winona Computer Science Undergraduate Research Symposium

April 19, 2006

Table of Contents

Bayesian Filter for Blocking Spam

Brandon Wienkes
Winona State University
MIS/CIS Majors BUSA Minor
608-553-0483
BDWienke1130@winona.edu

ABSTRACT
Spam emails are a nuisance and as spam blockers get more efficient, spam writers get more creative. Spammers love to attack corporations and even personal email. The Bayesian filter is an effective filter that blocks spam emails. A JavaMail program for spamming has been developed to study the effectiveness of Bayesian filters.

Categories and Subject Descriptors
D.2.2 [**Design Tools and Techniques**]:

D.2.5 [**Testing and Debugging**]: *Testing*

General Terms
Networking, Spam Filtering

Keywords
Bayesian, filter, spam, ham

1. INTRODUCTION
In the corporate world, a company's connection to the outside world is a key to its success. A company should not have to worry about the credibility of its incoming emails.

> "Spam refers to electronic junk mail or junk newsgroup postings. Some people define spam even more generally as any unsolicited e-mail. In addition to being a nuisance, spam also eats up a lot of network bandwidth. Because the Internet is a public network, little can be done to prevent spam, just as it is impossible to prevent junk mail. However, the use of software filters in e-mail programs can be used to remove most spam sent through e-mail." [27]

Even though there are tools for preventing spam from getting into an inbox [2], none can prevent all of the spam from getting through [4]. Far too many times, a valid email may be blocked and put in the junk mail folder. It is someone's job to check the emails even though they have gone through the blocking process already. Even at a personal level, it is a nuisance to sort through emails when just trying to check our daily emails. This takes time and we all know "time is money." In fact, Barracuda Networks wrote:

> "Spam accounts for 45% of all e-mails, or 15 billion messages every day, and costs business world-wide a total of $20 billion a year in lost productivity and technology expenses, according to the Radicati Group, a market research firm in Palo Alto, CA. The firm predicts the number of daily spams will rise to more than 50 billion by 2007, and costs will reach almost $200 billion per year." [16]

In the last five years, there has been considerable effort to stop spam. Sipior goes in depth about the legislative action that has taken place and its failures [13]. Paul-Alexandru Chirita, Jörg Diederich, and Wolfgang Nejdl have come up with new ideas of using mail ranking systems where people on a network can combine spam lists to block spammers [3]. Paul Graham shares with us effective ways to block spam. Graham explains different ways of blocking spam: mail server blacklists, Signature-Based Filtering, Bayesian (aka Statistical) Filtering, Rule-Based (aka Heuristic) Filtering, Challenge-Response Filtering, Laws, FFBs, Slow Senders, and Penny per Mail [8]. However, when trying to block spam, we may also block valid emails. Shlomo proposes an idea of combining email models to prevent false positives (i.e., emails that are blocked when they shouldn't be) [9]. Dr. Neal Krawetz gives a good overall summary of each aspect of spam and its future [10]. The future looks promising for spammers given that 45% of emails are spam and this number is rising [4], which means we need to upgrade our algorithms and improve our filter and spamming techniques. From this, we have learned that the Bayesian spam filter has become an efficient spam filtering system on the free market.

The Bayesian filter described in section 2.1 gives us an introduction into Paul Graham's filter. This section includes the background information, the theory, and the algorithms behind the Bayesian filter. The JavaMail API and the experimentation using a JavaMail program to test email filters is introduced in section 2.3. The email filters are briefly explained in section 2.3.3 and their results are displayed in section 3. Finally, my analyses of the results from the filters are explained and the future of the Bayesian filters is taken into consideration in the last section of the paper.

Proceedings of the 5th Winona Computer Science Undergraduate Research Seminar, April 20-21, 2006, Winona, MN, US.

2. METHODOLOGY

2.1 Bayesian Filter

Bayesian spam filters calculate the probability of an email being spam based on its body, which is partitioned into tokens. Unlike simple content-based filters, Bayesian spam filtering is a learning filter. It learns from both positive examples (i.e., spam) and negative examples (i.e., legitimate email or ham) making it robust, adaptable, and an efficient anti-spam approach. One solid aspect of this type of filter is the low number of false-positives, which are legitimate emails that are classified as spam, when truly they are not. Often, it is more hazardous to the recipient to have their email classified as junk when it truly is not, than actually receiving spam.

2.1.1 Background

Bayesian filtering was proposed in 1998 by Sahami et al. [12], but did not gain attention until 2002 when it was described in the paper "A Plan for Spam" by Paul Graham [5]. Since that time it has become a popular filtering system that is used to separate spam email from ham email. Many modern mail programs such as Mozilla Thunderbird implement Bayesian spam filtering. Server-side email filters, such as SpamAssassin and ASSP (Anti-Spam SMTP Proxy Server), make use of Bayesian spam filtering techniques, and the functionality is sometimes embedded within mail server software itself [12].

```
(let ((g (* 2 (or (gethash word good) 0)))
      (b (or (gethash word bad) 0)))
   (unless (< (+ g b) 5)
     (max .01
        (min .99 (float (/ (min 1
        (/ b nbad))(+ (min 1 (/ g ngood))
        (min 1 (/ b nbad)))))))))
```

Figure 1. The probability that each token in the email contains spam. [5]

2.1.2 Theory

Paul Graham explained his theory in [5], and gave an overview of how his Bayesian filter was used. Graham started off with two groups of email messages: one group of spam and one group of non-spam email both which are hash tables in code. The emails were scanned in, including the headers, embedded html, and JavaScript. In [5], Graham considered alphanumeric characters, dashes, apostrophes, and dollar signs to be part of tokens, and everything else to be a token separator. Graham improved his technique by ignoring tokens that were all digits, and also ignoring html comments [13]. Some improvements are: Case is preserved, Exclamation points are constituent characters, Periods and commas are constituents if they occur between two digits, $20 and $25, and finally tokens that occur within the To, From, Subject, and Return-Path lines. [6] From there, Graham counts the number of times each token was found in each group (spam and ham) and puts them into separate hash tables. A third hash table is created mapping each token to the probability that an email contains spam. Figure 1 shows us how this probability is calcualated.

Word is the token whose probability we are calculating. *Good* and *bad* are the hash tables Graham created in the first step (two groups of emails) and *ngood* / *nbad* are the number of occurrences of the each word in those tables. To avoid false positives, Graham has found that by doubling the numbers in the good hash table, it helps the probability of having false positive emails (Figure 2: line 1 `(* 2…)`). From here, he only considers words that occur more than five times in total (Figure 2: line `(unless (< (+ g b) 5))`. At first, Graham decided to use .01 and .99 as the probability to assign words that occur in one hash table but not in the other [5]. (Figure 2: using .01 and .99 to determine the probability)

"When new mail arrives, it is scanned into tokens, and the most interesting fifteen tokens, where interesting is measured by how far their spam probability is from a neutral .5, are used to calculate the probability that the mail is spam" [5]. You calculate the combined probability by looking at figure 2.

```
(let ((prod (apply #'* probs)))
  (/ prod (+ prod (apply #'* (mapcar
#'(lambda (x) (- 1 x)) probs)))))
```

Figure 2. Lisp code for combined probability. (*probs* is the list of the top 15 spam words) [5]

To better understand combined probability and a better visual, figure 3 shows us in a normal math equation, rather than in pseudo code [7].

If a and b are the probabilities associated with two independent pieces of evidence, then combined they indicate a probability of:

$$\frac{ab}{ab + (1 - a)(1 - b)}$$

Figure 3. A mathematical representation of combined probability.

As for words that have never been seen, Graham recommends using a probability of .4, but thinks this can be adjusted for improvements.

2.2 JavaMail

The JavaMail API provides a platform-independent and protocol-independent framework to build mail and messaging applications [19]. The JavaMail API is implemented as a Java platform optional package and is also available as part of the Java platform, Enterprise Edition. The API provides the user with a set of

abstract classes defining objects that comprise a mail system. From there, the API can be extended and can be sub-classed to provide new protocols and to add functionality when necessary. Developers can subclass JavaMail classes to provide the implementations of particular messaging systems, such as IMAP4, POP3, and SMTP [21]. IMAP stands for **I**nternet **M**essage **A**ccess **P**rotocol. It is a method of accessing electronic mail or bulletin board messages that are kept on a (possibly shared) mail server [28]. Short for ***P**ost **O**ffice **P**rotocol*, POP is a protocol used to retrieve e-mail from a mail server. Most e-mail applications (sometimes called an *e-mail client*) use the POP protocol, although some can use the newer IMAP (Internet Message Access Protocol) [30]. Short for ***S**imple **M**ail **T**ransfer **P**rotocol*, SMTP is a protocol for sending e-mail messages between servers [31]. For the purpose of my paper and experimentation, all that was needed is the SMTP portion of the JavaMail package. To use this feature, we added the Mail.jar and Activation.jar files to our project build path to send out emails on a SMTP mail server. These jar files are available for download at [24] and [20], respectively.

2.3 Experimentation

To examine how Bayesian filters work and how efficiently they can stop spam, I created a spamming program where I spammed my own email box with different spam filters. Section 2.3.1 gives an overview of setting up a JavaMail program to send emails with SMTP on a server. Section 2.3.2 shows how I created the "random" spam emails and finally section 2.3.3 provides an overview of the spam filters used in the experiment.

2.3.1 JavaMail Coding

Prior to coding the application, the Mail.jar and Activation.jar files were downloaded. When adding them to the project build path, make sure that these jars are stored in the project folders so that it is easier to create executable jar files.

SendMail(S1, S2, S3, S4, S5) shows us how to send an email in pseudo code. The inputs to this method are: S1 (the server we plan to connect and send mail from), S2 (the Address from where the email is from), S3 (the Address to where the email is intended to go), S4 (the subject line of the email), and S5 (the body of the email). All of these parameters are Strings.

SendMail is a void method but will send the email if there are no exceptions and the 'to Address' is relevant. It is wise to display a message to the console to determine if the mail was sent.

We use the variables: properties: Properties (Java class), session: Session (JavaMail class), message: MimeMessage (JavaMail class), and tr: Transport (JavaMail class). The SendMail method needs no other variables to accomplish its mission and it is described below. sendMail(S1, S2, S3, S4, S5): The method goes in the order as followed:

1. (Get the system properties.)

properties = getSystemProperties

"The Properties class represents a persistent set of properties. The Properties can be saved to a stream or loaded from a stream. Each key and its corresponding value in the property list is a string [22]. The properties class allows us to store our SMTPs (secure line) host or SMTP host, which we use when sending the email.

2. (Add the server to the properties)

properties = properties.put("mail.smtp.host", S1)

3. (Add the server authorization to the properties)

properties = properties.put("mail.smtp.auth", "true")

4. (Create an instance of a Session based on the properties. We can also set the .setDebug feature to true, which will display the connections to the console) We then create a default instance of a session, part of the JavaMail API, based on these properties. "The Session class represents a mail session and is not sub-classed. It collects together properties and defaults used by the mail API's. A single default session can be shared by multiple applications on the desktop. Unshared sessions can also be created. The Session class provides access to the protocol providers that implement the `Store`, `Transport`, and related classes." [23] From the creation of this session, we can finally create or MimeMessage (JavaMail class), which is essential our email.

session = session.getDefaultInstance(properties)

5. try {

6. (Create a new MimeMessage based off of the Session just created)

message = new message(session)

7. (Set the recipients to the email)

message = message.setRecipient(S3)

8. (Set the from address of the email) If we are on an open relay network, we can set the fromAddress to anyone whom we please, which is pretty scary. By this, I mean you can receive email from a legitimate source and it still may be spam. As tests for our open relay SMTP server, I sent emails to myself from jesus@winona.edu and also emails from my friends to myself to determine if a reply would actually send (which it did).

message = message.setFrom(S2)

9. (Set the subject of the email)

message = message.setSubject(S4)

10. (Set the body of the email)

message = message.setText(S5)

11. (Create a new Transport based off of the session. This transport will do the connecting and sending of the email)

tr = session.getTransport("smtp")

12. (Connect to the server using a relevant ID and Password)

tr.connect(S1,"UserID", "password")

13. (Send the message to the recipients)

tr.sendMessage(message, message.getAllRecipients())

14. (Close the connection to the server)

tr.close()

15.} catch (exception) {

16. (Print the exception to the console to find out why the email did not send)

print exception}

That is basically a mini JavaMail program in a nutshell. We can increase the options of our email with more coding to include priority, confirmation email, and even to see if they have opened the email.

2.3.2 The Spammer

In order to send out spam emails, I needed to know which spam words are actually included in an email. For this purpose, I used the spam phrases from the list provided in [1]. Some included phrases are: free cell phone, free degree, free diploma, free game, free games, free gas, free gift, free list, etc. While creating the emails, I kept track of the subject, which is determined whether it is a spam or ham email, body, spam phrase count, and ham phrase count. I used the java random generator to randomly pick how many words/phrases and spam words will be in each email. If it is a spam email, in this example, I figured that at most 50% of the words could be considered spam and the rest filled in with ham phrases. This shows us steps of doing this in java.

1. randomGenerator = new Random()

2. numberOfWords = Math.abs(randomGenerator.nextInt(500) + 1)

3. numberOfSpam = Math.abs(randomGenerator.nextInt((int)Math.ceil((numberOfWords*.5))));

By taking the absolute value, we prevent possible errors from happening, where the pre-conditions to the generator are that the numbers being passed in must be positive. From there, I keep appending to a String by adding a random spam phrase then followed by a ham word until all of the spam phrases have met their requirements. This can be viewed in figure 4.

```
for(int i = 0; i < numberOfWords; ++i){
    if(numberOfSpam > 0){
        spamString += (spam[randomGenerator.nextInt(spam.length)] + " ");
        --numberOfSpam;
    }
    spamString += (ham[randomGenerator.nextInt(ham.length)] + " ");
}
return spamString;
```

Figure 4. Java representation of filling the email with words.

To keep the data sets the same, the program actually uses a file writer and prints out the emails to a text file. From there, I can read in the text file as many times as needed while keeping the same data but also allowing us to test different spam blockers. I have tested each blocker with five text files each with 500 words. The first text file is used for the Bayesian learning. I manually went through the emails and classify each of the emails as ham or spam. The second test is one with no spam emails. All of the emails are considered non-spam. The third test is an email with all spam emails having the possibility of having up to 30% of the total phrase count being spam. The fourth and fifth are the same but with 40% and 50%, respectively.

A randomly generated sample email is shown below in figure 5.

Body: get your reading consistent make money at home condemn free list feudal slot-machine relevant cemetery separate caricature occasion(ally) professor guarantee humour prevalent proceed secretary schism omitted occur sensible sacrilegious grammar anxious temperament preceding exhilaration millionaire forty ninety opinion pilgrimage conscientious existence occurrence desperate primitive analysis conscience acquire embarrass receive meant psychology character fictitious sensible fulfill (fulfil) prejudice prevalent

Subject: Ham

TotalSpam: 4

TotalWords: 47

Figure 5. A sample email filled with spam and ham phrases.

Obviously the email does not make grammatical sense. But, the Bayesian filter does not look to see if the phrases are meaningful. When we describe the tokens, we mean that each word is separated like so: get, your, reading, consistent, make, money, at, home…etc.

2.3.3.1 Junk-Out for Microsoft Outlook (Version 1.15.0049)

Junk-Out™ is an e-mail filtering program which adds directly into Microsoft Outlook® and only runs when you need it [25]. Junk-Out™ scans the content of each message's body and header, then uses past experience of 'good', 'bad' and 'neutral' words in both junk and non-junk messages to arrive at an overall probability for the current message being junk. It then uses probability thresholds to classify each message [26]. To make this possible, Figure 8 shows how the filter is setup. Junk-Out™ includes a 'seed vocabulary' of over 30,000 'bad' words to provide initial 'past experience' to the filter. Junk-Out™ tailors Bayesian filtering to your individual e-mail style. Use the Start Up Wizard to provide the initial 'past experience' of non-junk mail from your Sent Items, and other folders you choose as good examples. Junk-Out™'s Bayesian / content-based filter goes on learning as long as you go on using it. Junk-Out™ adapts to changes in your own e-mail style and to the ever-changing stratagems used by junk e-mailers. [26]

2.3.3.2 SpamBayes (Version 1.90)

The SpamBayes project developed a statistical (referred to as Bayesian) anti-spam filter, initially based on the work of Paul Graham. The major difference between this and other Bayesian filters using Graham's work is the emphasis on testing newer approaches to scoring messages.

> "The SpamBayes team tinkered with new algorithms, tweaking existing algorithms, and, most importantly, did enormous test runs, slamming tens of thousands of messages against each other, in an attempt to quantify whether or not a change to the system was beneficial. The new algorithm is a combination of work from Gary Robinson and Tim Peters, and provides not just a 'spam' and

'ham' rating, but also an 'unsure' rating, for those messages where it can't work out how to rate the message [17].”

2.3.3.3 Outlook Spam Filter 3.0

Outlook Spam Filter 3.0 is an easy-to-use Microsoft Outlook® add-on designed to provide an advanced protection against spammers and unsolicited emails. The program uses Bayesian filtering technology that identifies about 97% of incoming spam messages [14]. The spam filter catches spam and puts it in its own folder.

2.3.3.4 Spam Bully Outlook Version (3.0.0.30)

This filter uses a Bayesian spam filter and places the junk mail inside the junk folder. This also allows for black/white lists, the ability to report junk mail, forwarding good emails to your cell phone, statistics, and multi-language interfaces [15].

2.3.3.5 Spam Reader 2.25

Spam Reader is a powerful anti-spam filter for Microsoft Outlook combining ease-of-use and a high degree of protection against unsolicited emails. A Bayesian algorithm based on statistical analysis detects up to 98% of spam messages. Spam Reader is easily integrated into Microsoft Outlook and needs no additional adjustments. It starts working immediately after the installation [11]. Spam Reader uses the Bayesian filter like the other software filters being tested.

3. Results and Analysis

There are two measures that are important when comparing spam filters: the number of spam emails missed by the filter (the false-negative ratio) and the number of legitimate emails incorrectly categorized as spam (the false-positive ratio). Of these, the false-positive ratio is by far the most important; if one spam should happen to slip by the filter it is easy to just click the spam button and allow the Bayesian filter to place it in the spam folder and by doing so, learning from its mistake. However, if a legitimate email is placed in the spam folder, the recipient may never receive it. The recipient should not have to check the spam folder for legitimate emails; otherwise this would defeat the purpose of the filter. Figure 6 shows us these numbers in percentages. Each filter was tested with the same data sets. However, two of these filters grabbed online sources for further learning which seems to have skewed their results and in fact, made them slightly less efficient.

Product	% Caught	% False Positives	% False Negatives
Out-Junk	95.59%	0.00%	4.15%
Spam-Bayes	100.00%	0.00%	0.00%
Outlook Spam Filter	98.31%	1.63%	1.82%
SpamBully	99.35%	1.63%	1.95%
Spam Reader	97.92%	1.71%	2.08%

Figure 6. Shows us the products results for catching spam, false positives, and false negatives.

The Spam-Bayes software performed flawlessly in my tests.

Figure 7 shows the results of the 500 emails sent out that did not contain spam phrases.

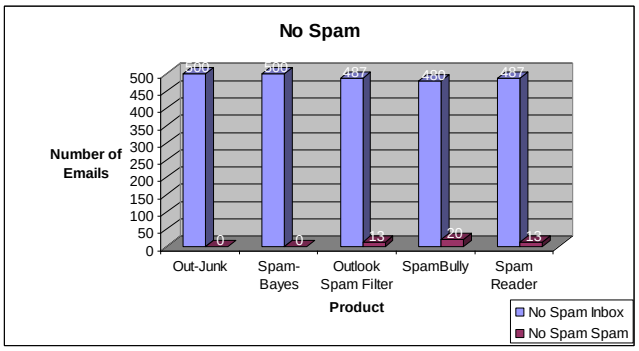

Figure 7. This chart shows us the number of emails classified as legitimate and spam emails when testing with no spam.

Out-Junk and Spam-Bayes performed the best when all emails were considered legitimate emails. The other software packages are using previous knowledge from a source off the internet or its own files which is why they are considering some emails as spam.

Figure 8 shows us the results for sending out spam that had a possibility of having 30% of the total phrases being spam. In total, 233 legitimate emails and 267 spam emails were sent out in this data set.

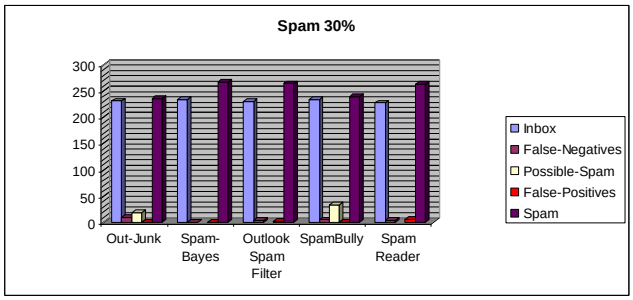

Figure 8. The results of the spam emails containing up to 30% spam phrases.

Again, the Spam-Bayes performed flawlessly and Spam-Reader performed poorly having 6 false positive emails. Out-Junk and SpamBully come with the option to have an “unsure” or “possible” junk folder where if the probabilities are undetermined, it places them there. This is a nice feature which can help prevent false positives when we are allowed to change the restrictive probabilities.

Figure 9 shows the results of spam emails containing up to 40% spam phrases. We should see a more efficient filter in each of the

software programs. The spam and legitimate emails were deadlocked at 250 a piece.

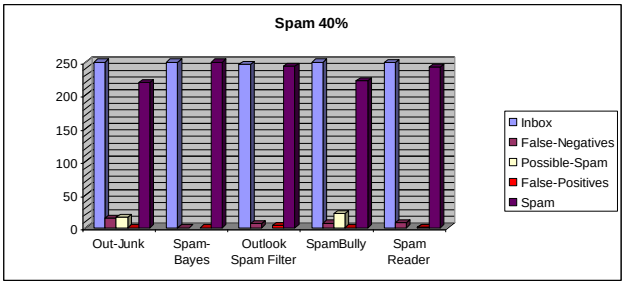

Figure 9. Results of the spam emails containing up to 40% spam phrases.

Spam-Bayes performed near perfection again, while the Outlook Spam filter performed poorly with 3 false positives. Out-Junk, Outlook Spam Filter, SpamBully, and Spam Reader all allowed a few spam emails to slip by into the inbox but is not necessarily a terrible thing. Figure 13 shows us the percentages of where the emails fell into which folders when using SpamBully.

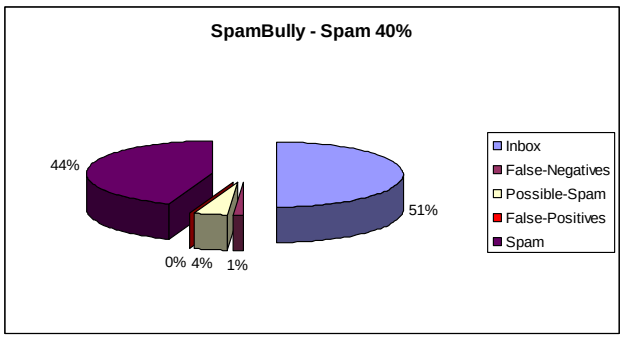

Figure 10. SpamBully's results for filtering out the emails with up to 40% Spam phrases per spam email.

In Figure 10, 1% of the spam emails were allowed into the inbox. Since half of the emails were spam, SpamBully successfully filtered 98% of the spam while avoiding filtering any legitimate email. Another feature is that 4% of the emails were considered "possible," the results showed that all of these emails were spam.

Figure 11 shows the results of the final test allowing spam emails to have up to 50% spam phrases in them.

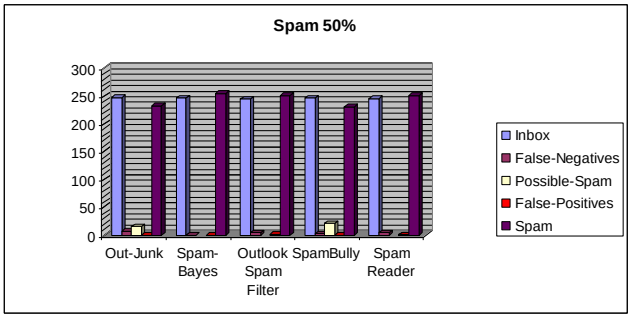

Figure 11. Results of the spam emails containing up to 50% spam phrases.

Outlook Spam Filter ended up with two false positives with four false negatives and Spam Reader allowed one false positive with four false negatives too.

My overall results from my testing conclude that Spam-Bayes to be the most efficient filter. It showed perfection in my tests but obviously is not perfect in real life situations. Spam-Bayes uses an advanced algorithm using the chi-squared approach. This allows for an unsure folder to be created and to address the problem with Graham's algorithm of producing a score of 1 (spam) or 0 (ham) [18].

3.1 The Future

The future looks promising to the Bayesian filter and negative to spammers. However, sometimes installing and using the Bayesian filter can be somewhat complicated. I had problems setting up my Junk-Out software to filter though the POP rather than the IMAP account. Now to a beginner, this may make no sense at all. Also, it does take some time, a few minutes to scan all legitimate mails into the learning process. However, the main feature is that when trained to spot what is spam and what is legitimate mail, these smart filters can catch, in many cases, more than 99% of junk messages [32].

Eventually, people will find a way to break past the filters. Graham has even stated that if someone sends an email with just a link in it, the filter will probably not catch it. From there, Mr. Graham-Cumming, who is a member of the Sophos Anti-Spam Task Force, has found a way to beat Bayesian filters.

> "To find out how to beat the filters Mr. Graham-Cumming sent himself the same message 10,000 times but to each one added a fixed number of random words. When a message got through he trained an "evil" filter that helped to tune the perfect collection of additional words. Soon he had generated a short list of words that, if added to a spam message, would guarantee its safe passage into his inbox." [29]

He admits that this was a long process. Basically what he did was he took time to find out which words his Bayesian filter thought were safe and then skewed the probabilities of the spam words inside the email. He basically broke down the probability algorithms. It is possible to stump these filters, but as for today, it is a very beneficial software program that I would recommend to everyone to prevent spam from entering their inbox.

4. REFERENCES

[1] Active Web Hosting. "Spam Keywords To Add To Your Filter Lists" http://www.activewebhosting.com/faq/email-filterlist.html Viewed: 3/21/06.

[2] Bradley, Tony. CISSP-ISSAP, Your Guide to Internet / Network Security. "Free Spam-Blocking Software" http://netsecurity.about.com/od/emailspamblocking/a/aafreespam.htm Viewed: 3/1/06

[3] Chirita, Paul-Alexandru., Diederich, Jörg., Nejdl, Wolfgang. "MailRank: Using Ranking for Spam Detection" in the ACM Press of October 2005.

[4] Evett, Don. "Spam Safety Tips" ©2005 TopTenREVIEWS, Inc.http://spam-filter-review.toptenreviews.com/spam-safety-tips.html Viewed: 3/1/06

[5] Graham, Paul. "A Plan For Spam" In August 2002 of http://www.paulgraham.com/spam.html Viewed: 3/19/06

[6] Graham, Paul. "Better Bayesian Filtering" In January 2003 of http://www.paulgraham.com/better.html Viewed: 3/20/06

[7] Graham, Paul. "Probability" http://www.paulgraham.com/naivebayes.html Viewed: 3/29/06

[8] Graham, Paul. "Stopping Spam" in August 2003 of http://www.paulgraham.com/stopspam.html Viewed: 2/12/06.

[9] Hershkop, Shlomo, Stolfo, Salvatore J. "Combining Email Models for False Positive Reduction" in the ACM Press of August 2005.

[10] Krawetz, Neal. "Anti-Spam Solutions and Security." In 2/26/04 of http://www.securityfocus.com/infocus/1763 Viewed: 2/12/06.

[11] LuxContinent LLC © 2004-2006. March 28, 2006 http://www.spam-reader.com/ Viewed: 3/31/06.

[12] Sahami, M., Dumais, S., Heckerman, D., Horvitz, E. *A Bayesian approach to filtering junk e-mail*, AAAI'98 Workshop on Learning for Text Categorization, 1998.

[13] Sipior, Janice C., Ward, Burke T., Bonner, P. Gregory. "Should Spam be on the Menu?" in the ACM Press of June 2004.

[14] SoftLogica LLC. Copyright © 2003-2006, All rights reserved. http://www.outlook-spam-filter.com/ Viewed: 3/31/06.

[15] Spam Bully Copyright 2002 – 2005 Axaware, LLC. http://www.spambully.com/features.php Viewed: 3/31/06.

[16] "Spam Cost Calculator" Copyright © 2006 Barracuda Networks http://www.barracudanetworks.com/ns/resources/spam_cost_calculator.php Viewed: 3/1/06

[17] SpamBayes Project. 2/16/2005. http://spambayes.sourceforge.net/ Viewed: 3/31/06.

[18] SpamBayes Project. 2/16/2005. http://spambayes.sourceforge.net/background.html Viewed: 4/4/06.

[19] Sun Microsystems. Copyright 1994-2006 Sun Microsystems, Inc. http://java.sun.com/products/javamail/ Viewed: 3/1/06.

[20] Sun Microsystems. Copyright 1994-2006 JAF Downloads. http://java.sun.com/products/javabeans/glasgow/jaf.html Viewed: 3/22/06.

[21] Sun Microsystems. "Java Mail API Design Specification." September 2000. http://java.sun.com/products/javamail/JavaMail-1.2.pdf Viewed: 3/19/06

[22] Sun Microsystems. Copyright 1994-2006 Sun Microsystems, Inc. "Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification" http://java.sun.com/j2se/1.4.2/docs/api/index.html Viewed: 3/21/06.

[23] Sun Microsystems. Copyright 1994-2006 Sun Microsystems, Inc. "JavaMail API documentation" http://java.sun.com/products/javamail/javadocs/index.html Viewed: 3/21/06.

[24] Sun Microsystems. Copyright 1994-2006 JavaMail Downloads. http://java.sun.com/products/javamail/downloads/index.html Viewed: 3/22/06.

[25] The Office Maven copyright © 2003-06 - All rights Reserved. http://www.theofficemaven.com/junkout/download.html Viewed: 3/29/06.

[26] The Office Maven copyright © 2003-06 - All rights Reserved. http://www.theofficemaven.com/junkout/features.html#3_Bayesian_Filtering Viewed: 3/29/06.

[27] U.S. Department of Education. National Center for Education Statistics. National Forum on Education Statistics. *Weaving a Secure Web Around Education: A Guide to Technology Standards and Security*, NCES 2003-381. Washington, DC: 2003.

[28] University of Washington. "The IMAP Connection" http://www.imap.org/ Viewed: 3/19/06.

[29] Ward, Mark ."How to Make Spam Unstoppable" BBC News. 2/4/2004 http://news.bbc.co.uk/1/hi/technology/3458457.stm Viewed: 4/4/06.

[30] Webopedia Computer Dictionary. "POP" http://www.webopedia.com/TERM/P/POP2.html Viewed: 3/19/06.

[31] Webopedia Computer Dictionary. "SMTP" http://www.webopedia.com/TERM/S/SMTP.html Viewed: 3/19/06.

[32] Wikipedia: The Free Encyclopedia. "Bayesian Spam Filtering" Last Modified: 17 March 2006. http://en.wikipedia.org/wiki/Bayesian_spam_filtering. Viewed: 3/19/06.

Automated Labeling of a Segmented Image using JESIA

Chris Lohfink
Department of Computer Science
Winona State University
Winona, MN 55987
cnlwsu@gmail.com

ABSTRACT

A Java-based Expert System for Image Analysis (JESIA) is being developed at Winona State University for building a knowledge-based image analysis system. It is capable of segmenting an image using a Fuzzy C-Means (FCM) algorithm. This paper describes a method for automatically labeling centroids of one of these segmented images. The centroids are labeled by comparing a segmented image to an annotated image generated by a human expert, which is then used to find the feature space. This feature space can then be used to label centroids in another segmented image without the annotated image.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Machine Learning
I.5 [**Pattern Recognition**]: Image Segmentation

General Terms

Algorithms, Image Segmentation, Expert System.

Keywords

JESIA, Image Segmentation, clusters, Fuzzy C-Means.

5. INTRODUCTION

Representing data with clusters has been around for awhile. The area was researched by Duran and Odell [10] and by Diday and Simon [12] and Michalski [11]. The use of centroids is a widely accepted approach of representing these clusters [1]. This method allows for easier understanding and processing in both symbolic and conceptual clustering [2]. The clustering being used in image segmentation was documented in Schachter et al. [13].

According to Greg Hamerly, the Fuzzy C-Means (FCM) algorithm is one of the best segmentation algorithms currently available [4]. But a problem that comes up with image segmentation using clusters is the high computational complexity [1]. The performance of a FCM algorithm for image segmentation is also affected by the number of clusters in the data, uneven distribution of data points, initialization of a clustering algorithm, large variations of cluster's sizes, and the shape of clusters, etc.

Proceedings of the 6th Winona Computer Science Undergraduate Research Seminar, April 20–21, 2006, Winona, MN, US.

[3]. A. K. Jain also suggests that the segmentation like this may create non-isotropic or elongated objects [1].

The FCM algorithm works using fuzzy logic. Each datum has a degree e of belongingness to each cluster. The degree of belongingness is represented by a membership function μi given for a point x.

$$\forall x \quad \sum_{k=1}^{\text{num. clusters}} u_k(x) = 1. \qquad (1)$$

The membership function can be defined as :

$$\mu i(x) = \frac{1}{\sum_{j=1}^{numclusters} \frac{d(x,i)}{d(x,j)}^{\frac{2}{m-1}}} \qquad (2)$$

Where d(x,i) represents the distance from x to the centroid i. and is computed as:

$$d(x,i) = | f(x) - cluster(i) | \qquad (3)$$

Where cluster(i) is the value for a i, cluster and f(x) is the value of a point x. These values come from images as grey value on grey scale images [2]. The values of the cluster centroids are thus computed based on the colors between areas of the images..

2. HYPOTHESIS

An algorithm can be developed for the labeling of fuzzy clusters generated from the segmentation of an image under the guidance of a human expert.

3. THE ALGORITHM

The labeling of a cluster starts by comparing a cluster to an annotated training image created by a human expert. If there are enough of the points in a cluster mapped to an annotated object, the cluster will be labeled. As there is a chance that an object may actually be separated into multiple clusters, so it is important for an algorithm to be able to combine multiple clusters and label them belonging to a single object.

Each point is compared to the human expert annotated image. The algorithm stores the information found by this comparison as given by Figure 1.

```
Type Record :
  Var fcmId  : Integer
       { value FCM gave for this cluster }
  Var totalPointsCluster : Integer
       { total number of points in this
        cluster }
  Var positivePoints : Integer
       { number of points in a expert
        annotated object }
  Var unknown : Integer
       {number of points not found in any
        annotated object}
  Var label : Integer
       { label given after the comparison }
```

Figure 1: Structure for data stored for processing in algorithm

```
Type Pixel :
  Var x : Integer
       { a coordinate used to represent
         horizontal position }
  Var y : Integer
       { a coordinate used to represent
         vertical position }
  Var location : Integer, Integer
       {x,y coordinate pair representing
        location}
  Var value : RGB
       { the RGB value of this pixel }
```

Figure 2: structure of a pixel, or point on the image

Within Figure 1, *fcmId* is an integer the FCM algorithm assigns to a cluster. The value of *totalPointsCluster* is an integer of the total number of points contained in this cluster. The value of *positivePoints* is the number of points that fit into an annotated region. The value of *unknown* is the number of points that do not belong to any regions of interest. The *label* is a new label assigned to the cluster given; the cluster has its majority points fall into a region of interest. The value will be outside the range the FCM algorithm used for its labeling but is not terribly relevant beyond that, just a detail for implementation. The Records will be stored in a data structure (e.g. a hash table) that can perform quick lookups for each pixel. How points are defined as pixels within the algorithm is shown in Figure 2. The x, and y values represent the location on the image. This is also given as a location which gives the x and y coordinates respectively. The RGB value is the value given to each of the bands combined. Although using only three bands in the image is not a requirement.

```
1.Procedure label( image: Array of Pixel,
   expertImage : Array of Pixel, mark: Integer )
2.      Var recordList : Array of Record
3.      For each Pixel p in image
        { set the id = to the label the FCM
                  gave it }
4.      recordList[ p.RGB ].fcmId = p.RGB
5.      If expertImage pixel at p.location
           is not background color
           { increment the number inside the
             experts annotated object }
6.      recordList[ p.RGB ].positivePoints ++
7.      Else {
           increment the number outside the
            experts annotated object }
8.      recordList[ p.RGB ].unknown += 1
        { increment number of pixels to that
          id }
9.      recordList[p.RGB].totalPointsCluster++
10.     For each Record r in recordList
11.       If ( r.positivePoints / r.unknown )
                > .75 then
          { this cluster contains enough
            points in the experts annotated
            object to be counted }
12.         Label r, r.fcmId, mark
13          r.label = mark
14.     Else
15.         r.newLabel = 0
```

Figure 3: The pseudo code for the labeling algorithm

This procedure accepts two images and a number for input (see line 1). One image is the segmented image and the other one is created by the expert such that there is a background color, and different colors to annotate where an object is in the original

image, acting as a mask. The algorithm will then examine each pixel in the segmented image. If the pixels corresponding location on the expert image is part of an annotated region, the recordList data structure defined on line two will increment the count for pixels with the same id. If the percent of pixels within a annotated objects mask is greater then a given threshold (in the pseudo code 75% was used in line 11) then all pixels in that centroids will be labeled as part of that object.

Table 1. Result of algorithm described in paper on mask in Figure 4

	1	2	3	4	5
Total points in cluster	157715	25957	42728	8595	158019
Points in annotated region	1676	25957	42728	8032	1323

4. METHOD

This algorithm once implemented in JESIA will be invoked as a JESS script. In order to verify the results data will instead be in the form of pictures of varying complexity. These are taken with a simple digital camera with the three RGB bands. They will be in a form such that the human expert can correctly identify the regions of interest and therefore the algorithm can be used in validation. All the mappings of clusters to annotated regions will be marked in a center file along with the band information.

The images will be read in and data from each cluster contained in the annotated regions will be used to generate a function, which will be applied on a Cartesian plane that uses different color bands as the ordered pair. The function will define a feature space that can be used to identify an object based on the band information provided in the center file.

5. RESULTS

The results vary on the image. With simple backgrounds that don't share similar colors as the object, the algorithm can identify the image with great accuracy.

Figure 4: Mask hanging on white wall

Figure 4 is an image of a mask hanging on a wall, since the mask is of varying colors and has a degree of reflectivity the FCM algorithm broke the mask into two different clusters. The algorithm shown in this paper was able to identify this and group them together. From the expert mask which was done by hand, 100% of the two clusters were contained in the annotated regions as seen in Table 1.

Of the remaining clusters, the third cluster contains 93%. These three centroids labeled as part of the mask can then be used for defining the feature space.

A Cartesian plot of the green and blue components makes this more visible as shown in Figure 5. A simple feature space for this image would be for any centroids where each of the RGB values is less then 150.

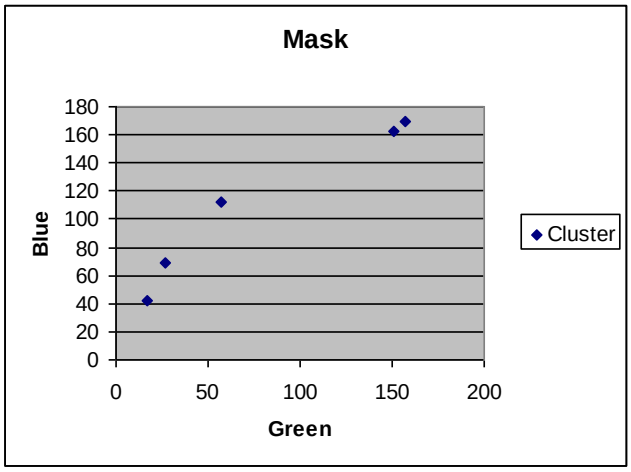

Figure 5: A mapping of the centroids that make it easier to identify feature space.

By running through this many times the feature space can be trained to be used on a raw segmented image to decide if a cluster belongs to the mask or not. Different textures and lighting can affect the results significantly. For an example an image with a textured background and shade has a wider range of colors on the background to confuse with the regions of interest like in the one on Figure 4.

Figure 6: Original picture of a key lying on a carpet

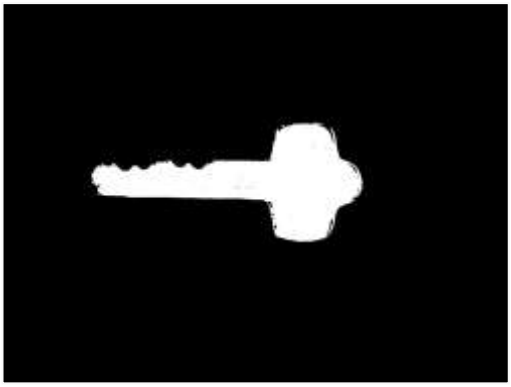

Figure 7: Annotated image of Figure 6

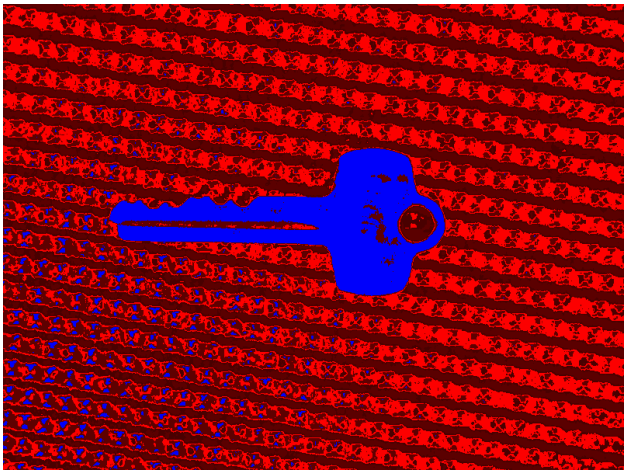

Figure 8: Result of the segmented image and annotated image through the algorithm

	1	2	3	4	5
Total points in cluster	96161	101194	38536	85317	158792
Points in annotated region	3761	663	29967	796	617

Table 2: Results of the Algorithm on Figure 6

In these cases the clusters were fairly easy to identify. Although with some images whose colors are very similar to the background along with lighting infrequencies from glare the results are much worse. Like the case of a key on a polished wooden desk as shown in Figure 9.

The results for the algorithm shown in Table 3 are much worse then those in the previous two. There is no cluster that is inside of an annotated region enough to be considered accurate. This being the case there is no way to obtain a feature space. The best matched cluster is actually just the glare from the table and the key as shown in Figure 10.

Figure 9: Difficult image for the FCM algorithm, region of interest has a lot of similarity with the rest of the image

	1	2	3	4	5
Total points in cluster	44236	27195	19490	8332	26411
Points in annotated region	1062	2612	4255	2619	2144

Table 3: Results of the algorithm on Figure 7

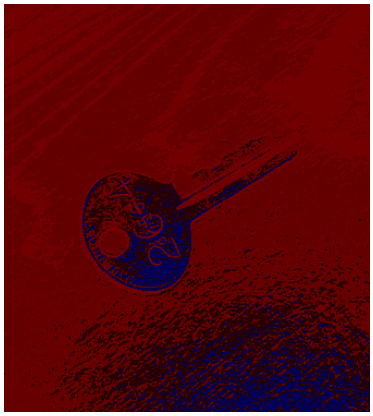

Figure 10: Visual representation of results of segmentation and algorithm on Figure 7. Only glare is different between annotated region and the desk.

6. CONCLUSION

The algorithm works well on specific types of images. Glare, shadow, and other lighting effects can cause many problems for the segmentation. If the region of interest does not contain any significant differences in color then the unknown parts of the image it is hard to expect the FCM algorithm to effectively separate them out and for the algorithm to correctly label them. A solution to this can be re-segmentation. By taking those clusters that belong to part of the region of interest and re-segmenting

them. Through this there is still a chance to obtain enough data to extract a feature space. For image that use many color bands besides the just RGB and taken at higher resolutions, the segmentation has much more information to work with, as does the algorithm in this paper. I believe that with the quality of images like those from SeaWiFS and the amount of information provided in them will be enough such that the FCM will create accurate clusters that are part of the red tide. Providing this, the algorithm should do an accurate job of combining all the data of said clusters into a meaningful file that can be used in finding the feature space, Thereby labeling clusters of a segmented image through training provided by a human expert.

7. ACKNOWLEDGMENTS

Thanks to Dr. Zhang for his help and work on JESIA. Thanks to ACM SIGCHI for allowing us to modify templates they had developed.

REFERENCES

[1] A.K. Jain, M.N. Murty, P.J. Flynn, Data Clustering: A Review, ACM Computing Surveys, Vol. 31, No. 3, September 1999

[2] Bloch Isabelle, FUZZY SETS IN IMAGE PROCESSING, Symposium on Applied Computing archive Proceedings of the 1994 ACM symposium on Applied computing. 1994.

[3] Xuejian Xiong, Kap Luk Chan, Kian Lee Tan, Similarity-Driven Cluster Merging Method for Unsupervised Fuzzy Clustering, UAI 2004, XIONG ET AL. pgs 611-618

[4] Hamerly Greg, Elkan Charles, Alternatives to the k-means algorithm that find better clusterings, CIKM '02, November 4-9, 2002, McLean, Virginia, USA.

[5] Rafael C. Gonzalez, Richard E. Woods, Digital Image Processing, 2nd ed. Prentice Hall, New Jersey, 2002

[6] Zhang Mingrui, Hu Chanmin, Khanal Amit, A knowledge-guided System for Tracking Ocean Color Anomalies.

[7] Zhang Mingrui, Hall Lawrence, Goldgof Dmitry, A Generic Knowledge-Guided Image Segmentation and Labeling System Using Fuzzy Clustering Algorithms, IEEE transactions on Systems, Man, and Cybernetics-Part B, Vol 32, No 5, October 2002Zhang Mingrui, Hall Lawrence, Goldgof Dmitry, Knowledge Extraction and Refinement From Multi-Feature Images Through (Re-)Clustering, JIG Vol. 5, 2000

[8] Masuda Gou, Sakarnoto Norihiro and Kazuo Ushijima, Applying Design Patterns to Decision Tree Learning System, 6th ACM SIGSOFT, 111-120pg, 1998

[9] Duran, B. S. and Odell, P. L. 1974. Cluster Analysis: A Survey. Springer- Verlag, New York, NY.

[10] Michalski, R., Stepp, R. E., and Diday, E. 1981. A recent advance in data analysis: Clustering objects into classes characterized by conjunctive concepts. In Progress in Pattern Recognition, Vol. 1, L. Kanal and A. Rosenfeld, Eds. North-Holland Publishing Co., Amsterdam, The Netherlands.

[11] Schachter, B. J., Davis, L. S., and Rosenfeld, A. 1979. Some experiments in image segmentation by clustering of local feature values. Pattern Recogn. 11, 19-28.

Functionally Testing PHP/MYSQL without Specifications

Jesse Benson
Winona State University
JPBenson4849@winona.edu

ABSTRACT
A special case functional testing tool is implemented to validate the existence of malformed mysql queries due to user input influence. The testing tool processes every filename linked on a website with every combination of parameters the file accepts using the single fault theory. Methods are used to automate the analysis of test case results. The tool is composed of generating, executing, and analyzing units. This tool is tested with 15 test applications and 15 publicly available applications. The test applications were all correctly generated, tested, and analyzed. Nine of the 15 publicly available applications contained malformed mysql queries recognized by the testing tool's analysis.

Categories and Subject Descriptors
D.2.5 [**Testing and Debugging**]: Testing Tools

General Terms
Measurement, Performance, Reliability, Experimentation, Security, Verification.

Keywords
Functional Testing, Web Applications, Test Case Generation, Test Case Execution, Test Case Analysis.

6. INTRODUCTION
PHP Hypertext Preprocessor language was created in 1994 by Rasmus Lerdorf for his personal use of tracking users who visited his site [8]. It later evolved into a more mature language with the addition of mysql and other database support. Mysql [5] is commonly used with PHP, with many publicly available PHP projects utilizing Mysql. Even though it may be used for non web, or even offline projects, PHP was designed for use on the internet, more specifically to develop dynamic webpages.

When implementing freely available PHP applications from the internet there is an issue with the unknown level of quality assurance being offered [10]. One does not know how well the application has been developed and tested. There is no guarantee that the application works correctly, or that any incorrectness will be tested and fixed in the future. This leaves measuring the quality of the application in the hands of those who implement it.

Proceedings of the 6th Winona Computer Science Undergraduate Research Seminar, April 19, 2006, Winona, MN, US.

The traditional software testing methods focus on testers who have full access to the specification and design of a project [6]. They match up test cases to the specifications to make sure a good coverage of cases is used. These methods cannot be used in the case where the tester does not have access to the specifications.

Queries to a Mysql database from a PHP application may not be static. These queries may contain variables which change depending on the user's input. This makes user input validation important for valid queries to be executed. When user input can influence a mysql query, and cause it to be malformed, the software is not correct. These malformed queries may cause incorrect data to be given to the user, or data corruption on the server. To prove that malformed queries are an issue for software correctness a tool has been implemented. The implemented tool will also prove that it is possible to us functional testing to locate and report cases where user input can cause malformed queries.

My hypothesis is that it is possible to identify the existence of certain malformed mysql queries in a web-based PHP application by functionally testing it without the use of software specifications. Furthermore, that it is possible to provide a solution in the form of an automated tool to conduct this testing. This paper covers the background, methodology, measurements, testing setup, results and analysis, and conclusions dealing with an implemented testing tool to test the hypothesis.

7. BACKGROUND RESEARCH
There are only a few tools publicly available to help test PHP applications [2,3,7,12] These tools generally do not require specifications, although how well they test the applications and the amount of human input can be large disadvantages. One related tool is an unnamed project at Stanford which uses static analysis to help find possible mysql issues [12]. This project is not public as of yet, but a paper explaining the algorithm used is available. The authors of this tool announced 99 possible mysql injection flaws they found in popular public PHP applications their tool located. The errors found by Stanford's tool helps to demonstrate the need for quality testing tools for PHP applications.

Implemented was a tool which allows for automatic testing of how mysql queries are constructed with user input influence. In this paper, a malformed query is a query which produces an error message or gives away other unintended information. The problem of having possible malformed mysql queries within a web application is that it produces incorrect results. This often gives information about the database or web server such as usernames to the mysql database or tables and columns existing in the database. More extreme cases could prohibit the injection of mysql logic into the query resulting in otherwise restricted access to the mysql database.

The aim of the tool is to identify these malformed queries so that they may be corrected. The tool may also be used to determine to what extent the problem of incorrect formation of mysql queries are in PHP applications. With the completion of this project vital errors such as mysql injections, as well as error messages generated by PHP or mysql were identified.

8. METHODOLOGY

In order to test the hypothesis we created a tool to automate testing. Experiments were set up an executed to demonstrate the tool's ability to prove the hypothesis correct.

3.1 Testing Tool

This tool has three units. First unit is the web crawler unit [4]. This unit creates a tree with URLs, parameters for each URL, and nominal values for each parameter. The second unit is to generate the test cases. This unit will use the tree constructed by the spidering to generate test cases. Test cases use single fault theory [6] with every permutation of parameters and nominal values [6]. The third unit is the execution of the test cases. Each test case only contains one URL with at least one GET or POST parameter, and makes a single request to this URL. In order to automate the analysis of the test case this unit retrieves the resulting data from a request of all nominal values for the same parameters used in the test case. It then uses this data to check for php error messages in the test case results. An example run of the program is provided in Figure 1. In this run the input is the URL to the start page of the application to be tested. The output is a list of URLs which the tool determined contained malformed queries from the test cases. The tool's complete process is further illustrated in Figure 2.

```
Commandline>testingtool http://127.0.0.1/app1/index.php
  http://127.0.0.1/app1/f.php?firstname=a%27 gave a SQL
  syntax error.
  http://127.0.0.1/app1/results.php?table=a/**/or/**/1=2
  with post data: sortType=2 gave an invalid sql result
  resource error.
```

Figure 1. Example run of testing tool.

From Figure 1 the firstname parameter is used within f.php to aid in construction of the mysql query SELECT username,first,middle,last FROM users WHERE first='$_GET["firstname"]' Setting firstname GET parameter's value to John may receive desired effects. Setting firstname GET parameter's value to a%27 (%27 is the escaped encoding for the character ' [9]) will result in the mysql query to be malformed. When a mysql query is malformed it produces incorrect results. All of the test case payloads used by the tool are listed in appendix A. Each test case attempts to introduce mysql or PHP logic in order to get a mysql query to be malformed.

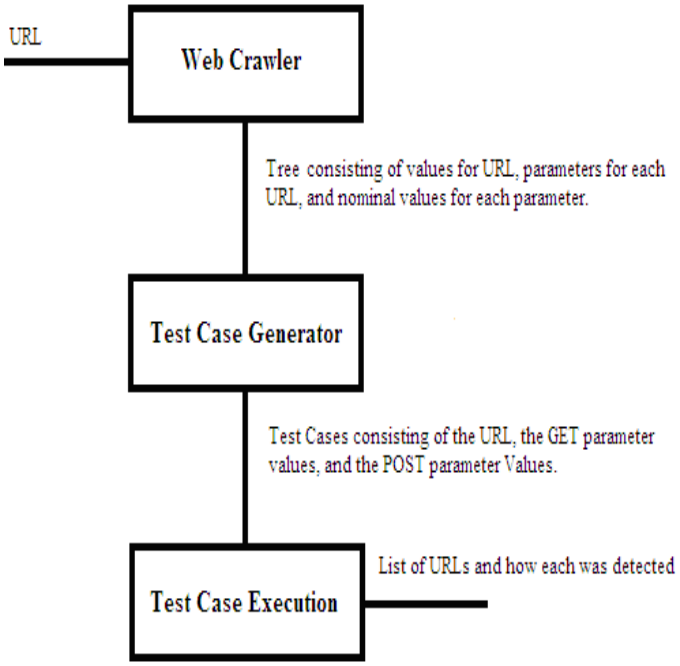

Figure 2. Data flow between the three units which compose the testing tool.

Figure 2 shows how the input to the testing tool goes into the web crawler and is used to follow links and forms in the resulting html to crawl through the web application. During this crawl the tree is made. The tree will be in the form of Figure 3.

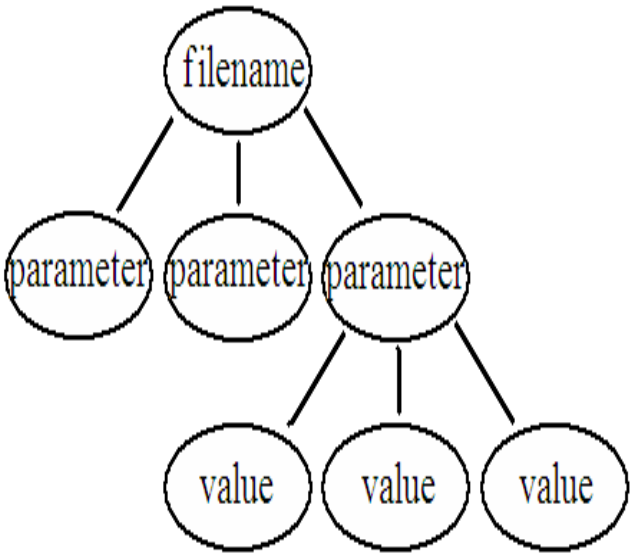

Figure 3. Data flow between the three units which compose the testing tool.

A tree in this form will be sent to the test case generator. Here the tree will be used to create all the test cases. The test cases contain a URL, GET parameters, POST parameters, and expected results. Expected results are gathered at the test case generator unit by getting the results of a request with all parameters set to nominal values. The test case execution unit will get the results from each test case and compare them with the expected output. The results of the test cases are outputted.

8.2 Measurements

The tests focused on determining the existence of malformed mysql queries in PHP code. The measurements taken on the test applications were (A) time (in seconds), (B) number of requests issued. The time (A) and request (B) measurements are of interest in seeing the relationship they have on each other and to verify that the tool worked correctly. The measurements taken for the publicly available applications were (C) number of test cases which identified a malformed query, and (D) if there was a successfully verified malformed query. The (C) and (D) measurements were recorded publicly available applications to check with the hypothesis.

8.3 Testing Setup

All tests were conducted using Windows XP Tablet PC edition with service pack 2, Apache 2.0.46, Mysql version 12.20 distribution 4.0.13, and PHP 4.3.2 (register_globals on; magic_quotes_gpc off) with Zend Engine v1.3.0. The computer had an Intel Pentium M processor at 1.5 GHz with 760 MB of RAM. No other applications were running besides default windows services and tablet services. Although tablets are not commonly used as servers it did not interfere with the testing results significantly.

To test the implemented tool, 15 test applications were written in PHP, and 15 publicly available PHP applications were installed. Of these test applications there was five without mysql use, five with mysql use but without malformed queries, and the other five contained different types of malformed mysql queries based on client input. Each group of test applications had one application with one parameter, one with two parameters, one with three parameters, one with four parameters, and one with five parameters.

4. RESULTS AND ANALYSIS

4.1 Results from the Test Applications

Test application results are from the 15 test applications. These were created for the purpose of testing if the tool can identify malformed queries. There were 40 test cases used. Each test application has two links in the form

testX.php
testX.php?parameters

To get the results of each of these links accounts for 2 requests. These spidering results will be known as A. Requests are made by the tool to save the resulting data from a request using nominal values. These results will be known as B. B is used to compare to the data resulting from the 40 tests (Appendix A). The tests will be known as C. This totals for 43 requests if 1 parameter is being tested, A=2, B=1, C=40. For 2 parameters the same process is done, but with the 40 tests being applied to both parameters. Then the same process is done twice more to test each parameter with the other set to a nominal value. In this case A=2, B=4, C=160.

The following table, Table 1, holds the results from the test applications. All three different types of test applications took the same amount of time (A) and issued the same amount of requests (B). Because of this all three types of test applications are grouped together. The three applications with one parameter are all grouped in the same row. The parameter column is the number of parameters each group had. The time (A) is in seconds, and is directly dependent on the number of requests made (B). The number of requests made (B) column is to demonstrate how the number of parameters is directly proportionate to the number of requests required to execute all the test cases. The time increases a few milliseconds with each additional request.

Table 1. Results of test application testing

Parameters	Time (Seconds)	Requests Made
1	0	43
2	1	166
3	2	494
4	6	1314
5	14	3282

Table 2. Results of publicly available PHP application testing

Application Name	Errors Reported	Errors Verified
BlazeBoard 0.55	508	True
Clevercopy 3.0	123	True
E107 0.7.2	0	False
Interact 2.1	8	True
MG2 0.5.1	0	False
Mybb 1.10	0	False
Papoo 3 beta 1	9555	True
Phpsurveyor 0.993	0	False
Simpog 0.9.2	42817	True
Sylphagora 1.2	0	False
Textpattern 4.0.3	0	False
Thatware 0.4.6	424	True
Tikiwiki 1.9.2	20	True
UniWakka 0.5.2	0	False
Wordpress 2.0.2	0	False

4.2 Results from Publicly Available PHP Applications

Publicly available PHP applications were chosen by popularity (number of times downloaded) and availability (source code was freely available to download). These applications and their source code are available to anyone on the internet.

The Errors Reported column in Table 2 is the number of test cases in which the expected results did not match the results and a malformed mysql query was the cause. This indicates that there is an issue, but does not indicate how many issues there are. For example, with the test on Interact 2.1, in Table 2, there was 8 Errors Reported. During this test 8 of the test cases did not match expected outputs because of a malformed query. After analyzing the code of Interact 2.1 it was determined that there was a single source causing all 8 of the test cases to fail. The Errors Verified column in Table 2 is a boolean value showing if at least one malformed query could be constructed in the web application. This was done through static analysis [11] and functional testing.

5. CONCLUSIONS

From the results of the test applications it is clear to see that the tool was able to make the correct amount of requests in an adequate amount of time for URLs with less than five parameters. It is missing the feature of suspending testing on a parameter once a possible malformed query has been identified. This is what caused the number of test cases whose results did not match the expended results. The experiment also did not show any signs of false positives. This is not proof that the tool would never have any false positives. The tool also issues the correct number of requests and is adequate in its amount of time to complete the tests.

There is an issue with the amount of time it takes to complete a test. If an application was to have a file with a high number of parameters (greater than 10) or if there were a high number of nominal values for any parameter (greater than 10) then this time may reach an inadequate amount.

It is also impossible to determine if all malformed queries were found in the publicly available applications without further research into those specific applications. The fact that some malformed queries were reported and verified does coincide with the hypothesis.

The tool may be improved with more special test cases. More research is required to gather the information to form better test cases. The tool may also be improved if filenames, parameters, and nominals not found during spidering could be tested. This could be in the form of brute forcing URLs, parameters, and nominals, or reading the applications source code. If any source code is read then the tool would become a hybrid testing tool (using both static and functional testing methods).
Overall the tests successfully proved the hypothesis correct. Due to the scale of the problem the tests were very limited. This causes the correctness of the tool to be in question. While it is true that the tool has the ability to locate possible issues with malformed queries, the testing does not prove the tool finds all malformed queries. It also does not prove that all reports of malformed queries are true.

6. ACKNOWLEDGMENTS

Thank you to RFP for the libwhisker framework. Thank you to Stephen Esser, Pokelyzz, Waraxe, James Bercegay, and Infamous41md for their publications dealing with PHP with mysql issues. Thank you to Dr Narayan Debnath for leading me in the teaching of software testing principles.

REFERENCES

[12] B. Michael, F. Juliana, and G. Patrice. Veriweb:automatically testing dynamic web sites. In Proceedings of 11th International WWW Conference, Honulolu, May 2002.

[13] Jason Huggins, Paul Gross and Jie Tina Wang. 2005. Availible at http://www.openqa.org/selenium/

[14] Marcus Baker. 2006. Availible at http://www.lastcraft.com/simple_test.php

[15] Michael Chau , Daniel Zeng , Hinchun Chen, Personalized spiders for web search and analysis, Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries, p.79-87, January 2001, Roanoke, Virginia, United States.

[16] MYSQL AB. 2006. Availible at http://www.mysql.com/

[17] Paul Jorgensen. Software Testing: a Craftsman's Approach. Boca Raton: CRC Press, 1995.

[18] PhpUnit. 2005. Availible at http://phpunit.sourceforge.net/

[19] Stig Sæther Bakken with Zend staff. 2000. Available at http://www.zend.com/zend/art/intro.php

[20] T. Berners-Lee, R. Fielding, U.C. Irvine, L. Masinter. 1998. Request for Comments: 2396.

[21] United States Food and Drug Administration. 2002. Availible at http://www.fda.gov/cdrh/comp/guidance/938.html

[22] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D. Lee, and S.-Y. Kuo. Securing web application code by static analysis and runtime protection. In Proceedings of the 13th International World Wide Web Conference, 2004.

[23] Yichen Xie, Alex Aiken. 2005. Availible at http://glide.stanford.edu/yichen/research/sec.pdf

APPENDIX A

-1%2540

-1%2500

-1%27%20or%20%27a%27=%27b

-1%27%20or%20%27a%27=%27a

-1%22%20or%20%22a%22=%22b

-1%22%20or%20%22a%22=%22a

-1/**/or/**/1=2

-1/**/or/**/1=1

-1%27/**/or/**/1=2

-1%27/**/or/**/1=1

-1%22/**/or/**/1=2

-1%22/**/or/**/1=1

-1%2527%20or%201=2

-1%2527%20or%201=1

-1%2522%20or%201=2

-1%2522%20or%201=1

-1%2527/**/or/**/1=2

-1%2527/**/or/**/1=1

-1%2522/**/or/**/1=2

-1%2522/**/or/**/1=1

-1%20union%20select%201%2f%2a

-1%20union%20select%201,2%2f%2a

-1%27%20union%20select%201%2f%2a

-1%27%20union%20select%201,2%2f%2a

-1%22%20union%20select%201%2f%2a

-1%22%20union%20select%201,2%2f%2a

-1%2527%20union%20select%201%2f%2a

-1%2527%20union%20select%201,2%2f%2a

-1%2522%20union%20select%201%2f%2a

-1%2522%20union%20select%201,2%2f%2a

-1/**/union/**/select/**/1%2f%2a

-1/**/union/**/select/**/1,2%2f%2a

-1%27/**/union/**/select/**/1%2f%2a

-1%27/**/union/**/select/**/1,2%2f%2a

-1%22/**/union/**/select/**/1%2f%2a

-1%22/**/union/**/select/**/1,2%2f%2a

-1%2527/**/union/**/select/**/1%2f%2a

-1%2527/**/union/**/select/**/1,2%2f%2a

-1%2522/**/union/**/select/**/1%2f%2a

-1%2522/**/union/**/select/**/1,2%2f%2a

An Automated Tool for Computing Software Metrics

Nripendra Rai
Department of Computer Science
Winona State University
Winona, MN – 55987
(507) – 313 – 8311
nripendrarai@gmail.com

ABSTRACT

This paper describes an automated tool that produces a set of software metrics for an application program written in Java. The tool computes software metrics such as Lines-of-code metrics, Halstead's metrics, McCabe's cyclomatic number, and maintainability index along with its functionalities. The main focus of the paper is on design and implementation of the tool. The results collected during the testing of the tool are analyzed.

General Terms

Algorithms, Measurement, Documentation, Design, Experimentation, and Theory.

Keywords

Java Automated Tool, Software Metrics, Measurement, LOC, Halstead Metrics, McCabe's Cyclomatic Metrics, Maintainability Index.

1. INTRODUCTION

Measurement is a major factor in software development since developing software is a costly process. Most software projects are large, complex, and may have many quality problems [11]. The measured software characteristics may help developers to make decisions. The commonly used characteristics in measuring software complexity are called software metrics [3, 6, 11]. They are Lines-of-codes metrics, Halstead metrics, McCabe's cyclomatic number, and maintainability index [1, 2, 8, 12, 13]. This paper is about designing and implementing an automated tool that produces a table of software metrics for an application program. The whole project follows the software engineering steps of development cycle which are specification, design, implementation, test, and analysis [11].

Proceedings of the 6th Winona Computer Science Undergraduate Research Seminar, April 19, 2006, Winona, MN, US.

The designed tool takes syntactically correct Java programs and generate direct measures of lines of codes [1, 9, 12], number of operators [2, 12], number of operands [2, 12], number of occurrences of operators and operands [2, 12], program length [9, 12], computed time [2, 9], and many others. These measures are further used to calculate the various advanced software metrics that includes difficulty level, estimated number of errors in the program, program level, program volume, effort to implement, McCabe's Cyclomatic Number, Maintainability Index without comments, Maintainability Index comment weight, and Maintainability Index [13]. While developing software, software developers or managers come across different kind of approaches and they have to make a decision in such a way that their goal can be achieved as planned. During that time, an automated tool will help comparing different aspects of approaches such as efficiency, error rate, cost, and time so that decision can be made. Further-more, programmers may use the obtained metrics to improve the quality of the program [1]. Hence, the paper will be helpful to those who are interested in the field of software engineering.

The paper continues with background research on software metrics tools followed by the definitions of software metrics measured by the designed tool. Then experimentation and design for the automated tool is given. Finally, the result and analysis part shows the output of the tool and its evaluation along with the conclusion as well as future concerns.

2. BACKGROUND RESEARCH

To my best knowledge, few automated tools [10] have been developed for the purpose of computing software metrics. One of the few tools is Testwell CMT Java developed by Verify-soft Technology [13] that analyzes the static complexity of software written in Java using software metrics. The metrics calculated by CMT Java are industrial standards established in research projects during several years. The metrics computed by CMT Java are as follows: Lines-of-codes (LOC) metrics, McCabe Metrics, Halstead's Metrics, and Maintainability Index.

CMT Java is available on many platforms including Windows and several UNIX environments. One or many files can be run at a time to calculate metrics and CMT Java support different types of file formats such as text, HTML, Excel CSV, and XML. After analysis is done, CMT Java

generates reports using one of the above mentioned file formats. The output file will include the metric values for each selected file and summary results where the calculated metric values are compared with the alarm limits [13]. Similarly, there have been few other tools mentioned theoretically that computes different software metrics such as Lines-of-codes, cyclomatic complexity, effort measure [4], operators, operands lists, their frequency of occurrences, number of terminators, different types of control structures, internal and external variables [1], and Halstead metrics [2]. Moreover, most of the tools are written for languages other than Java.

3. METHODOLOGY

3.1 Software Metrics - Definitions

This section introduces the metrics that are to be computed by the designed automated tool of this project. The metrics considered in the project are Lines-of-codes metrics, Halstead metrics, McCabe's cyclomatic number, and maintainability index.

Table 1. Halstead Metrics

Metrics	Equation	References
num of unique operators (n1) and operands (n2)		[9, 12]
total occurrences of operators (N1) and operands (N2)		[9, 12]
The vocabulary of the program (n)	n = n1 + n2	[7, 9, 12]
The length of the program (N)	N = N1 + N2	[7, 9, 12]
The volume of the program (V)	V = N * log2(n)	[7, 9, 12]
The level of the program (L)	L = (2/n1)*(n2/N2)	[2,12]
The difficulty of the program (D)	D = 1/L	[9, 12]
The effort of the program (E)	E = V/L or E = V * D	[2, 4, 9, 10, 12]
The computed time (T)	T = E / 18	[9, 12]
The number of delivered bugs (B)	B = (E **(2/3))/3000	[9]

3.1.1 Lines-of-codes metrics

Lines-of-codes metrics are the most traditional and simplest metrics that are used to measure software complexity [9, 13]. Lines-of-codes metrics which are to be computed are the lines of executable codes and the number of commented lines [4, 13]. The percentage of commented lines should be between 30 percent and 75 percent of total number of lines. If the percentage of commented lines is less than 30 percent then the program is poorly explained. On the other hand, if the percentage of commented lines is more than 75 percent then the file should be considered as a document not a program [13].

3.1.2 Halstead metrics

Halstead metrics were developed by the late Maurice Halstead with a purpose to determine a quantitative estimate of complexity directly from the operators and operands in the module from its source code [13]. Halstead metrics are the earliest software metrics and are considered as strong indicators of code complexity [7, 13]. Halstead metrics is totally based upon the number of operators and operands [2]. An Operator is any symbol or keyword group in a program that specifies an algorithmic action of the computer [11]. An Operator consists of any reserved words that specify storage and qualify type along with logical, arithmetic, and relational operators [13]. Similarly, an operand is any symbol that represents data [11]. An Operand consists of identifiers, name types, type specifiers, and constants [13]. Halstead metrics are summarized in Table 1.

The number of unique operators (n1) and operands (n2) are calculated by collecting the frequencies of each operator and operand token of the source program [11, 13]. The total occurrences of operators (N1) and operands (N2) are calculated by counting the total number of operators and operands [11, 13]. Vocabulary (n) is the sum of the number of unique operators and operands [7, 11, 13]. Length of the program is the sum of the total number of operators and operands [7, 11, 13]. Volume (V) describes the size of the program [7]. V is calculated based on the number of operations performed and operands handled in the algorithm. Volume of a program should be between 100 and 8000. If Volume exceeds 8000 then that means the program has functions that do too many things [13].

Similarly, Level of the program (L) is calculated using the number of unique operators and operands along with the total occurrence of operands. Difficulty (D) or error proneness is inversely proportional to Level of the program [11, 13]. Effort (E) is correlated to the software errors [2, 4, 8, 11, 13]. Computed Time (T) is the time in seconds to implement the program and is directly proportional to Effort [11, 13]. Number of Delivered Bugs (B) represents the overall complexity of the program. B is an estimate for the number of errors in the implementation and should be less than 2 [13].

3.1.3 McCabe's Cyclomatic Number

McCabe's Cyclomatic Number was introduced by Thomas McCabe in 1976 and is considered a broad measure of

soundness and confidence for a program [13]. Cyclomatic Number is the most widely accepted metric that helps estimate the clarity and maintainability of a software and defined as a function of the number of predicates in the program [4, 9, 11]:

V = Number of decision points (predicates) + 1

The decision points or predicates are reserved words such as if, while, for, do, switch, &&, ||, etc [7]. The cyclomatic number should be less than 15 otherwise the program is hard to identify and test [13].

3.1.4 Maintainability Index

Maintainability Index (MI) helps to reduce or reverse a system's tendency toward code entropy or degraded integrity and also indicates when to rewrite the code instead of changing it. Maintainability Index can be defined as a single number value to estimate the relative maintainability of the code. The calculation of Maintainability Index is based upon Lines-of-codes metrics, McCabe's metrics, and Halstead metrics. MI with value 85 or more means that the program has good maintainability. MI between 65 and 85 indicates the program with moderate maintainability. And MI with value less than 65 means the program has really bad pieces of code and is hard to maintain. MI has two components. They are maintainability index without comments (MIwoc) and maintainability index comment weight (MIcw). MIwoc and MIcw can be represented as follows:

$$MIwoc = 171 - 5.2 * \ln(aveV) - 0.23 * aveG - 16.2 * \ln(aveLOC)$$

$$MIcw = 50 * \sin(\sqrt{(2.4*perCM)})$$

where aveV is Average Halstead's Volume per Module, aveG is Average Cyclomatic Number, and aveLoc is Average Lines of Codes.

These two components MIwoc and MIcw are summed up together to get MI [13].

3.2 Experimentation

In order to test the automated tool, a set of G-string programs written in Java was collected from CS 410 Software Engineering class. The G-string programs were provided by Dr. Narayan Debnath, one of the professors of Winona State University. Dr. Debnath states that a G-string program takes a G-string as an input. The G-string consists of symbols (a-z) and the special symbol +. It can be further defined as:

a) Any symbol (a-z) is a G-string.
b) Given two elementary G-strings, say a and b, +ab is a G-string.
c) Given any two arbitrary G-strings, say G1 and G2, +G1G2 is a new G-string.
d) A valid G-string is only those constructed following the rules from (a) to (c) [11].

A G-string program scans the input string from right to left, and produces all possible substring(s) specified as follows:

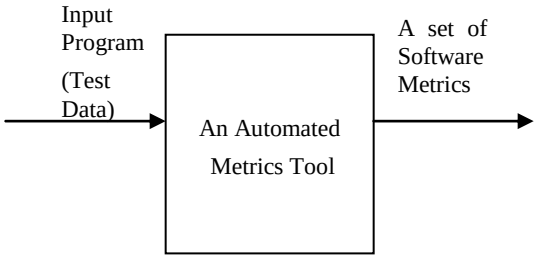

Figure 1: The Basic Design

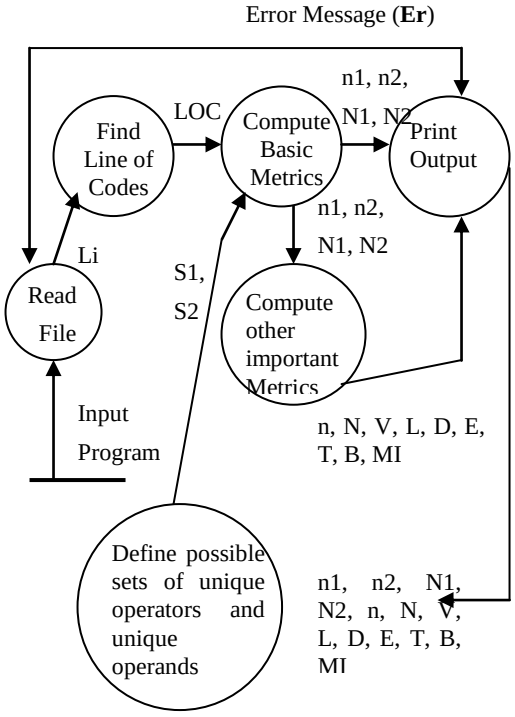

Figure 2: An Abstract Design

Given a G-string, the possible substrings are of the form + S1 S2, where S1 and S2 are single alphabetic characters (a through z). If the input string is invalid, a G-string program will produce an error message along with any partially processed substring(s). A set of eight different G-string programs was processed by the automated tool. The automated tool was developed in Java using Eclipse as an IDE (Integrated Development Environment) under Windows platform.

3.3 Design

The automated tool for software metrics was designed using Top-Down Modeling approach. The Top-Down Model is an approach that starts at the highest level of abstraction and goes down towards the lowest level providing details [11].

4. RESULTS AND ANALYSIS

The output from the tool was a table of software metrics consisting of total lines of codes (executable), total lines of commented lines, total number of predicates, cyclomatic number, all the metrics from Table 1, maintainability index without comments, maintainability index comment weight, and maintainability index. Analysis was done on different categories which are as follows:

Table 2. LOC metric and Cyclomatic Complexity

Test Data	LOC metric	Cyclomatic Number	Program Relation
1	63	9	Agree
2	42	11	Disagree
3	133	26	Agree
4	77	3	Disagree
5	49	3	Agree
6	40	2	Agree
7	60	12	Disagree
8	38	7	Disagree

Table 3. Vocabulary and Volume Metrics

Test Data	Vocabulary	Volume	Program Relation
1	294	4217.216	Agree
2	172	1585.42	Agree
3	143	1717.144	Disagree
4	88	707.4192	Agree
5	61	374.08952	Agree
6	128	980.11011	Agree
7	183	2552.64	Agree
8	125	956.0061	Agree

In Table 2, LOC metric of testData1, 3, 5, and 6 agree with its corresponding cyclomatic number since the number of decision points matches with the number of executable lines of codes. In case of testData2, 4, 7, and 8, there are more decision points where as the number of executable lines of codes is less. This means the programs are hard to identify and test, especially testData3 has Cyclomatic Number 26. The program relation shows that LOC metric and Cyclomatic Number of 4 test data agree with each other where as other 4 did not agree.

In Table 3, Vocabulary and Volume of all the test data are matched up. Individually, all the volumes are in the limit range i.e. 100 to 8000. However, testData3 does not agree when it comes to Vocabulary and Volume together. The volume of testData3 is little bit higher as compared to other test data where as Vocabulary looks fine. Again, the program relation for this table clearly shows that 7 out of 8 test data agreed when it came to Vocabulary and Volume.

Table 4. Difficulty and Effort Metrics

Test Data	Difficulty	Effort	Program Relation
1	24.82472	104691.2	Disagree
2	19.05479	30210.01	Agree
3	19.15966	32899.90	Agree
4	11.97183	8469.1032	Agree
5	6.97959	2610.992	Agree
6	12.5504	12300.83	Agree
7	28.7547	73400.67	Agree
8	16.75	16013.102	Agree

Table 5. Number of Delivered Bugs and Maintainability Index

Test Data	Num of Delivered Bugs	Maintainability Index	Program Relation
1	0.73474	108.39	Disagree
2	0.32111	119.59	Agree
3	0.33988	97.052	Disagree
4	0.13766	115.80	Agree
5	0.06287	126.443	Agree
6	0.17651	124.95	Agree
7	0.58000	111.106	Disagree
8	0.21040	124.763	Agree

In Table 4, Difficulty of testData1 does not agree with its Effort. Otherwise, all other test data has agreeable Difficulty and Effort. Although the error proneness or difficulty of testData1 is little higher as compared to other test data, effort seems to be higher than it should be. In total, all the 7 test

data agreed with each other according to the program relation.

Table 5 shows the overall complexity of the programs which was defined by Number of Delivered Bugs which seemed to be relatively low for all the programs. Also, all the input programs had good maintainability with Maintainability Index more than 85. When both Number of Delivered Bugs and Maintainability Index are compared together, testData1, 3, and 7 do not seem to agree. Maintainability Index is lower than it should be since their complexity is higher than other programs. Over all, only 3 out of 8 test data failed to agree with each other.

Table 6. Time Comparison

Test Data	Computed Time (Mins)	Actual Time (Mins)	Program Relation
1	96.93	1,485	Disagree
2	27.97	1,675	Disagree
3	30.46	555	Disagree
4	7.8417	630	Disagree
5	2.4175	240	Agree
6	11.389	380	Agree
7	67.963	330	Agree
8	14.826	320	Agree

Computed Time in Table 5 is the time it took to do the implementation and is calculated by using the other metrics. Comparing Computed Time with Actual Time provided by students, the difference was much higher than expected. Some of the test data, testData1-4, took much more time than it should have. Again, in case of testData5's Computed Time, it seems confusing that it should only take 2.4175 minutes to implement. It should be obvious to have around 200 – 300 minutes difference because Computed Time is the time taken to design and implement where as Actual Time for writing the algorithm, designing, coding, and testing. However, 4 out of 8 test data agreed with each other in terms of Actual Time and Computed Time.

6. CONCLUSIONS

The designed automated tool computed different software metrics such as Lines-of-codes metrics, McCabe's Cyclomatic Number, Halstead metrics, and Maintainability Index as expected. A set of java programs, G-String programs, was used as inputs and a table of software metrics was created for analysis. Analysis was done based on human judgment and some restrictions provided along with the definitions. Some of the test data proved to be efficient than others. For example, testData5 had lesser Lines-of-codes, only 3 decision points, lower Length and Volume, and higher Maintainability Index.

The automated tool clearly distinguished the better and efficient program among 8 different programs that had the same functionality. Hence, the program relation on each of the output tables showed clearly that the tool worked correctly since the percentage of agreement was 50% (LOC metric and Cyclomatic Number), 87.5% (Vocabulary and Volume), 87.5% (Difficulty and Effort), 62.5% (Number of Delivered Bugs and Maintainability Index), 50% (Computed Time and Actual Time). This paper can further be continued by adding more metrics to compute that explains more about software complexity. Also, it would have been clearer to define complexity if we could come up with standard limits for all the above mentioned metrics. Object Oriented metrics might be one good choice to add. But, there are a lot of metrics emerging in the field of software engineering which opens the door for researchers to investigate more and find more ways to estimate software complexity.

7. ACKNOWLEDGEMENT

Special thanks to Dr. Narayan Debnath, Department of Computer Science, Winona State University, Winona, Minnesota.

REFERENCES

[1] Aggarwal, K. K. A Tool for the Collection of Industrial Software Metrics Data. Guru Jambheshwar University: Department of Computer Science and Engineering, Hisar Haryana, India, 1997.

[2] Bailey, C. T., & Dingee, W. L. A software study using Halstead metrics, in Proceedings of the 1981 ACM workshop/symposium on Measurement and evaluation of software quality (January 1981), Volume 10, Issue 1, 189 - 197.

[3] Dandashi, F. Software engineering: theory, application and practice: A method for assessing the reusability of object-oriented code using a validated set of automated measurements, in Proceedings of the 2002 ACM symposium on Applied computing (Madrid, Spain, March 2002), 997 - 1003.

[4] Debnath, N.C., Lee, R. Y. & Abachi, H. R. An analysis of software engineering metrics in OO environment, IEEE/ACS International Conference on Computer Systems and Applications (Beirut, Lebanon, 2001), 492 - 494.

[5] Dumke, R., Neumann, K., & Stoeffler, K. The metric based compiler: a current requirement, ACM SIGPLAN Notices, Volume 27, Issue 12 (December 1992), 29 - 38.

[6] Fenton, N. E. & Neil, M. Software metrics: roadmap, in Proceedings of the Conference on The Future of

Software engineering (Ireland, May 2000), ACM, 357 - 370.

[7] Hilda, K. B. A study of software metrics. The State University of New Jersey: Department of Electrical and Computer Engineering, New Brunswick, New Jersey, May 2003.

[8] Leach, R. J. Using metrics to evaluate student programs, ACM SIGCSE Bulletin, Volume 27, Issue 2 (June 1995), 41 - 48.

[9] Mathias, K.S., Cross, J. H., Hendrix, T. D., & Barowski, L. A. The role for software measures and metrics in studies of program comprehension, ACM Southeast Regional Conference (1999).

[10] Niwot Ridge Resources. Metrics Tools [Online] 2005; Available from http://www.niwotridge.com/Resources/PM-SWEResources/MetricsTools.htm. Accessed Feb 12, 2006.

[11] Pressman, R. S. Software Engineering: A Practitioner's Approach. New York: McGraw-Hill, 2001.

[12] Scotto, M., Sillitti, A., Succi, G., & Vernazza, T. A relational approach to Software metrics, in 2004 ACM symposium on Applied computing (Nicosia, Cyprus, March 2004), 1536 - 1540.

[13] Verify soft Technology. CMT Java - Complexity Measures [Online] 2005; Available from

http://www.verifysoft.com/en_cmtjava.html.

Accessed Feb 15, 2006.

The Correlation of Immersion and User Satisfaction in Video Games

Matthew Knutson
Saint Mary's University Computer Science Student
700 Terrace Heights #724
Winona, MN 55987
(507) 457-7085
mjknut02@smumn.edu

ABSTRACT

This paper describes the correlation between immersion in video games and player enjoyment. A commonly accepted definition of immersion in the context of video game playing is established and used. Surveys given before and after game-play were used as a primary method of gathering data. Two surveys were given to participants prior to game-play, participants played a game for an hour, and a final survey was given. The data from the surveys was used to show relationships between immersion factors and player enjoyment. Such correlations could provide new and more effective methods of video game development and testing.

Categories and Subject Descriptors

D.2.8 [Metrics]: Performance measures

K.8.m [Personal Computing]: Miscellaneous

General Terms

Measurement, Documentation, Performance, Design, Experimentation, Human Factors

Keywords

Immersion, Absorption, Video, Game, Engagement, Sense, Presence

1. INTRODUCTION

In this project we define immersion simply as a sense of presence, the sense of one being in a fictional world outside of the real one. The definition follows Emily Brown and Paul Cairn's [3] recently published work in determining a grounded structure to immersion from the myriad of different ideas. A common example would be a horror movie patron who becomes immersed in the movie and jumps off of the seat during unexpectedly frightful moments. This is not to say, however, that every person experiences immersion the same way, or even the same amount. Immersion is known as a very subjective experience [2-11].

Video games are a billion dollar industry in which producers like Atari Inc. are making millions of dollars every year by making products with little real-world value other than user satisfaction [1]. Just as for any programmers, video game programmers depend on strong designs and heuristics to produce profitable games. There are dozens of different heuristics and tests currently in use for game evaluation; however, all heuristics that I have seen undervalue the idea of player absorption (immersion) [4, 9]. I believe that this is a critical oversight, as immersion could be major factor in user satisfaction and good game design.

One example of common game evaluation comes from an anonymous game developer in a study by Penelope Sweetser and Peta Wyeth [9]. The evaluation consisted of three main areas: game mechanics including environment physics and realistic reactions, game interface including ease of use and overall transparency of controls, and game play including winnability, art and sound effects, and rewards. Several evaluations like this list elements of immersion (ex. 'user feels a part of the game' or 'user loses sense of time') as small relevant factors, but none are giving the matter enough weight.

I have found a relationship in video game play between satisfaction and immersion in that the more a part of the game the user feels the more positive that user will feel about the entire game experience. This means that a user feeling as though he/she is inside the game 'world' will have a more enjoyable experience than a user who does not. To gather evidence, a user test was conducted involving 17 participants playing a game and filling out surveys to establish game preference, satisfaction, and immersion.

Proceedings of the 6th Winona Computer Science Undergraduate Research Seminar, April 19, 2006, Winona,

The data showed some strong relationships between player satisfaction, immersion, and the amount of time a user would wish to play that game at one sitting (without other time constraints). Other relationships were also noted. In the conclusions, I detail the likely use of my findings to increase the play-time, replay value, and likelihood of purchase for video games through better video game design heuristics and testing.

This paper continues with a small discussion of the current status of immersion research. Following this discussion the methods I used are detailed, including specific questions from the surveys, and some of the noted limitations of the study. Once the methodologies are described I conclude by talking about some of the noticed relationships and possible uses of this information.

2. BACKGROUND RESEARCH

Researchers [2, 3, 6-8, 10] in both the science and psychological fields have begun to link the phenomena of game absorption/immersion with a sort of subconscious openness. Psychologists [6] study the emotional responses and effects of violent video games with children. In their research, children showing higher levels of immersion (only one of the factors studied) seemed to have more of a reaction – they were affected more by the games. Ravaja and Salminen, et al. [8] studied emotional response to video gaming. They concluded that immersion might result in a user losing some sense of self, caring less about normal morals/norms, and being generally more open to connect what that user is doing directly with how they are feeling. For example, an absorbed user who is playing a video game with violence may connect the happiness of playing it directly with that game even if that user normally disapproves of violence.

Studies show immersion occurs in increasing levels [3]. A person can experience small, moderate, or high levels of immersion. The first level of immersion is called engagement, and is characterized by a player becoming transfixed by a game. The player begins to put more importance on the game and starts to spend larger amounts of time playing or thinking about playing. From engagement, a player can move on to the next level called engrossment. At this point a player is spending a great deal of time playing the game, and considers the game very important. Finally, a player may reach the highest level called immersion. When a player reaches a state of immersion, the game experience becomes intrinsically rewarding, and the player may lose track of time or may even lose a sense of what's going on around him or her.

Right now there is no established method to measuring immersion. To help score some of the important factors of immersion, I decided to use a survey and scoring tool from the United Kingdom's Independent Television Commission [5]. This survey is called the Sense of Presence Inventory (SOPI), and was developed to help measure immersion and sense of presence through various media (television, radio, video games, etc…). The SOPI uses Likert scale questions with 1 to 5 scales that are scored into four main categories: spatial presence, engagement, ecological validity/naturalness, and negative effects. (More details in the methods section)

To my best knowledge, there has been no study on the direct effect of user immersion or the response from a user who 'gets immersed' in a game. Most studies that have an element of immersion aren't designed to study it alone [2, 6-8, 10]. Therefore, we are lacking evidence to describe the potential of focusing the design of games to promote it. Through the use of my user test and surveys, I'm able to describe the usefulness of immersion as a heuristic.

3. METHODS

As mentioned in the introduction, the primary method of experimentation was a user test with a few surveys. Two surveys were given prior to playing the game, the user test consisted of a single group game-play experience with a first-person shooter game - Half Life: Deathmatch, and one survey was given directly following game-play. The surveys were conducted in a group setting to allow for questions and explanations. A more detailed explanation follows.

3.1 Pre-Game

To gather participants, I collected a list of previous Saint Mary's University LAN party attendants and emailed them. A group of 17 participants was established. I met with them all as a group, and explained the experiment process in more detail. Noteworthy is the fact that a majority of the participants had experience playing the game, and had a moderate to extensive level of video game experience in general. The participants were told that I was conducting an experiment involving video games, and were offered both pizza and the possibility of cash prizes once the experiment was over. The top three places and two other random players would receive cash prizes.

The participants were also told to avoid caffeine, mood altering chemicals, and high physical exertion for at least 24 hours prior to the game date; participants were asked to limit their non-game

interactions with other players as much as possible when playing that day. Then I administered the first survey about game preference and experience (see Figure 1 for sample questions).

Due to the highly individualized nature of immersion, it was important to quantify some personal characteristics about the participants. These characteristics included genre preferences (role-playing games vs. sports or first-person shooters); individual worth of graphics, sound, storyline, etc…; and the participant's level of video gaming experience in general.

These factors are highly relevant to the person's willingness/ability to become immersed in a video game. Of course, some of these factors will inhibit some participants as well: for example, a person who doesn't normally like playing a first-person shooter and doesn't have experience doing so will have more of a barrier to immersion than an experienced player that regularly plays them.

On the experiment date participants were given a survey to determine their current physical condition and overall mood to ensure that results weren't skewed by players who were impaired by sickness, drugs, depression, etc… This second survey included mostly Likert scale questions (see Figure 2 for sample questions).

After all of the participants had completed the survey, I told them again to try to limit their non-game interactions with others, that the first 2 out of 5 rounds of 10 minutes would be practice before scores were kept, pizza would be served after the game was over, and that there were cash prizes for the top three players and two other random players.

Please order your preference of game genre 1 - 5 (5 being highest):

RPG's_____ First-Person Shooters_____ Strategy games_____

Sports games_____ Other (name)_____ _____

Overall, I generally play video games

< 1 hour per week _____ 1 to 4 hours per week _____

5-10 hours per week _____ > 10 hours per week _____

Figure 1. Sample questions from the first pre-game survey to determine gaming preferences and overall gaming experience

	Strongly Agree				**Strongly Disagree**
I am feeling good today	1	2	3	4	5
I want to play this game right now	1	2	3	4	5
Today has been a good day so far	1	2	3	4	5

Figure 2. Sample questions from the second pre-game survey for determining overall player condition

3.2 Conducting Play

For the experiment one computer lab was equipped with 20 Dell PCs that were wired into two switches. These two switches were wired into a final switch that was connected to the game server PC. The server was equipped with a Steam administrative package, and was set to change maps and reset player scores every ten minutes. By resetting the maps so often, I hoped to limit the advantage the experienced players would have in finding the best places on the map. Unfortunately, this setup does force breaks in a player's experience (even if only for 15-30 seconds) that weaken an immersive experience.

Before play started, the participants with no experience with the game were given basic instructions on how to move and use weapons. Once these players asserted that they understood the instructions, I asked everyone if there were any questions. No questions or objections were brought up, and all participants were instructed to begin game-play. Six separate maps were played for a total game time of one hour. Once the seventh map began to load, participants were asked to stop playing and take the final survey.

3.3 Post Game

The final survey consisted of mostly Likert scale questions dealing with the game experience and satisfaction and questions from the SOPI presence assessment tool. Figures 3 and 4 show sample questions from the post-game survey dealing with satisfaction and immersion respectfully.

Questions from the satisfaction survey determined how much fun the player had, and if the player had more fun, less fun, or a normal amount of fun during the experience as compared to usual experiences with similar games he or she has played (if any). The SOPI scored a series of questions related to experiencing some media (television, radio, video games, etc…) into four areas: spatial presence, engagement, ecological validity -naturalness, and negative effects. Each question has a scale from 1 (strongly disagree) to 5 (strongly agree). The mean score of each factor is generated and used to determine the level each factor was experienced.

Spatial presence is defined as the amount a player feels a part of the game or 'in the game' instead of sitting at a computer. Similar to the level of immersion defined in the introduction, engagement is the amount a player feels transfixed by the game and wants to continue playing. Naturalness measures how much a player finds the game environment and characters realistic. It has to do with graphical quality, game physics, and a range of other details that make the game believable. Negative effects are the adverse feelings the player experiences while playing (headache, eye-strain, dizziness, etc…).

Once the users completed the final survey, all of the surveys were collected, the participants were given pizza, and the top 3 and bottom 2 players were given cash prizes of $5, $2, $2, $5 and $5 respectively. Later, with help from Dr. Luttmers of the Saint Mary's University's Psychology department, all survey data was reviewed and input into SPSS software for ease of review and computing relations. SPSS is a powerful statistical program that allows for simple execution and processing of most normal functions (means, correlations, one way ANOVAs, etc…). Dr. Luttmers also assisted in quantifying this scoring information to make correlations between satisfaction and immersion.

	Strongly Agree				**Strongly Disagree**
There's something else I would rather have been doing	1	2	3	4	5
I enjoyed playing this game	1	2	3	4	5
I enjoyed the graphics	1	2	3	4	5
I had trouble using the keyboard	1	2	3	4	5

Figure 3. Sample questions from the post-game survey dealing with player satisfaction.

	Strongly Agree				**Strongly Disagree**
I felt I was a part of the game	1	2	3	4	5
I lost track of time	1	2	3	4	5
The displayed environment seemed real	1	2	3	4	5

Figure 4. Sample questions from the post-game survey copied from the SOPI to rate levels of engagement, presence, naturalness, and negative effects.

3.4 Limitations

There were several limiting factors involved in this study. First and foremost, this was a semester-long research project done for a class. This means that there were strong financial and time constraints. I was thus limited to using a gaming system and game that was already set up for use, and further limited by the number of participants I could accommodate.

The game itself was not the best choice for measuring immersion. Half-Life: Deathmatch is a FPS that puts all players in a free-for-all killing zone. Once killed, a player is immediately brought back to life in a new, random location. This feature leads to frantic, reaction-based play that weakens the ability to become immersed in the game.

4. RESULTS

As I explained before, there is no established method for determining a level of immersion. By using the SOPI I hoped to show a link between satisfaction and the three positive factors measured in the SOPI: presence, engagement, and naturalness. Unfortunately, I was only able to find a strong relationship between satisfaction and one factor-engagement. This does not mean that immersion is unrelated to satisfaction. The results found were

positive, and there is evidence that outside factors (small sample group, lack of resources, player bias, etc…) may have caused interference in the study.

Overall, the strongest relationships found were those dealing with engagement and others dealing with spatial presence. A strong, positive relationship was observed between engagement and player satisfaction whereby as the scores for engagement increased so did those for satisfaction. The SPSS software calculated an $r = 0.725$ where r ranges from 1.0 (directly positive relationship) to -1.0 (directly negative-inverse relationship); calculated significance (percent probability results were generated by chance) p was recorded as $p = 0.001$ where p ranges from 1.0 (100%) to 0.0 (0%).

Further, these two variables also associated with the time a user would wish to play the game at one sitting (given no other timely constraints): the greater the enjoyment or engagement the longer they would play. This was the expected outcome from the definitions. Also, negative effects show a negative relationship with these variables: the more the players enjoy the experience the fewer noticed negative effects.

Two strong relationships were found with spatial presence. In the first, as the amount the user felt challenged by the game increased, that user's spatial presence rating increased. Here the relationship was recorded as $r = 0.636$ and the significance as $p = 0.006$. In the second, the scores for user's spatial presence rating rose when the scores for the user's opinion about the quality of graphics rose. Also important to note is the breakdown of the levels of spatial presence. According to the data, the participants who reported playing video games an average of less than six hours per week (light to moderate players) experienced nearly significantly greater spatial presence than those who play six or more hours per week (heavy players).

Noticed points of importance related to the participant group follow. I found that heavy players reported significantly less negative effects than the moderate to light players. Light player's mean score was 2.576 compared to heavy player's mean of 1.472 on a 1-5 scale. A relationship between these players and realism also approached significance, whereby the moderate to light players found the game experience to be more realistic than the heavy players.

A relation with engagement was also found with the participant's game genre of choice. The participants who said that they favored first-person-shooter (FPS) games over all others reported significantly more engagement than those who favor some other genre over FPS. This outcome was also expected, and may have led to player bias towards the game.

Correlations

		Engagement	Enjoyed playing	Time would play at once	Negotive Effects
Engagement	Pearson Correlation	1	.725**	.710**	-.187
	Sig. (2-tailed)		.001	.001	.472
	N	17	17	17	17
Enjoyed playing	Pearson Correlation	.725**	1	.712**	-.416
	Sig. (2-tailed)	.001		.001	.097
	N	17	17	17	17
Time would play at once	Pearson Correlation	.710**	.712**	1	-.337
	Sig. (2-tailed)	.001	.001		.186
	N	17	17	17	17
Negotive Effects	Pearson Correlation	-.187	-.416	-.337	1
	Sig. (2-tailed)	.472	.097	.186	
	N	17	17	17	17

**. Correlation is significant at the 0.01 level (2-tailed).

Figure 5. SPSS table showing correlation data between each variable: level of engagement from the SOPI, amount player enjoyed experience, the amount a player would like to continuously play, and negative effects. Strong relationships noted between engagement, enjoyment, and time variables.

Correlations

		Spatial Presence	Ecological Validity/ Naturalness	Enjoy graphics	Was challenging
Spatial Presence	Pearson Correlation	1	.879**	.436	.636**
	Sig. (2-tailed)		.000	.081	.006
	N	17	17	17	17
Ecological Validity/Naturalness	Pearson Correlation	.879**	1	.359	.543*
	Sig. (2-tailed)	.000		.157	.024
	N	17	17	17	17
Enjoy graphics	Pearson Correlation	.436	.359	1	.599*
	Sig. (2-tailed)	.081	.157		.011
	N	17	17	17	17
Was challenging	Pearson Correlation	.636**	.543*	.599*	1
	Sig. (2-tailed)	.006	.024	.011	
	N	17	17	17	17

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

Figure 6. SPSS table showing correlation data between each variable: presence, naturalness, graphics enjoyment, and challenge. Fairly strong relationships noted between all variables.

5. CONCLUSIONS

While I am not able to say directly that immersion leads to player satisfaction, I have seen evidence that some factors of immersion (especially engagement) were strongly related to satisfaction for this game. This particular FPS generated a large amount of engagement and a related amount of enjoyment. There were some setbacks that negatively affected results, but I believe it can still be hypothesized that immersion will make a strong video game design heuristic for player satisfaction.

Users purchase video games and systems, spend their own time and efforts to learn to use them, and continue to purchase such items in the future with little to no interaction with the people who produce them. In order to fully capitalize on this occurrence, video game producers need to design games that users feel satisfied in purchasing. Satisfaction leads to replay, reputation building, and future purchase; ultimately, satisfaction leads to profit for the video game industry. And when you're talking about games in which users are already willing to pay real money to other users for money or items only usable in a game world, better game design means a lot more revenue for game producers. The results of my study will potentially help the game developers increase the amount of time a user wishes to play, the user's overall enjoyment in the product, and the likelihood that the users will purchase similar products from them in the future.

Acknowledgment. Dr. Larry Luttmers provided essential reviews and insights throughout this project and this author would like to thank him for all of his assistance.

REFERENCES

[1]. Atari Inc. Third Quarter Revenue (October to December 2005). Feb 9. 2006. <http://corporate.infogrames.com/uk/download/pr/corporate/atari_uk_653_CA3TR05.06FINALUK.doc>.

[2]. Cheng, Kevin; Cairns, Paul A. Behaviour, Realism and Immersion in Games. *ACM Press*. April 2005.

[3]. Emily Brown, Paul Cairns. A Grounded Investigation of Game Immersion. *ACM Press*. April 2004.

[4]. Federoff, Melissa A. Heuristics and Usability Guidelines for the Creation and Evaluation of Fun in Video Games. *ACM Press*. December 2002.

[5]. Jane Lessiter, Jonathan Freeman, Edmund Keogh, Jules Davidoff. A Cross-Media Presence Questionnaire: The ITC-Sense of Presence Inventory. *Presence, Vol. 10, No. 3*, 282–297. June 2001.

[6]. Jeanne B. Funk, Ph. D.; Pasold, Tracie; Baumgardner, Jennifer. How Children Experience Playing Video Games. *Carnegie Mellon University*. May 2003.

[7]. Pekala, R. J., Wenger, C. F., & Levine, R. L. (1985). Individual differences in phenomenological experience: States of

consciousness as a function of absorption. *Journal of Personality and Social Psychology, 48,* 125-132

[8]. Ravaja, Niklas; Salminen, Mikko; Holopainen, Jussi; Saari, Timo; Laarni, Jari; Järvinen, Aki. Emotional Response Patterns and Sense of Presence during Video Games: Potential Criterion Variables for Game Design. *ACM Press*. October 2004.

[9]. Sweetser, Penelope; Wyeth, Peta. GameFlow: A Model for Evaluating Player Enjoyment in Games. *ACM Press*. July 2005.

[10]. Tavinor, Grant. Video Games, Fiction, and Emotion. *Creativity & Cognition Studios Press*. November 2005.

[11]. Thomas Schubert, Frank Friedmann, Holger Regenbrecht. Decomposing the Sense of Presence: Factor Analytic Insights. *Second International Workshop on Presence*. January 1999.