2018 WL 3822751 (E.D.Pa.) (Expert Report and Affidavit)
United States District Court, E.D. Pennsylvania.

PGH GLOBAL (CAYMAN) LIMITED, Plaintiff,
v.
LIQUID INTERACTIVE, LLC, Defendant.

No. 5:17-CV-00454.
February 2, 2018.

Expert Report of Philip Greenspun, Ph.D.

Name of Expert: Philip Greenspun, Ph.D.
Area of Expertise: Computers & Electronics >> Software
Area of Expertise: Computers & Electronics >> Electronics, Devices, Circuitry & Systems
Area of Expertise: Computers & Electronics >> Computer Science
Area of Expertise: Engineering & Science >> Engineers/Engineering
Representing: Defendant
Jurisdiction: E.D.Pa.

TABLE OF CONTENTS

I. Introduction ... 1

II. Qualifications ... 1

III. Materials Considered ... 7

IV. Background: What is Software Development? ... 7

A. Database Management Systems ... 7

B. Database applications ... 9

C. Client/Server Database Applications ... 9

D. Web Applications ... 12

1. Static HTML Pages ... 13

2. Computer programs that generate HTML pages ... 16

E. Web-based Database Application ... 17

F. Software Toolkits ... 21

V. Background: What is Software Maintenance ... 21

VI. Background: What are Software Development Methodologies and How Many are there? ... 22

VII. Background: What is Version Control ... 26

VIII. Background: What is Issue Tracking ... 27

IX. Background: What do Business Managers Know about Over-Optimism and Project Risks ... 29

X. Background: What do Business Managers Know about Software Development Risks ... 30

XI. Background: How much CAN Managers Know? ... 32

XII. Background: Software Security Vulnerabilities ... 33

XIII. Background: Gathering Requirements ... 35

XIV. What was Liquid Hired by PGH to do? ... 36

XV. What is the v1.0 system that Liquid was given to maintain? ... 40

XVI. What did Liquid do in its work on the v2.0 system? ... 47

XVII. Did the v1.0 system work? ... 51

XVIII. What is the v2.0 system that Liquid Produced? ... 52

XIX. How many hours did Liquid bill to PGH for work on v2.0? ... 58

XX. Did the v2.0 system work? ... 58

XXI. The Golbeck Report ... 69

XXII. The Golbeck Report on Software Development Methodologies ... 72

XXIII. What specifications, requirements documents, block diagrams, and other planning documents were written for v1.0 maintenance? ... 74

XXIV. What specifications, requirements documents, block diagrams, and other planning documents were written for v2.0 development? ... 77

XXV. What was the quality of work done by Liquid? (The Data Import Project) ... 81

XXVI. What was the quality of work done by Liquid? (v2.0) ... 84

XXVII. Was PGH in any way responsible for the v2.0 disappointment? ... 87

XXVIII. Opinions ... 90

XXIX. Reservation Of Rights ... 92

I. INTRODUCTION

1. This report summarizes my opinions in the above-captioned case.

2. I have been retained by counsel for Liquid Interactive, LLC (“Liquid”) in connection with this matter to investigate and offer opinions regarding the types of agreements common in the software development industry; the allocation of risk associated with each; and whether Liquid's fulfilled the specific agreements it entered into during its relationship with PGH based on their allocation of risk.

II. QUALIFICATIONS

3. I received a Bachelor of Science Degree in Mathematics from the Massachusetts Institute of Technology (“MIT”) in 1982 and a Master of Science Degree in Electrical Engineering and Computer Science from MIT in 1993. As part of my education at MIT, I completed the coursework for a Bachelor's in Electrical Engineering, including digital logic systems. I subsequently served as a teaching assistant for the core electrical engineering and computer science classes at MIT.

4. In 1999, I received a Ph.D. in Electrical Engineering and Computer Science from MIT. My thesis concerned the engineering of large online Internet communities with a Web browser front-end and an RDBMS containing site content and user data. The overall architecture of the systems described in my Ph.D. thesis is similar to that of the software that is at issue in this case.

5. I have worked with relational database management systems (“RDBMS”) and Web-based database applications for over twenty years. Over the course of my career, many of the software and hardware engineering projects in which I have been involved presented issues relating to security, access control and the design of trusted systems. My resume is attached hereto as Exhibit 3.

6. I began working full-time as a computer programmer in 1978, developing a database management system for the Pioneer Venus Orbiter at the National Aeronautics and Space Administration's Goddard Space Flight Center.

7. From 1982 to 1983, I assisted with the construction of a prototype minicomputer implementing the Hewlett-Packard Precision Architecture, subsequently incorporated into Intel's Itanium 64-bit server processors, which remain available as of 2015.

8. From 1983 to 1984, I worked on the hardware and operating system software for the Symbolics Lisp Machine workstation, a $100,000 personal computer. While at Symbolics I developed the most popular game written for that platform, a multi-player coordinated-across-the-network version of the Spacewar game (developed in 1962 for the PDP-1 computer, but requiring the two players to sit at the same terminal).

9. In 1984 and 1985, I developed computer-aided engineering software system called “ICAD”. Running this program became the most common reason for commercial purchases of the Symbolics computer.

10. From 1986 to 1989, I co-developed a hardware/software system for determining the precise location of earthmoving equipment and providing instructions to the operators of bulldozers and other earthmovers.

11. At various points during the 1980s I entered into contracts with companies, including Fidelity Investments and Textron, that wanted software developed. Some of these were hourly contracts, like the agreements at issue in the present case, and some were firm fixed-price contracts.

12. I launched my personal Web site, which also included what is now the photo.net online community, in 1993. Prior to the end of 1993, my personal site included CGI scripts processing user-submitted form data. Shortly thereafter I began consulting work for commercial enterprises that were launching interactive web sites with dynamic web pages.

13. In October 1994, I developed a Web-based interface to an electronic medical record system. This was a joint project between MIT and the Boston Children's Hospital. The system is described in "Building National Electronic Medical Record Systems via the World Wide Web" (Kohane, Greenspun, et al. 1996; *Journal of the American Medical Informatics Association* 3:3). This system was built using the Perl language, then the most common CGI scripting language, using the Oraperl system that enabled Perl scripts to send SQL queries and commands to an Oracle relational database management system.

14. In 1995, I led an effort by Hearst Corporation to transform their New Media organization from CD-ROM publishing to Internet publishing and commerce. My first project was to set up an infrastructure for Internet Applications across all of its newspaper, magazine, radio, and television properties. This infrastructure included software for managing users, shopping carts, electronic commerce, advertising, and user tracking. We launched our first ecommerce site, in which consumers could build up a shopping cart of magazine subscriptions and check out and pay with a credit card, during the July 4 weekend of 1995 (shortly before the public launch of Amazon.com). This site was hosted on a Sun SPARCstation desktop computer running the Solaris version of Unix. All data regarding site content, user registration, and orders was held in a relational database management system (Illustra). My work for Hearst was done as an independent contractor and I also was involved in negotiating and managing other software development contractors.

15. Between 1995 and 1997, I significantly expanded the photo.net online community (which I had started in 1993) in order to help people teach each other to become better photographers. I began distributing the source code behind photo.net to other programmers as a free open-source toolkit, called "ArsDigita Community System." Photo.net enabled users to ask questions, answer others' questions, upload photos for discussion, etc.

16. In May 1997, Macmillan published my first textbook on Internet Application development, "Database Backed Web Sites." This started as a hardcopy collection of articles that I had been publishing on my personal web site starting in 1995 as tips for other web developers on a page titled "Web Tools Review."

17. In 1997, I started a company, ArsDigita, to provide support and service for the free open-source toolkit. Between 1997 and the middle of 2000, I managed the growth of ArsDigita to 80 people, almost all programmers, and $20 million per year in annual revenue. This involved supervising dozens of software development projects, nearly all of which were Internet Applications with a Web front-end and an Oracle RDBMS back-end. My work as CEO also involved negotiating contracts with customers, reviewing those contracts, and dealing with any disputes that arose regarding ArsDigita's work as a software development vendor.

18. As part of my work at ArsDigita, I wrote an article titled "Professionalism for Software Engineers" (2000) [1] . The article acknowledges that, depending on the circumstances in which a programmer is working, *professionalism* per se may be an impossible goal. The article lists some aspirational goals for "software engineering professionalism" and characterizes this as a "new definition of professionalism."

19. Between 2000 and 2017, I have done software development projects for philip.greenspun.com and photo.net, two online services that are implemented as relational database management applications. In addition, I have developed postclipper.com, a Facebook application that allows parents to create electronic baby books.

20. Separately from this commercial and public work, I have been involved, as a part-time teacher within the Department of Electrical Engineering and Computer Science, educating students at MIT in how to develop Internet Applications with an RDBMS back-end. In the spring of 1999, I taught 6.916, Software Engineering of Innovative Web Services, with Professors Hal Abelson and Michael Dertouzos. In the spring of 2002, this course was adopted into the standard MIT curriculum as 6.171.

I wrote 15 chapters of a new textbook for this class, *Software Engineering for Internet Applications.* This book was published on the Web at http://philip.greenspun.com/seia/ starting in 2002 and 2003 and also in hardcopy from MIT Press in 2006. I am the sole author of a supplementary textbook for the class, "SQL for Web Nerds," a succinct SQL programming language tutorial available only on the Web at http://philip.greenspun.com/sql/. From time to time I teach an intensive database programming course at MIT. The most recent version of this class was offered by the Department of Electrical Engineering and Computer Science in January 2015 as 6.S186.

21. In addition to my teaching in the EECS department at MIT, in January 2018 I taught 16.687 for the MIT Department of Aeronautics and Astronautics and joined the teaching staff for AISC610, Computationally-Enabled Medicine, at Harvard Medical School.

22. I have authored five computer science textbooks in total, including *Database Backed Web Sites* (Macmillan), *Software Engineering for Internet Applications,* and a SQL language tutorial.

23. I am a named inventor on three issued U.S. patents.

24. I have served as an expert witness for Amazon.com, Ford Motor Company, Google, IBM, and Philips in patent cases. I have served as an expert witness in software-related cases for Microsoft and Morningstar.

25. Within the past four years, I have testified via deposition or trial in the following cases:

• *Volusion v. Versata,* cases CBM2013-00017 and -0018 before the Patent Trial and Appeal Board (deposition);

• *Versata v. Zoho,* (1:13-CV-00371-SS WD TX) (claim construction hearing);

• *Wonderbox Technologies, LLC v. The Management Group, Inc. andIron Data Solutions, Inc.* (Case No. 14-CV-484; State of Wisconsin Circuit Court, Ozaukee County) (hearing);

• *Hoskin Hogan v. BP West Coast Products LLC and Retalix* (Case No. BC 460880, California Superior Court, County of Los Angeles);

• *ContentGuard Holdings, Inc. v. Amazon.com et al* (2:13-cv-01112-JRG, ED TX);

• *Parallel Networks Licensing, LLC, v. International Business Machines Corporation,* Case No. 13-2072 (SLR) (District of Delaware);

• *Old Republic General Insurance Group v. Intellectual Ventures.* IPR2015-01706, IPR2015-01707, IPR2016-00453;

• *Intellectual Ventures v. U.S. Cellular,* (Delaware) 13-cv-1636-LPS and -1637-LPS;

• *Ford Motor Company v. Versata Software, Trilogy Software, et al.* (Case Nos. 15-10628-MFL-EAS and 15-11624-MFL-EAS, ED Michigan)

26. I am a salaried employee of Fifth Chance Media LLC, which I understand is being compensated for my involvement in this case at the rate of $475.00 per hour. My compensation does not depend in any way on the outcome of this litigation and I am not an owner or part-owner of Fifth Chance Media LLC.

III. MATERIALS CONSIDERED

27. In forming my opinions, I have relied on my over thirty years of experience with computer science, and hardware and software engineering, including my over twenty years of developing and managing database-backed web sites. The materials that I have relied upon are cited throughout this report. I have also reviewed additional documents produced by both parties in this matter and all of the deposition transcripts to date.

IV. BACKGROUND: WHAT IS SOFTWARE DEVELOPMENT?

28. Software development, or "computer programming," is writing software in a human-readable language that can be compiled or interpreted and then executed by a processor so as to achieve a desired function. In the 1950s, when digital computers were new, a computer programmer would typically write a program that stood by itself, without substantial reliance on existing subsystems or building blocks. By the 1960s, however, it had become conventional for programmers of new systems to rely on building blocks.

29. This section describes some of the building blocks that business data processing and web sites typically rely on and that were in fact used in the software at issue in this case.

A. Database Management Systems

30. A database is an organized collection of information. A computer program that assists programmers with common challenges regarding creating, updating, and querying a database is a database management system (DBMS). The most popular type of DBMS is the relational DBMS or "RDBMS", the best conceptual model for which is "a big spreadsheet that several people can update simultaneously." Commercial DBMSes were widely available in the 1960s, notably the Information Management System (IMS) developed by IBM for the IBM 360 mainframe computers. The RDBMS was conceived by E.F. Codd, a researcher at IBM, in the late 1960s, but did not become available commercially until the late 1970s. IBM, Microsoft, and Oracle were the market leaders in commercial RDBMS in the 1990s and remain so today. MySQL is a free and "open-source" (the source code can be downloaded and modified, if desired) alternative that was released in 1995 and is the leader within this segment today. MySQL is the RDBMS that was used in the software that is at issue in this case.

31. Information within an RDBMS is stored in tables. Communication with the RDBMS is via the Structured Query Language (SQL), which includes statements such as CREATE TABLE, INSERT (add a row), UPDATE (change a data item within a table), DELETE (remove a row), and SELECT (return a report). The "SQL schema" or "data model" is a collection of CREATE TABLE statements that prepares the RDBMS to accept data items. Note that the SQL language was developed in the early 1970s by IBM employees Donald Chamberlin (retired from IBM in 2009) and Raymond Boyce (died of an aneurysm in 1974). SQL was adopted by competitive RDBMS vendors such as Oracle and became an American National Standards Institute (ANSI) standard in 1986.

32. For example, suppose that a Web site developer wished to record subscribers to an email newsletter. He or she would create a new table called mailing list with two columns (also sometimes called "fields"), email and name, both text strings that can be up to 100 characters in length ("varchar(100)"):

```
create table mailing list (

email varchar(100),

name varchar(100)

);
```


33. After a series of INSERT commands, the contents of this table may be viewed in a spreadsheet format, e.g.,

name ... email

Philip Greenspun ... philg@mit.edu

Bill Gates ... billg@microsoft.com

Scott Adams ... scottadams@aol.com

by typing a SELECT statement such as

select name, email from mailing_list

34. Each row in the table is also referred to as a record. Note that, unlike with a desktop spreadsheet application, every data item in the same column, e.g., email, must be of the same data type (in this case a character string).

B. Database applications

35. A database application is a computer program whose primary purpose is entering information into and retrieving information from a computer-managed database. Some of the earliest database applications were accounting systems and airline reservation systems such as SABRE, developed between 1957 and 1960 by IBM and American Airlines.

36. Users interacted with early database applications, such as SABRE, by typing at a terminal connected directly to a mainframe computer running the database management system. The IBM 3270 terminal, introduced in 1972, was a commonly used device. The terminal had no ability to process information, but merely displayed characters or “screens” sent from the mainframe.

37. Software development for an early database application was simply software development for the mainframe computer, which executed all of the program code centrally. Software would typically be developed in one of the “third-generation languages” developed in the late 1950s and early 1960s, e.g., COBOL, Fortran, or PL/I. Whatever conventional programming language was used, there may have been embedded database commands in a specialized query language.

C. Client/Server Database Applications

38. A client/server database application is a system in which the programs to organize data and interact with users have been split up among a central server and distributed personal computers. With the development of the inexpensive personal computer in the 1980s, it did not make sense to have a central mainframe do all of the work. Programmers realized that substantial performance and interactivity improvements could be achieved by distributing some of the logic, especially user interface software, to machines on users' desktops. The central computer (server) was retained, but it ran only the database management system (DBMS). The personal computers on users' desktops (clients) ran the software that displayed menus and reports, drew graphs and charts, and sent queries and transactions to the DBMS. Instead of screens and keystrokes being communicated from the central computer to the desktop, the desktop PC would send queries to the DBMS and receive results back in exchange. As the rise of the desktop PC coincided with the circa-1980 rise of the relational database management system (RDBMS), the most common language flowing between clients and servers was SQL.

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

D. Web Applications

39. The World Wide Web was a client/server system developed by Tim Berners-Lee at CERN in Geneva and was up and running late in 1990. In a 1992 paper by Berners-Lee, "World-Wide Web: The Information Universe," Berners-Lee places the Web in the context of earlier hypertext and client/server information retrieval systems: "Hypertext and text retrieval systems have been available for many years, and a valid question is why a global system has not already come into existence." He notes that "The W3 architecture must cope with a widely distributed heterogeneous set of computers running different applications that use different preferred data formats. This requires a client-server model." Berners-Lee notes that "writing a server for new data is generally a simple task because it requires no human interface programming." In other words, the programmer can build an application that can be used from Macintosh without writing user interface software (such as to produce menus) for the Macintosh operating system.

40. A "Web application" is a server-based computer program that can be used by a user sitting in front of a standard Web browser, such as Microsoft Internet Explorer, Firefox, or Google Chrome. Examples of Web applications include wikipedia.org, nytimes.com, facebook.com, and youtube.com. The advantage of the Web, compared to earlier client/server systems, is that no specialized software need be installed on the end-user's computer. The same personal computer or mobile phone and the same Web browser can be used to read a news article from nytimes.com, order a book from amazon.com, or calculate the cost of a new car using Google Spreadsheets. A classical Web application is very similar to an old mainframe application. The substantive computation is performed on the central computer in response to a request for a URL (e.g., "http://www.nytimes.com/pages/business/index.html requests the file "index.html" located in the "/pages/business/" directory on the nytimes.com server) and only information necessary to paint a "screen" is sent to the device on the user's desktop via the Hypertext Transfer Protocol (HTTP). Thanks to 30 years of progress in microelectronics, the desktop device can display color graphics rather than simply green characters, but the software ideas are similar to those used in the mainframe era.

41. Rather than the IBM 3270 terminal protocol, the specifications for the screen or "page" to be displayed are sent in Hypertext Markup Language (HTML), which is a specification or format for a document for display in a browser (analogous to the format in which Microsoft Word, for example, might save a file to disk). The structured data in HTML is distinct from a programming language, which generally either expresses a step-by-step algorithm for a computer to follow or specifies a computation to be performed.

1. Static HTML Pages

42. Some early Web sites, e.g., online manuals, were simply static collections of HTML documents and digital images. The server did no computation other than processing the request to see which file was being requested and then delivering the contents of the file via HTTP. These were informally referred to as "static sites". Every user who requested a file with a particular name would be served exactly the same contents, regardless of the identity of the user, the time of day, or what files had been previously requested by that user. Here is an example of a one-sentence .html page:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

43. The source HTML behind this page is a file whose entire contents are the character string "Texas is really big." In other words, a plain-text string of characters can be rendered by a Web browser as an HTML document. Every word appears in the default font and face. Suppose the author wishes to emphasize the word "really":

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

44. This is accomplished by surrounding the word with an "emphasize" tag: "Texas is really big." The Google Chrome browser, used to create the above screen shot, after reading the EM tag, elected to display the word "really" in italics. That's the *markup* part of HTML. A typical HTML document looks more like the following:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
Here's the HTML behind the preceding page:

<html>

<head>

<title>Facts about Texas</title>

</head>

<body bgcolor=white text=black>

<h2>Texas Facts</h2>

<hr>

Texas became a state in 1845.

<h3>Geography</h3>

Texas has an area of 268,820 square miles.

<p>

... a new paragraph ...

<h3>More</h3>

Wikipedia article on Texas

</body>

</html>

45. After the BODY, which represents the start of the document to be displayed, "Texas Facts" is marked up as an H2 headline. The HTML then specifies a horizontal rule (HR) across the page, includes a sentence about Texas's statehood, and then marks up the word "Geography" as a smaller H3 headline. At the bottom is a link to Wikipedia using the A tag. This is the *hypertext* part of HTML. As noted above, HTML is structured data representing a document for display in a browser. It is not a programming language and lacks the most basic capabilities for programming, e.g., it is not possible to instruct a computer to add two numbers via HTML. There is no "ADD" tag.

2. Computer programs that generate HTML pages

46. It was possible from the earliest days of the web, e.g., 1990-1992, to write computer programs that would run in response to user requests and generate HTML pages on the fly. The generated HTML pages output by these programs were referred to as "dynamic pages." Depending on the URL requested by the browser, the web server program would determine whether to deliver

the contents of a file from the file system or to run a program instead. A problem faced by early web developers was that page-generating programs written as extensions to the original CERN HTTP server, for example, would need to be modified in order to function as extensions to the NCSA HTTP server program, then becoming more popular. In 1993 NCSA promulgated the Common Gateway Interface (“CGI”) standard. Page-generating programs written in the CGI standard could function without modification even when the front-end web server or “HTTPD” (a generic acronym for “HTTP daemon”) was switched to a different program. CGI programs could be written in any programming language and the only restriction on their behavior was that their eventual output conformed to a standard Web server response. These programs are typically short and simple and referred to as “page scripts”. Here's an example of a page generated by a computer program (in the Perl programming language):

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
Here's the Perl program that generated the preceding page:

#!/usr/bin/perl

the (above) first line says where to find the interpreter

for the rest of the program

any time we print, the output goes to the user's browser print “Content-type: text/plain/n/n”;

#now we have printed a header (plus two newlines) indicating that the

document will be plain test; whatever else we write to standard

output will show up on the user's screen

Get the current time as a string.

$current time = localtime();

Run a program to get the system status.

$systemstatus = ‘/usr/bin/uptime’; print “As of $current time, system status is:/n/n$system status/n”;

47. Note that the output displayed by the browser is a combination of static text, e.g., “As of” and “system status is:”, and information obtained or calculated by the running Perl program. All of this is happening on the server; all that the browser knows is that it received a stream of plain text characters (based on the “text/plain” Content-type sent by the first line of code in the Perl program).

E. Web-based Database Application

48. A “Web-based Database Application” is simply a combination of the systems described above, i.e., “a computer program that can be used by a user sitting in front of a standard Web browser whose primary purpose is entering information into and retrieving information from a computer-managed database”. No additional software needs to be installed by end-users. The computer programs on the server, however, are modified so that they rely on a database management system (DBMS) for storing and retrieval of information. The user requests pages by URL, as before, and in response the server will run computer programs (page scripts) that will generally access the DBMS and then merge the results with a template in order to build up a complete HTML page to return.

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

49. The structure of one of the page scripts shown in Figure 2 is best explained by example. The flow chart below uses an example of a bank customer logging in to view the most recent transactions to his or her account. After typing in a username and password, the customer clicks on a "view transactions" link. This causes the browser to request the link "/transactions" from the Web server, which will run a page script (short computer program) in response. The page script will connect to the RDBMS, query the transactions table, and then merge the data items received back from the database with some fixed HTML (a "template") in order to build a complete HTML page to return to the customer. The final HTML page as viewed by the customer will contain some boilerplate content from the HTML template, but also a list of recent transactions that have been queried from the RDBMS.

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

F. Software Toolkits

50. Whenever multiple customers have had similar needs, software toolkits have sprung up to make the task of building information systems easier. For example, an early use of mainframe computers was the processing of accounting data. At least within a given country, all customers would be subject to the same accounting rules. Thus it became conventional to build accounting systems for new customers starting from software previously built. These bodies of software grew into packaged toolkits such as SAP and Oracle Applications.

51. The first toolkits for Web applications were released in the mid-1990s, including the ArsDigita Community System that I worked on. Wordpress, whose use in the systems at issue in this case is described below, is currently the most popular toolkit for building web sites. It was first released in 2003 and has been adopted by tens of millions of web sites worldwide. Data from automated crawlers suggest that roughly 29 percent of the world's web sites use Wordpress. [2]

52. The process of software development starting from a toolkit consists of determining if there are any gaps between what the toolkit can do out of the box and what is required for a web application. Those gaps are filled by a programmer writing additional software. The conventional way of extending the capabilities of Wordpress is to write a "Plugin". [3]

V. BACKGROUND: WHAT IS SOFTWARE MAINTENANCE

53. Software needs to be maintained continuously. Sometimes this is because a programming error ("bug") has been discovered and a "bug fix" is required. In general it is not straightforward to test all possible situations in which software might be used and therefore bugs can remain hidden in code for years or decades. A famous example of this was the "Y2K" bug in which enterprises did not know which programs would break because they were using only two decimal digits to represent a year, e.g., "67" for "1967", and the software's behavior after January 1, 2000 was unknown.

54. Another reason for making a change to a computer program is that there has been a change to underlying systems on which the program relies. For example, the operating system or RDBMS might change what it expects from an application program and therefore it becomes necessary to update the application to match the latest version of the RDBMS or operating system.

55. If the software has been developed using a toolkit and a security flaw is discovered in the toolkit it will be necessary to upgrade the version of the toolkit being used and then to make sure that any customizations of that toolkit remain functional given any changes that have been made to the toolkit.

56. A software maintenance task might take few just a minutes to accomplish, e.g., following receipt of a bug report from a user. Alternatively it could be a multi-year project, as was undertaken in preparation for Y2K.

VI. BACKGROUND: WHAT ARE SOFTWARE DEVELOPMENT METHODOLOGIES AND HOW MANY ARE THERE?

57. Unless the programmer is writing software for his or her own use, all software development includes finding out what the proposed software must do and then writing code in an attempt to make that happen. The degree of formality with which this process unfolds is highly variable and there is no agreement within the industry regarding any specific process that is best. "A Dynamic Framework for Classifying Information Systems Development Methodologies and Approaches" (*Iivari, et al.* 2001; *Journal of Management Information Systems* 17:3; Attached as Exhibit 1) notes that there are "over 1000" recognized methodologies and cites work referring to this as a "methodology jungle," which the authors characterize as "a seemingly impenetrable maze of competing ideas and notions" (page 180). The authors note that "recently, there have been a number of ambitious calls to establish systems development and software engineering as a profession [29, 32]. If one adopts strict criteria for what constitutes a profession, it is unlikely that ISD could be considered a profession in the foreseeable future [85]." Note that "ISD" stands for "Information Systems Development," a fancy term for "computer programming."

58. One reason why there are 1000 competing methodologies and nobody can agree on which one is "best" or should be adopted as an industry standard is that none actually is "best" and can be considered appropriate for all projects. Choosing an appropriate method for software development involves tradeoffs. Time spent on process, for example, is time that cannot be spent writing code. Carefully writing down requirements before writing any code incurs the risk that, once the customer or users see the running program, they'll say "actually that's not what we want even if we did previously say that is what we wanted."

59. One niche where software development processes are more constrained than in the rest of the industry is software for aviation. The FAA lays down rules in 14 CFR 23.1309, explains those rules in a 56-page "advisory circular," AC 23.1309-1E, which further refers to industry group standards from the RTCA, more than 1000 pages including DO-178 and DO-254. These rules govern, for example, the software that drives the displays incorporated into an aircraft panel ("dashboard"). These displays, and the avionics behind them, tell the pilots whether they are upside down, where they are going, what altitude and airspeed they are at, etc. Perhaps as a result of this process, the software behind what is displayed to pilots tends to be more reliable than ordinary desktop, phone, or web applications. However, even the FAA recognizes that these procedures will slow down innovation and drive up cost. Thus, the standards are different depending on the size of the aircraft, with a more elaborate process being applied to software within an airliner than in a business jet or four-seat airplane.

60. Why wouldn't everyone building software adopt the FAA-mandated standards for aviation software? This level of process delays development to such a large extent that there are now iPad aviation apps that are more powerful than the software driving the panel displays in a Boeing 737 or Airbus A320. A Silicon Valley company that used these methods would arrive in the marketplace years later than competitors having spent 10 or 100 times as much money. Small companies might be shut out of markets altogether.

61. Note that using a "heavy" FAA-style process does not guarantee that a project will be successful. In fact, an actual FAA-run project is often used as an example of failure in software development. From "Advanced Automation System Problems Need to be Addressed," (March 10, 1993 Testimony Before the Subcommittee on Aviation, Committee on Public Works and Transportation, House of Representatives) [4] :

In 1983, the total cost estimate for AAS was project to be $2.5 billion and completion was scheduled for 1996. When the contract with IBM was signed in 1988, FAA estimated the project would cost $4.8 billion and be completed in 1998. Since that time, the projected costs have increased to $5.1 billion, and the estimated completion date has slipped to 2002."

62. Five years later, the project remained in trouble. “Evolution and Status of FAA's Automation Program” (March 5, 1998 Testimony Before the Subcommittee on Aviation, Committee on Transportation and Infrastructure, House of Representatives) [5] states that, despite substantially relaxed requirements, the deadlines had continued to slip, e.g., from 2002 to 2004 (see page 14).

63. The textbook *Software Engineering, A Practitioner's Approach, Eighth Edition* (McGraw-Hill 2015) states that “these [process] models have brought a certain amount of useful structure to software engineering work ... software engineering work and the products that are produced remain on ‘the edge of chaos.’ ” (page 40) In other words, there is no methodology that the community of software developers agrees will guarantee improved results in every, or even most, programming situations.

64. Another way to understand the limits of academic approaches to software development methodologies is to recognize that it is primarily a set of conjectures based on unsuccessful projects. In this it is not different from Monday Morning Quarterbacking, in which we observe that people who are great at pointing out errors made during last Sunday's football game do not in fact possess valuable knowledge about how to win next Sunday's football game. One 2012 estimate of losses due to failed IT projects was at least $100 billion annually for companies in the S&P 500. [6] If there were surefire software development methodologies that could ensure that a company's IT projects would never underperform, go over budget, suffer a schedule slippage, or fail entirely, why wouldn't people who are expert in those methodologies be able to take up highly paid executive positions in any corporation that spends money on IT?

65. (In fact, demand for experts in software engineering methodology seems to be weak. Here's part of the dedication page of *Software Engineering: A Practitioner's Approach,* Eighth Edition:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

VII. BACKGROUND: WHAT IS VERSION CONTROL

66. Starting in the 1990s it became common for teams of programmers to use version control systems, also known as “revision control” or “source control.” Relying on inexpensive disk storage, the version control system keeps a record of every file of source code (software authored by a programmer) that is a component of a larger system. For every file, the version control system keeps a record of every version that has ever been “checked in,” the date on which the check-in occurred, and the username of the programmer who made the check-in. By inspecting the version control system's “repository,” it is possible to see when work was done, what work was done, and by whom work was done.

67. The use of a version control system may be considered a best practice and generally the use of a version control system is characteristic of a more organized effort to produce lasting software. This is not to say that all programming requires the use of version control. A programming making a throw-away experiment might not use a version control system. A student in a programming course probably would not use a version control system.

68. Version control systems are helpful when testing software because they can label a snapshot of all of the files as they exist at a particular moment, e.g., as “version 2.356” or “january-beta-test”. Development can then proceed in parallel with testing and bugs reported by the testers later referenced to specific versions of specific files.

69. Popular systems in the 1990s included CVS (free and open source; released in 1986) and Microsoft Visual SourceSafe. More popular systems today include Subversion (2000) and Git (2005). The contractor that built the “v1.0” system at issue in this case, FocusMX, seems not to have used any version control system. As discussed below, Liquid used the Git system for both its maintenance work on v1.0 and its development work on v2.0.

VIII. BACKGROUND: WHAT IS ISSUE TRACKING

70. A version control system tracks *what* was done with source code. A bug tracking or issue tracking system records *why* changes were made.

71. A bug tracking system typically allows a tester, a programmer, a customer, or even an end-user to enter an issue into a database. A programmer can then be assigned to investigate the issue or bug. The programmer can request additional information or a clarification from the person who originally entered the issue. The programmer, once a change has been made to the source code and checked into the version control system, can close the issue.

72. The same process and the same system can be used, and are typically used, for handling both defect reports (actual "bugs") and feature requests (for enhancements to software). The author of the DoneDone system, a popular bug tracker, points this out: "To us software engineers, the difference between a bug and a feature request is crystal clear. A bug is a discrepancy between how code is actually working and how our code was intended to work. A feature request requires *new* code to satisfy a case that can't be handled by the current codebase.... But, when it comes to just the pure practice of building good software, particularly if we're working on our own products, how much should we care about categorizing an issue this way?" [7]

73. Good software can be built and high quality maintenance can be performed without a bug tracking system. Bug reports and feature requests could be received by telephone, recorded on paper, and progress tracked via a file cabinet and folders. A list of outstanding tasks could be kept in a spreadsheet or word processing document and updated following the receipt of email. Nonetheless, in the age of ubiquitous web browsers and Internet access, the use of a bug tracking system is characteristic of an organized and efficient process.

74. Computerized "trouble ticket" tracking was being used by AT&T in the 1970s. The idea became practical for everyday programming projects in the 1990s as the cost of running a database management system fell. Popular bug tracking systems include Bugzilla (free and open-source, released in 1998), FogBugz (2000), and the DoneDone hosted service (2009). As described below, Liquid and PGH collaborated using the DoneDone service. (Note that I personally supervised the development of a ticket-tracking module within the ArsDigita Community System circa 2000. The customer for this module was a division of Nokia. The documentation for the module states "software occasionally has bugs. The ticket system is designed to record and track tasks, bugs, and feature requests. The intent is to have clearly defined responsibilities for tickets in every state and to have auto-notification capabilities which ensure that no ticket slips through the cracks." [8])

IX. BACKGROUND: WHAT DO BUSINESS MANAGERS KNOW ABOUT OVER-OPTIMISM AND PROJECT RISKS

75. Most people have experience with hearing or making overly optimistic predictions regarding how much was likely to be accomplished within a set number of hours. This has been formally studied by academics. For example, "Exploring the 'Planning Fallacy': Why People Underestimate Their Task Completion Times" (Buehler, et al; Journal of Personality and Social Psychology 1994, 67:3):

Academics who carry home a stuffed briefcase full of work on Fridays, fully intending to complete every task, are often aware that they have never gone beyond the first one or two jobs on any previous weekend. The intriguing aspect of this phenomenon is the ability of people to hold two seemingly contradictory beliefs: Although aware that most of their previous predictions were overly optimistic, they believe that their current forecasts are realistic. It seems that people can know the past and yet still be doomed to repeat it. The phenomenon, we propose, is not peculiar to academics. In a classroom survey we conducted, students reported having finished about two thirds of their previous projects (M=68%) later than they expected.

76. The paper opens by describing construction projects that were delayed and over-budget, e.g., the Sydney Opera House, which was completed 10 years late at 14 times the originally planned cost. It closes with references suggesting that academics have been studying the tendency to be over-optimistic when making plans since at least the 1970s.

X. BACKGROUND: WHAT DO BUSINESS MANAGERS KNOW ABOUT SOFTWARE DEVELOPMENT RISKS

77. In my experience working with businesses that wish to have software developed, tailored, or maintained, the managers understand that every project carries a risk of cost-overrun, time-overrun, or outright failure.

78. The most authoritative book on software engineering is *the Mythical Man-Month,* published by Addison-Wesley in 1975 and updated in 1995. This book is required reading for many undergraduates in Computer Science, Software Engineering, and Systems Engineering courses. The author, Fred Brooks, was one of the managers of IBM's OS/360 operating system project. IBM was then the most sophisticated information technology company on the planet and OS/360 was its flagship software product. Brooks:

The effort cannot be called wholly successful, however. Any OS/360 user is quickly aware of how much better it should be. The flaws in design and execution pervade especially the control program, as distinguished from the language compilers .... Furthermore, the product was late, it took more memory than planned, the costs were several times the estimate, and it did not perform very well until several releases after the first. (page xi)

79. Did things improve over the following 50 years? The International Project Leadership Academy [9] summarized surveys done by organizations such as IBM, KPMG, McKinsey and the U.S. General Accounting Office. The IBM survey from 2008 found that "[o]nly 40% of projects met schedule, budget and quality goals." A McKinsey and University of Oxford study found that "17 percent of large IT projects go so badly that they can threaten the very existence of the company."

80. Given that the risk inherent in software development is well-understood both by customers and vendors, whenever software work is contracted to an outside firm the contract typically makes clear which party bears the risk of cost or time overruns. A contractor may work by the hour, in which case the customer enjoys the savings when a project is easier than expected or must pay extra and wait longer when a project is harder than expected. A contractor may work on a "firm fixed-price basis" in which case there is typically a specification phase, for which the customer will pay an agreed-upon amount, followed by a contract that software meeting the developed specification will be delivered. As it can take months to develop a sufficiently precise specification to be used in a contract, this firm fixed-price way of working will usually be slower than hiring a contractor to work by the hour. Also, the contractor will pad his or her estimates substantially so as to be sure of not losing money in the event that the project does turn out to have some unexpected complexities.

81. I have personal experience with both kinds of contracts. One easy way to see that a contract is being done on a customer-bears-the-risk basis is the presence of hourly rates. If the contract calls for Specification X to be achieved for $Y, there no reason for the contract to include an estimated number of hours or an hourly rate to be charged. Another indicator of where the risk is being placed is a description of deadlines and penalty clauses for what happens if a deadline is not met.

82. In this respect hiring a firm to do software development is no different than hiring a firm to do carpentry, plumbing, or construction. The agreement may be for an hourly fee or for a fixed price. If the parties agree and the contractor feels that he or she can understand all of the risks sufficiently, there may be agreed-to penalty clauses for late completion.

XI. BACKGROUND: HOW MUCH CAN MANAGERS KNOW?

83. Given the version control and bug tracking software described above, one might think that it would be typical for business managers to have, at all times, a good understanding of the current state of a project.

84. Steve McConnell, an authority on software engineering and project management with experience at Microsoft and Boeing, writes that this is seldom the case in practice:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

85. (from *Rapid Development,* Microsoft Corporation and O'Reilly Media, 1996; note that the MIT Libraries considered this book of sufficient continued relevance that they purchased a new copy as late as 2013 and maintained four copies on the shelf in the Barker Engineering Library)

XII. BACKGROUND: SOFTWARE SECURITY VULNERABILITIES

86. Securing Internet-accessible computer systems against attack is one of society's greatest current challenges. This is so challenging that in 2015 the Zurich Insurance Group estimated that "the annual cost of protecting our digital world from hackers will exceed the benefits of being connected by 2019." [10] There are entire academic degree programs devoted to computer security, e.g., at the University of Southern California. [11] Despite the best efforts of capable people, vulnerabilities continue to be discovered in software and hardware that we use every day.

87. *Building Secure Software* (Viega and McGraw 2002; Addison-Wesley), page xix, says "bad software is more common than you probably think. The average large software application ships with hundreds, if not thousands, of security-related vulnerabilities. Some of these are discovered over the years as people deploy the applications .... The rest of the software vulnerabilities remain undiscovered, possibly forever."

88. "Towards Evidence-Based Assessment of Factors Contributing to the Introduction and Detection of Software Vulnerabilities" (Matthew Finifter 2013, Technical Report UCB/EECS-2013-49) [12] is a PhD thesis describing an experiment in which 9 "professional programming teams" were given the same task to implement using different tools, including the PHP language that was used in the software at issue in this case. Each team wrote software for a 30-hour period and the software was then reviewed both manually and with an automated tool. Despite the small scale of these applications and the fact that popular and therefore well-tested toolkits were used, the typical implementation was found to have at least 5 security vulnerabilities and it was generally the case that not all of those found with the automated tool were found via manual inspection or vice versa (see Figure 3.1): "Out of a total of 91 vulnerabilities found by either technique, only 19 were found by both techniques (see Figure 3.3)."

89. One database of known software vulnerabilities, the "Common Vulnerabilities and Exposures" database at cve.mitre.org, includes 14,712 vulnerabilities discovered in 2017. [13] This list includes problems primarily with popular software, such as Microsoft and Apple products. Although a competent programmer may be able to identify at least some of the security problems with a given information system, there is no guaranteed procedure for finding all vulnerabilities within a fixed amount of time. In fact, a vulnerability that had lurked in Intel processors for 20 years was only recently discovered. See "Researchers Discovery Two Major Flaws in the World's Computers," *New York Times,* January 3, 2018 [14]:

Computer security experts have discovered two major security flaws in the microprocessors inside nearly all of the world's computers.

The two problems, called Meltdown and Spectre, could allow hackers to steal the entire memory contents of computers, including mobile devices, personal computers and servers running in so-called cloud computer networks.

Amazon told customers of its Amazon Web Services cloud service that the vulnerability "has existed for more than 20 years in modern processor architectures."

90. It is not reasonable to expect that an ordinary software developer will be able to quickly find all of the security problems in a system that a different programmer built.

XIII. BACKGROUND: GATHERING REQUIREMENTS

91. As noted above, unless a programmer is building software for his or her own use, it is necessary to find out what the customer and/or users want and need. There is no standard procedure for doing this and the degree of formality depends on factors such as the following:

• the complexity of the work to be done

• the number of programmers who will be working on the project

• the amount of time that the project is expected to require

• whether a contract is fixed-price or hourly

92. *Building Secure Software* (Viega and McGraw 2002; Addison-Wesley), page 17, describes the conventional academic wisdom that software should be engineered with a "well-structured process," but notes that "pressure to be first to market and retain what is known as 'mind share' compresses the development process so much that software engineering methods are often thrown out the window .... The Internet time phenomenon has exacerbated the software engineering problem .... Given the compressed development schedules that go along with this accelerated kind of calendar, the fact that specifics are often very poorly written (if they exist at all) is not surprising. It is not uncommon to encounter popular consumer-oriented systems that have no specifications." (emphasis added) In other words, from the point of view of the authors it would be ideal if everyone wrote software the FAA/Boeing way. Yet the typical business owner might rather have a "popular" product and the revenue associated with it than a gold star from a professor of software development methodology.

93. Even someone who went to the trouble of writing an entire book on the subject of gathering requirements acknowledges that a formal process may not be necessary or even beneficial. See, for example, *Designing Requirements* (Britton 2016; Pearson), page 12: "Ad hoc design is extremely common in IT, and I have used it myself a lot." (emphasis added) Does a more elaborate process guarantee success, in this author's view? "IT has used planned design a great deal, especially in large projects. I think it is fair to say that planned design has a mixed record of success in application design." (page 14, emphasis added)

94. As will be explained below, there were written requirements and specifications for the programming work to be done by Liquid. That said, there is no industry standard or agreement among programmers on what is an optimum process for gathering requirements or even whether there should be written requirements for a typical software development project.

XIV. WHAT WAS LIQUID HIRED BY PGH TO DO?

95. Based on my review of the April 23, 2015 contract (Exhibit C to the Golbeck Report), Liquid was hired to step in and perform maintenance on existing web sites operated by PGH. The agreement itself is titled "maintenance," not "software development" or "programming." The forthcoming "work requests" contemplated include "fixes" (page 2). The Golbeck Report states (page 10) that "Liquid agreed to completely all of the work in time for the NAFSA conference," but also notes that there was no "formal scope" and therefore there is no way that this maintenance agreement can be a commitment to a specific punchlist of items. In fact, the April 23, 2015 agreement explicitly contemplates that some of the work would be directed toward "Discovery" (see page 2) (Def. Discoverable Documents, Bates #000019), which is understood by people in the industry to mean determining what work should actually be done. The April 23, 2015 agreement contains a "Workflow" section on page 3. (Def. Discoverable Documents, Bates #000020). This explains that tasks will begin with a request from PGH ("ideally accompanied by documented requests of the work"), that Liquid may then seek clarification regarding the request, and that Liquid will then "work diligently."

96. The agreement refers to "an important conference in late May" on page 2 (Def. Discoverable Documents, Bates #000019) and a time period of 4/23/2015 through 5/31/2015 (page 4) (Def Discoverable Documents, Bates #000021). However, my understanding of this agreement, as someone who has been a software development contractor, is that Liquid's only obligations are to work for 80 hours at $165 per hour.

97. In fact, none of the other agreements that Liquid subsequently entered into with PGH contain a formal scope. For instance, the parties indicated that the second agreement, dated April 28, 2015, would "cover, but not be limited to, work requests," including "discovery around production and development"; "brand consulting and guidance"; "backend fixes within the PGHStudent site"; a few front-end UX and workflow fixes on the PGHStudent site"; "and more." (page 2) (Def Discoverable Documents, Bates #000033). Although the agreement did list various tasks "targeted to be completed before the NAFSA Conference," (page 4) (Def. Discoverable Documents, Bates #000035), it, like the original agreement, provided for 58 hours of support time at $165.00 per hour. (page 3) (Def. Discoverable Documents, Bates #000034). Therefore, my understanding of this agreement, as someone who has been a software development contractor, remains the same as my understanding of the original agreement - Liquid's only obligation under the agreement was to work for an additional 58 hours at $165.00 per hour.

98. For these reasons, my understanding of each of the subsequent agreements that the parties entered into, apart from a September 28, 2015 agreement to provide updates on Guarded Info. (Def. Discoverable Documents, Bates #000087-000091) and an October 13, 2015 agreement for Logo Development (Def. Discoverable Documents, Bates #000093-000095), remains the same - PGH was contracting with Liquid to provide a certain number of *hours* of support services. See, e.g. "PGH and First Risk Advisors Web UX - Scope Change," dated May 7, 2015 (page 2) (Def. Discoverable Documents, Bates #000038) (indicating that Liquid would continue to provide PGH various services, including but not limited to, discovery, brand consulting and guidance, back-end fixes, "a few front-end UX and workflow fixes and more"); "PGH Student and First Risk Advisors Security and Top Priorities Maintenance Agreement," dated May 28, 2015 (page 3) (Def. Discoverable Documents, Bates #000043) (indicating that Liquid would provide PGH with "171 hours of support time at a discounted rate of $165.00" to complete various tasks, with rate calculations, specified dates, purchase process, and exports being top priorities); "PGH Student and First Risk Advisors June 2015 Maintenance Agreement," dated June 22, 2015 (page 3) (Def. Discoverable Documents, Bates #000405) (indicating that Liquid would provide PGH with "160 hours of support time at a discounted rate of $165.00" to complete various tasks that could be modified by PGH as needed or requested); "PGH Student and First Risk Advisors August 2015 Maintenance Agreement," dated July 31, 2015 (page 3) (Def. Discoverable Documents, Bates #000084) (also indicating that Liquid would provide PGH with "160 hours of support time at a discounted rate of $165.00" to complete various tasks that could be modified by PGH as needed or requested); "PGHStudent.com Redevelopment Initiative SOW," dated December 8, 2015 (page 10) (Def. Discoverable Documents, Bates #000133) (indicating that Liquid "felt confident" that it could have a full site and code structure ready for testing "contingent upon meeting agreed upon review and feedback timelines" by February 1, 2016, at a cost of $39,600.00, or an estimated 240 hours at $165.00 per hour); "First Risk Advisors 2016 Brand Maintenance Agreement," dated January 25, 2016 (page 5) (Bates #000151) (providing that Liquid would engage in approximately 43.75 hours of "maintenance work" per month, at a rate of $150.00 per hour);"First Risk Advisors 2016 Project Services Agreement," dated January 26, 2016 (page 5) (Bates #000145) (indicating that Liquid would engage in approximately 52.5 hours of "project-related work" per month, at a rate of $150.00 per hour); PGH Existing System Retrofit (pages 2-3) (Bates #153-54) (indicating that Liquid would spend approximately 308 hours preparing PGH's existing website for the upcoming enrollment periods at a cost of $130.00 per hour, for a total of $40,040.00).

99. Based on my review of the documents produced in this case, it would not have been possible for PGH and Liquid to enter into the kind of contract that the Golbeck Report assumes the two parties had. PGH's business requirements were evolving too rapidly for a set of rigid specifications to be used. PGH's deadlines for source code changes were too tight for the two companies to spend one or two months working collaboratively on specifications for the various modifications contemplated (even if PGH had wanted to pay for this kind of non-coding work). Thus, since neither party could know exactly what would be done during the course of a contract, the only sensible approach was to make a contract for a set number of hours at a set hourly rate.

XV. WHAT IS THE V1.0 SYSTEM THAT LIQUID WAS GIVEN TO MAINTAIN?

100. The source code for the v1.0 system as provided to Liquid in April 2015 was made available to me for inspection by Leigh Jamgochian, a Liquid employee.

101. The directory received from FocusMX for the PGH "student" site contains 7,839 files occupying approximately 100 MB.

102. In order to get at least a rough characterization of the contents of this directory, I ran the cloc Perl script [15] on the directory and received the following output:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

103. Note that the above report shows nearly 1 million non-comment lines of "code," but not all of the above "languages" are actually programming languages, e.g., JSON and XML are data storage formats while HTML is a markup language for web pages. There are nearly 400,000 lines of PHP code. The cloc tool eliminates duplicate files from the count. If a programmer were to look at only the PHP code and the associated comments, it would be more than 600,000 line or roughly 12,000 single-spaced printed pages.

104. I found two installations of the Wordpress toolkit within this directory. One was WordPress version 3.5.1 in the /blog subdirectory. This version was released on January 24, 2013. [16] An additional installation of WordPress was within the /student directory. This was version 4.1.1, released Feburary 18, 2015. [17] Despite the presence of two installations of the WordPress toolkit, the customized features of the site do not appear to have been implemented as WordPress plugins. The /student/wp-content/plugins directory, for example, contains only public-domain plugins. The same is true for the /blog/wp-content/plugins directory.

105. Software that seems to be specific to PGH's business is found, for example, at the top level, underneath the /admin/ subdirectory, and in /admin/includes/.

106. My understanding is that this software was supplied without a corresponding version control system repository. It would seem that conventional version control was not being used by the programmers at FocusMX because they would leave multiple versions of the same file within a directory. For example, the "home page" for the site is "index.php" and the directory also contains "index.php_01aug2013" and eight additional variants.

107. Below are screen shots showing some of the structure of the software as delivered to Liquid.

108. One characteristic of the v1.0 software was that, for "white label" sites, the entire code base would have to be copied and then modified to deliver a slightly different user experience and/or different design and graphics. A disadvantage of this "copy and paste" approach to software engineering is that, in the event that a security or business logic problem is found in the code, any fix that is developed must be propagated to all of the previously copied codebases.

109. Another disadvantage of the v1.0 system was that there were multiple versions of the student database being run. Thus a customer could potentially go to three different PGH-run sites and receive three different quotes for the same insurance (different schools had negotiated different rates). A student who transferred from one school to another, e.g., for a subsequent academic year, might be starting over on a new PGH-operated web site and there was no way to relate that customer to his or her previous record unless the customer happened to use the same email address when registering at the new site.

110. An additional disadvantage of the v1.0 system was that "protected health information" (PHI) regulated by the federal **HIPAA** law was mixed into the same database as information that controlled the web site's behavior and appearance. Thus,

programmers and administrators with no need to access PHI would have access to PHI. This is the kind of practice that **HIPAA** seeks to prevent, e.g., with the "Minimum Necessary Requirement," 45 CFR 164.502(b), **164**.**514**(d). Here's a U.S. Department of Health & Human Services explanation of the regulation:

The minimum necessary standard, a key protection of the **HIPAA** Privacy Rule, is derived from confidentiality codes and practices in common use today. It is based on sound current practice that protected health information should not be used or disclosed when it is not necessary to satisfy a particular purpose or carry out a function .... For uses of protected health information, the covered entity's policies and procedures must identify the persons or classes of persons within the covered entity who need access to the information to carry out their job duties, the categories or types of protected health information needed, and conditions appropriate to such access. [18]

111. Technical means of complying with regulations tend to be more reliable than relying on humans, who can be inconsistent. For example, Liquid began working with the production database for the legal v1.0 site in April 2015. **HIPAA** required that get business partners with access to PHI to sign agreements, but it was not until May 31, 2015 that David Opperman emailed Liquid to ask the company to sign **HIPAA**-required agreements. This is a great illustration of why, whenever possible, it is better to rely on separate databases and similar technical measures than on humans to remember to obtain signed documents.

XVI. WHAT DID LIQUID DO IN ITS WORK ON THE V1.0 SYSTEM?

112. Upon receiving the v1.0 code, Liquid established a standard version control repository so as to track all of its changes to the legacy system going forward. As discussed above, the use of a version control system facilitates collaboration among multiple contributors and testing. It also permits the "backing out" of any changes in the event that a change to source code results in unanticipated negative effects on a web site.

113. Liquid established development servers for use by its programmers and a staging server for testing. This enabled any changes to the v1.0 system to be tested in a staging environment before being "pushed live" to the production server. The use of a staging server is a conventional practice among competent web application developers.

114. Once version control, development servers, and a staging server were in place, Liquid was in a position to begin making changes to the v1.0 source code.

115. Based on my review of the documents in this matter, in making these changes, Liquid did what PGH asked it to do, just as contemplated in the contracts. Here is an example task put into the DoneDone bug/issue tracker set up by Liquid for PGH's use:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

116. In the above exchange, on May 23, 2016 at 8:59 am, Amber Opperman from PGH raises an issue regarding a problem exporting information from the PGH database. Leigh Jamgochian responds at 10:16 am on the same day that he has added generated IDs for 63 students. He marked the issue "Ready for Retest." At 2:00 pm that day, Amber Opperman used the DoneDone system to mark the issue as "Fixed."

117. It seems as though sometimes requests for work on the site would come in via email. For example, on July 27, 2015 there was an email exchange that seems to have been started by Brianna Melching of PGH, noting a problem with exporting data regarding certain students. Leigh Jamgochian emails PGH and FRA employees his analysis of the problem and a proposed fix:

I've managed to find the issue with the export which is missing the "last day"

The Coverage End Date that is used for some of the calculations comes from the Students Table. The Period Coverage End Date used for other calculations comes from a different table (student coverages) The Coverages table was the one day short.

To resolve this I'm doing the following logic

If the Coverages Table has an entry that is a DAILY code (i.e. NX) and a start date >=7/1/2015

Which

Does NOT have a coverage_end that matches the coverage_period_end_date date

Set the coverage_period_end_date date equal to the coverage_end

Recalculate the Coverage Duration for that entry.

This will work because the DAILY rate is ALWAYS the last few entries in the system. (After Yearly for 2015± school year and after Yearly, Monthly, Weekly for earlier)

118. Two hours later, Mr. Jamgochian emails "[t]he patch for the missing students has been rolled out to all the sites (Compass and White Labels)" and David Opperman of PGH responds with "Gotta love it when a plan works!!!" Brianna Melching of PGH closes out the exchange with "Just did another export for the dates 7/24/15 - 7/27/15 for Compass. Looks like all the students exported that we have received payments for!!!!!! This is GREAT news!"

119. Liquid might learn about seemingly significant changes to the business via email. For example, when PGH contracted with Liquid it was a site for college students, aged 16 and over. On August 18, 2015, however, Shane Opperman of PGH emailed Leigh Jamgochian "I was wondering if we can allow students under the age of 16 enroll. We have taken on some High school account and need this ability as soon as possible. Thanks." Note that Mr. Jamgochian had the role of software developer, not project manager. Therefore, at least for this project, PGH was directing Mr. Jamgochian's efforts and thereby necessarily taking on the responsibility that some other task on which Mr. Jamgochian had been working would have to be deferred.

120. An additional example of a PGH employee telling a Liquid programmer what to do and when to do it is a June 16, 2015 email from David Opperman to Leigh Jamgochian saying "Important that we fix this sooner rather than later" with respect to a newly discovered rate discrepancy issue.

121. Similarly, on July 31, 2015 at 7:43 am, Dave Opperman of PGH emailed Leigh Jamgochian to ask him to modify the logo displayed on one of the PGH web sites that Liquid was maintaining. Based on a forwarded email, the logo had been received by PGH at 11:55 pm the previous evening. Mr. Jamgochian responded at 8:25 am that the change had been made.

122. (There is nothing improper about a company hiring contract programmers by the hour and then telling them what to do on a day-to-day and task-by-task basis, of course. But it is unreasonable for the company, having given the programmers a lot of last-minute work, to later complain that the contract programmers were not sufficiently organized in working toward a previously discussed goal. Changing the logo on a web site is not a challenging task, but any time spent on that task cannot be spent on a previously drafted grander plan.)

XVII. DID THE V1.0 SYSTEM WORK?

123. My understanding is that Liquid was hired in April 2015 and ceased its work on and responsibility for the live PGH sites in October 2016. That's nearly 18 months. My understanding that the v1.0 system operated with only a handful of brief outages throughout the Liquid-PGH relationship despite hundreds, if not thousands, of changes to the source code that had been made by Liquid at PGH's behest. This kind of reliable operation in the face of added features, updated business logic, and other changes to source code is not a given. That Liquid was able to deliver this is evidence of general competence and also of at

least a reasonably fair exchange between PGH and Liquid. PGH did not have its own programming staff. My understanding is that Liquid was paid roughly $270,000 during this period. Based on my experience in this industry, this would not be an unreasonable amount for the operator of an ecommerce site to pay for site maintenance and upgrades over an 18-month period. (And, in any case, my understanding is that Liquid included within that amount some graphic design,banner production work, and video work. My understanding is that PGH expressed satisfaction with all of this work from the Liquid design staff.)

124. The Golbeck Report takes Liquid to task for being short of perfect, but it neglects to point out that there is a wide spectrum of performance in the world of IT. How bad can it get? The technical staff at Danger, an acquired subsidiary of Microsoft, made mistakes in administering the RDBMS supporting the T-Mobile Sidekick smartphones. The result was temporary or permanent data loss for 800,000 users. [19] This was described in an industry publication as "an almost incomprehensible data disaster." [20] The cause of this data loss was attributed to (a) inadequate backup procedures, and (b) a botched upgrade of an array of hard disk drives. An additional example is the attempted upgrade of a commercial software package supporting some operations at the Royal Bank of Scotland. The upgrade was "backed out," but it took the bank three business days before operations were back to normal: "By the time operators finished a successful run, millions upon millions of customer transactions were waiting to be processed. Customers will continue to experience problems until the bank works through this massive backlog of transactions." [21]

125. Evidence that Liquid's work on the v1.0 site was competent and acceptable to PGH includes the following: (a) the PGH public sites were generally up and running, processing orders, and generating revenue for PGH, (b) PGH did not seek to replace Liquid for maintenance and extensions to the v1.0 site.

XVIII. WHAT IS THE V2.0 SYSTEM THAT LIQUID PRODUCED?

126. The V2.0 source code was made available to me in a source code directory named "fra." Based on my inspection of the source code, this is WordPress version 4.4.2 (released February 2, 2016 [22]) that has been extended using the standard WordPress plugin API (Application Programming Interface). This is documented in the WordPress Plugin Handbook, [23] which explains

If there's one cardinal rule in WordPress development, it's this: Don't touch WordPress core. This means that you don't edit core WordPress files to add functionality to your site. This is because, when WordPress updates to a new version, it overwrites all the core files. Any functionality you want to add should therefore be added through plugins using approved WordPress APIs.

Plugins allow you to greatly extend the functionality of WordPress without touching WordPress core itself.

Plugins are packages of code that extend the core functionality of WordPress. WordPress plugins are made up of PHP code and other assets such as images, CSS, and JavaScript.

By making your own plugin you are *extending* WordPress, i.e. building additional functionality on top of what WordPress already offers. For example, you could write a plugin that displays links to the ten most recent posts on your site.

At its simplest, a WordPress plugin is a PHP file with a WordPress plugin header comment. It's highly recommended that you create a directory to hold your plugin so that all of your plugin's files are neatly organized in one place.

127. Liquid seems to have carefully followed the WordPress standards. There are three custom-developed plugins in the /wp-content/plugins subdirectory: fraAdmin, frapp, and FRAForm.

128. Liquid seems to have followed the U.S. Department of Health & Human Services guidelines regarding **HIPAA** compliance. Application data were split into three separate databases: fraCommon, fraPHI (for the **HIPAA**-covered information), and fraLogs. /fraapp/lib/config/defines.php, lines 4-6:

// Databases

define(‘DB_PGH_COMMON’, ‘fraCommon’);

define(‘DB-PGH_PHI’, ‘fraPHI’);

define(‘DB-PGH_LOGS', ‘fraLogs');

The file /fraapp/lib/pgh/app/queries/queries_phi.php contains examples of the fraPHI database being used (note the “connectToDb” command below):

public static function get Purchases From Individuals($individual id) »

$db = self::connectToDb(DB_PGH_PHI);

$purchases = [];

$collection = null;

$query = “SELECT * FROM purchases WHERE individual id =

‘$individual id’ ”;

$result = $db->query($query);

$purchases = $result->fetchAll();

$collection = self::buildCollection(DB_PGH_PHI,

‘pgh/data_objects/fraphi/Purchases', $purchases);

$db->disconnect();

return $collection;)

The file /fraapp/lib/pgh/data objects/common/rates.php contains examples of the fraCommon database being used (note the “lookup_db” line below):

$overrides[‘deductible id’] = array(

‘type’ => ‘deductible _name_display_link’,

‘lookup_db’ => DB_PGH_COMMON,

‘lookup table’ => ‘plans',

‘lookup_filter_field’ => ‘parent_type_id’,

‘lookup-filter sort’ => ‘sort’,);

The above source code shows that it would be possible for a PGH employee, for example, to adjust the deductible associated with an insurance policy without ever having access to the fraPHI database containing **HIPAA**-covered information.

129. The v2.0 fra directory comprises 3,918 files occupying 120 MB of space. cloc found a total of 279,463 lines of PHP code within this directory:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

130. Within the three plugins mentioned above, fraAdmin, frapp, and FRAForm, cloc found roughly 52,000 lines of PHP code, over 9,000 lines of JavaScript. and over 3,200 lines of CSS code. Some of this is open-source code, not authored by Liquid, but the bulk of the PHP is work by Liquid. The most substantial module of non-Liquid is contained within the /fraapp/lib/pgh/authorizenet/sdk... directory. This is code intended to assist developers building ecommerce systems that actually bill customer credit cards. It contains 12,151 lines of PHP code. Thus it is reasonable to say that Liquid developed approximately 40,000 lines of PHP code for this v2.0 project. I have attached as Exhibit 2 the output of the cloc script with the “--by-file” option selected.

131. Based on my review of the v2.0 software, it is a complete system that is capable of welcoming a customer, offering various insurance options, collecting information necessary to sell a policy, and finally billing a credit card and recording the purchase of a policy in the RDBMS.

132. /fraAdmin/lib/pgh/adminplans.php is the source code file that enables an administrator to enter information regarding a new plan to be offered.

133. /fraapp/lib/pgh/app/wp/adminpages.php and /fraapp/lib/pgh/app/wp/page_ actions/editpage_go.php contain code that process the submission, which is ultimately inserted into the database by the validate_form function invoking a DTOHandler, defined in /fraapp/lib/pgh/app/dtohandler.php.

134. When a customer comes to the site to purchase a policy, the code that collects information from the student is organized in /FRAForm/form_body.php. This software pulls information from files within a “steps” subdirectory and makes them available on one browser page:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

135. To the person who has never worked as a programmer, it seems doubtful that the above source code will make gripping reading. Nonetheless, there is evidence here of good craftsmanship. There are six sections of HTML code that will require substantially identical framing. What's different in each section is pushed into separate files, e.g., /steps/pay.php. Instead of repeating substantially the same HTML six times, with the risk that if there is a mistake it might be fixed in four or five of the repeats rather than in all, the HTML is generated by “foreach” loop that runs through each step. There is error handling. If one of the files in the /steps subdirectory is missing, the software will check for that (see the “file_exists” call, above) and display an “invalid step” error message.

136. Individual steps in the above process may contain code that requests information from the RDBMS. For example, / fraapp/lib/pgh/api/endpoint/schoolPlans.php is invoked during the Product step (see line 82 of FRAForm.php) and it ultimately relies on a MySQL stored procedure sp_getSchoolPlans to get information from database tables. The use of stored procedures is a good practice for ensuring consistency across multiple applications. Rather than three or four application programs having to be kept in sync if there is a change in the way that PGH decides which plans shall be offered to students from a particular school, if the stored procedure is updated then automatically all of the application programs will display the correct information.

137. The final step in the process is “Confirm.” This invokes the function submitForProcessing defined at line 542 within /FRAForm/j s/fraform.js. The actual credit card processing is done by software within /fraapp/lib/pgh/api/endpoint/processCC.php. At line 32 an insertion is made into the RDBMS purchase table and at line 49 there is an insertion into the payments table. This code relies on an API and library code published by Authorize.Net, a division of Visa that operates an Internet credit card gateway. The Liquid-developed software is modularized in such a way that, in the event that PGH decided to switch to a different credit card processor, it would be a straightforward modification process to work with a company other than Authorize.Net. This is an example of good software engineering practice.

XIX. HOW MANY HOURS DID LIQUID BILL TO PGH FOR WORK ON V2.0?

138. Based on a spreadsheet titled “FRA Project Audit” prepared by Liquid staff, my understanding is that PGH was billed for 240 hours of work on the v2.0 system, a project named “PGHStudent.com Redevelopment.” The cost for this work was 240 times $165 per hour or $39,600 total. It is a shame that, like many other software development projects since the invention of digital computers, the v2.0 project cannot be considered a success. However, even if one were to accept PGH's allegations that Liquid was 100 percent responsible for the project failure, based on my industry experience and the written contracts in place, in my opinion there is no possible basis for PGH demanding anything more than a refund of the $39,600 paid.

XX. DID THE V2.0 SYSTEM WORK?

139. I connected to a running version of the v2.0 system, lacking only the connection to the Authorize.Net credit card billing system, in three ways: (a) as a potential customer, from a desktop Google Chrome browser, (b) as a potential customer, from an Apple Safari browser on an iPhone 7 Plus, and (c) as an administrator, from a desktop Google Chrome browser.

140. Here is the home page on desktop then mobile:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
As may be seen above, similar information is present on the two screens, but arranged differently. This is an example of “responsive” design in which a site tailors itself to the connected user's device.

141. From my mobile browser, I found that I was able to select a state (Massachusetts) and was then shown an updated menu of schools within Massachusetts:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
I was able to enter a date range for coverage, add or remove a child, and get a selection of plans, then a rate quote:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
142. My experience as a potential customer from a desktop browser was similar, but with a different layout:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
143. When I connected to the admin site, I found the familiar WordPress admin pages with four Liquid-authored plugins:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
144. I found an administrative page on which a PGH employee could change text on the site:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
145. I found an administrative section from which a non-technical PGH employee could update plan information:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
146. I found forms from which a PGH employee could change the financial information regarding an insurance policy:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
147. I found an administrative page from which a PGH employee could maintain the categories of customers:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
148. Clicking on the "Add New" button above resulted in a page from which a non-programmer could add a category of customer:

149. I found an administrative page on which a PGH employee could change text on the site:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
150. After I added this "insured group" I found that it appeared in the preceding page's list of entries pulled from the RDBMS.

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
151. I was able to delete the "SeniorStudent" row that I had created.

152. Based on my testing, the v2.0 source is at least basically functional and enables non-programmers, such as members of PGH's team, to maintain the significant business information, such as what rates will be charged to which categories of customers. The design of the site also enables much of the text that will be shown to users to be edited by a non-programmer.

XXI. THE GOLBECK REPORT

153. I have reviewed the Export Report of Jennifer Golbeck, Ph.D. (the "Golbeck Report").

154. The report lacks any technical analysis and it is unclear if Dr. Golbeck reviewed any of the source code, despite the fact that Page 2 of the Golbeck Report says that she was "engaged by PGH to review certain code that Liquid provided to PGH sometime in May/June 2016 ('the Version 2.0 Code')."

155. The primary technologies and languages used in both v1.0 and v2.0 were the WordPress toolkit, the PHP and SQL languages, and the MySQL RDBMS. The custom-developed software on the server was also in PHP and SQL, for both v .0 and v2.0. The architecture of the v2.0 system was a WordPress Plugin. None of the following terms are used in the Golbeck Report: WordPress, PHP, SQL, MySQL, Plugin, RDBMS. Imagine an "analysis" of a desktop computer user's environment that did not mention whether the device on the desk was running the Chrome, Macintosh, or Windows operating system. Or an "analysis" of the novel *Anna Karenina* that did not mention that the book had been written in the Russian language.

156. Golbeck does not distinguish among four different types of programming: (1) fixing bugs in an existing system, (2) adding a small feature to an existing system, (3) developing a new information system based on a toolkit, and (4) developing a new information system from scratch. Her ideas make the most sense with regard to the fourth type of programming (developing software from scratch), but there is no evidence that either Liquid or PGH was ever engaged in this type of project.

157. The amount of process, planning, and documentation that is conventional and appropriate depends on which of above four types of programming is being done. A simple way to understand this is to suppose that an ecommerce site is down due to a previously undiscovered error in the code. No customers can place orders. The Golbeck Report says "Before writing new code or modifying any existing code, it is imperative for any developer to have a detailed plan for how to complete the work." (page 25, emphasis added) Imagine the programmer who heeded this unconditional rule and told the site owner "I think that I've found the bug. I'm going to spend today writing up a detailed plan to modify the existing code so as to fix the bug. I'll stop at Starbucks tomorrow morning and then come in fresh to start modifying the code in accordance with the detailed plan." Much of what is in the Golbeck Report regarding the software development process is conventional (albeit often not useful) wisdom when software is being developed from scratch (Type 4) and where avoiding bugs is more important than getting into the marketplace quickly. A software engineer at Airbus or Boeing, for example, or a regulator at the FAA, would likely find much to agree with. However, these are folks for whom time has a different meaning than for a web site operator. The Boeing 787 project begun in 2003 based on engineering work actually started in the 1990s. The plan was to deliver customer airplanes in 2008. The B787 first flew in 2009 and was first delivered to a customer in 2011, three years later than planned. Note that, despite the requirements to follow an FAA-approved software development process, at least some of these delays were blamed on software development not proceeding as planned. [24]

158. PGH was not subject to FAA regulation. PGH did not want to wait 8 years from project inception to customer launch. In fact, the first project for which PGH hired Liquid contemplated software modifications to be done and pushed live to customers within a month or two. For those business reasons, as well as the fact that no Type 4 (from scratch) software development was being done, comparing the work done at Liquid to the standards set forth in the Golbeck Report does not make any technical sense.

159. In any case, to the extent possible consistent with the available hours, the timeline imposed by PGH's business needs, and the responsiveness of the PGH managers, Liquid actually did gather written requirements, did produce system architecture documents and plans, did develop written plans for software development work, and did document code produced. Liquid's work was not out of line in these respects with what is conventional for Web development, especially when starting from a toolkit such as WordPress. The Golbeck Report offers no analysis of these documents and does not explain in what ways, if any, they are deficient. Indeed, as with the source code, the Golbeck Report does not reference these documents and therefore it is unclear whether Dr. Golbeck reviewed them as part of her analysis.

XXII. THE GOLBECK REPORT ON SOFTWARE DEVELOPMENT METHODOLOGIES

160. Starting on page 27, the Golbeck Report includes a summary of software development methodologies.

161. The textbook cited in the Golbeck Report, *Software Modeling and Design* (Gomaa 2011), was not intended to cover web development projects based on a toolkit such as Wordpress. Page 29 of Gomaa describes the management and technical methods of software engineering as being applicable to "large-scale software system[s]" (emphasis added). Developing the fly-by-wire flight control software for an airliner or fighter jet fits into this category; a 6-week project to fix bugs and add features to a web site does not.

162. Even if Gomaa were a book that was intended to cover Wordpress-based web development or site maintenance, it does not support the Golbeck Report's assertions or conclusions. For example, Gomaa page 30: "... the waterfall model presents the following significant problems: [1] Software requirements, a key factor in any software development project, are not properly tested until a working system is available to demonstrate to the end-users .... [2] A working system becomes available only late in the life cycle. Thus a major design or performance problem might go undetected until the system is almost operational, at which time it is usually too late to take effective action .... evolutionary prototypes can help resolve the second problem."

163. Gomaa advocates “Evolutionary Prototyping by Incremental Development” (page 34), which is essentially the process that Liquid was using: “a form of incremental development in which the prototype evolves through several intermediate operational systems ... into the delivered system.”

164. Gomaa defines “software architecture” as “A high-level design that describes the overall structure of a system in terms of components and their interconnections, separately from the internal details of the individual components.” (Glossary, page 533)

165. *Software Engineering,* 8th edition, by Ian Sommerville (2007; Pearson), is similarly skeptical: “the waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development.” (Section 4.1.1) The documentary record shows that PGH, like other companies that do business on the web, had needs that could change from month to month or even day to day. Professor Sommerville, like Gomaa, advocates for “evolutionary development” (Section 4.1.2): “An evolutionary approach to software development is often more effective than the waterfall approach in producing systems that meet the immediate needs of customers .... For small and medium-sized systems (up to 500,000 lines of code), I think that the evolutionary approach is the best approach to development.”

166. Liquid had no choice but to pursue evolutionary development on the v1.0 system: Liquid was specifically hired to make changes in hopes of evolving v1.0 into something that would work better and meet PGH's evolving business requirements.

167. For v2.0, the documentary record shows that Liquid did pursue evolutionary development and the cloc numbers (above) show that the Liquid-developed lines of code within v2.0 were on track to be smaller than the 500,000 lines of code threshold set forth by Professor Sommerville.

XXIII. WHAT SPECIFICATIONS, REQUIREMENTS DOCUMENTS, BLOCK DIAGRAMS, AND OTHER PLANNING DOCUMENTS WERE WRITTEN FOR V1.0 MAINTENANCE?

168. Liquid seems to have taken careful notes during a 4/24/2015 kickoff meeting. (Def. Discoverable Documents, Bates #000028-31). Some of these contain specifications from which a programmer could start work. For example, the second page states that PGH wants to be run reports “themselves” (reasonable to infer that this should be from an admin web page) and that the format of the reports should be csv (comma-separated values, readable by Microsoft Excel and other spreadsheet programs). The same page also talks about information to be added to emails that the site is sending out, that the application should be programmed to generate an ID number for students who don't provide one.

169. The July 31, 2015 PowerPoint presentation titled “FRA Technology” (“July PPT”) contains block diagrams showing the high-level structure of the legacy PGH code:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

170. Contrary to what is alleged in the Golbeck Report, it seems that the Liquid developers did come up with written plans prior to writing code. For example, here is a June 10, 2015 email from Leigh Jamgochian to fellow employees at Liquid:

Ok this is the late afternoon update.

The Export isn't working properly. The dependents are not being output correctly. I've told David this and will be reworking it tomorrow and returning on Friday.

I wasn't able to get to the DoneDone stuff today but will be working on it tomorrow in preparation for meeting with David and Shane on this.

Part of the complexity is that the database structure is inconsistent. For example

osi_plans = Plans that students sign up for.

osi_futureplans = Student list of future users.

osi_students= Current Student List

osi_students_old_record = Old student list

Fields are inconsistent across the tables also such as id, old_id, and sid are all valid IDs for a student.

So it's really not a fun thing to track this down. The filters are working but just getting these dependent records consistent, accurate and not duplicated is being a pain.

Completed today

• Fixed yearly rates

• Rolled out Yearly Rate fix

• Testing of Export (failed due to dependents)

So for tomorrow

• Working at Liquid (in late by 10:30)

• Fixing this export (will try and leverage Sean with this and any other devs...)

• Compiling list of outstanding items. Poss. coordinate with Justin and Shane, David

• Merging password fixes into the current staging and testing

Friday

• At FRA

• Test and Roll out of Password fixes

• Test and Roll out of Export

• Level Set task list with client

Thanks!

171. The above email describes four RDBMS tables, e.g., osi_plans and osi_ students, and names some of the columns within those tables, pointing out that, contrary to standard RDBMS practice, there can be multiple column names that serve as IDs for a student (the standard practice would be to have a "student_ id" column in every table where it was necessary to refer to a student).

172. On some occasions, an organized list of requirements was sent to Liquid via email. For example, on June 18, 2015, David Opperman forwarded to Leigh Jamgochian, cc'd to two other Liquid employees, a 30-row spreadsheet containing detailed requests for modifications to the PGH Chinese Plan site, e.g., "Your school currently auto-populates to Chinese Service Center; however, the client has asked if this field can be changed to allow the student to type in their school's name?" Based on the email text it seems that Mr. Opperman received this email from Jacqueline Waldron, an employee of UnitedHealthcare StudentResources.

XXIV. WHAT SPECIFICATIONS, REQUIREMENTS DOCUMENTS, BLOCK DIAGRAMS, AND OTHER PLANNING DOCUMENTS WERE WRITTEN FOR V2.0 DEVELOPMENT?

173. Liquid produced a 25-page specification document, dated 9/25/2015, Def Discoverable Documents, Bates #000096-123. The document describes the overall structure, including the directory and file structure, of the legacy v1.0 site. It contains four pages of codes that are used on the site, e.g., for interaction with the credit card processor or "VFS" for "Visiting Faculty/Scholars." The document specifies, on page 8, that "Saved information [in server logs] must be filtered for PCI/PHI information." Page 9 of the document contains a flowchart for processing payments and a description of additional functionality that is required. Page 10 describes and illustrates a new structure for the database of insurance plan data. Page 13 documents the way in which rates will be calculated. Pages 16-18 describe a role-based access control system, including a full-page table of contemplated roles and capabilities. Page 20 sketches the capabilities of the administrative pages and highlights the situations in which **HIPAA**-regulated data will become available to an administrator. Page 21 is devoted to the problem of ensuring that the same code base can serve pages in multiple languages. It is flagged as unfinished (at least as of 9/25/2015) and there are links to potential resources for WordPress plugins and documentation that may be useful. Page 23 lays out the situations in which the application will send out email notifications. Page 24 is a glossary that is more critical than it appears. For example, every speaker of English knows what a "student" is. But what does it mean to be a "student" in the database of this application? Page 24 lays this out: "Primary customer of the system. Must be enrolled in a school. Must have an email address and may or may not have a student ID. Their records that contain healthcare or identification information are considered PHI." From this "glossary" entry, a programmer knows that the database table holding "students" should have a NOT NULL constraint on the email address column, but that the student ID column may contain NULLs. The programmer is also alerted by this documentation to the requirement that this be stored in the fraPHI database so that **HIPAA**-regulated information does not become accessible to those without a need to see it.

174. Liquid seems to have obtained at least some of the business requirements and rules from PGH via email. For example, on January 9, 2016, Liquid received an email from David Opperman with a subject of "Enrollment rules" containing such information as "We would like to ability to set up automatic recurring charges for coverage for studnets. So say we allw monthly coverage, we would set up onthly recurring charges, or quarterly or every 90 dayes whatever. System will need to check age at each recurring charge to make sure someone doesnt cross over into a new age bracket." The email referred to previous emails, e.g., "This was covered in a separate email earlier." I infer from this that there was a continuous flow of written requirements via email from PGH to Liquid.

175. The January 9, 2016 email contained a vague description of possible order status: "The problem here is where do people stand in their purchase process. Some start and do not finish. Some buy today for coveraeg that may not start for several months. We need a way to checdk on all pending purhcses, not completed purchases etc. System currently has a future purchase status but not really sure what it means or how it works. If someone buys a plan, wheter or not it starts today or in 3 months, they should be charged when they buy and they should export immediately." The response, dated January 10, 2016, from Mr. Jamgochian, turns this into a precise requirement for seven possible "Purchase Status" values, e.g., "Plan Selected" and "Payment Processed."

176. This is precisely the kind of organized requirements gathering and software planning that the Golbert Report asserts was never done by Liquid.

177. The July PPT contains a block diagram of the proposed v2.0 structure:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
178. The July PPT also contains a high-level written description of the overall plan for v2.0:

Utilize WordPress

• Display code is all within a WP theme.

- Flexible display platform

- Independent of the content and data

- Easy to update/change

• Functional Code is within WP Plugins

- Operates outside of display

- Consistent code across different sites

- Update process is simplified

179. As discussed above, the source code in the /fra directory is consistent with the above plan being followed.

XXV. WHAT WAS THE QUALITY OF WORK DONE BY LIQUID? (THE DATA IMPORT PROJECT)

180. Liquid did one project for PGH for which there was no legacy code and only limited input from PGH was required. The code for this project is found in the "data-import-fra" directory. My understanding of the purpose of this project is that data were gathered back from the insurance company actually writing the policies sold on the site. These data had not been previously recorded by the PGH server database management system. The goal was to put data back into the server's MySQL database so that when a student returned to purchase insurance he or she would find forms pre-filled. Mr. Jamgochian informed me that the software was built, run, modified with feedback from PGH, run again, etc. This is a conventional iterative or "evolutionary" development process.

181. The cloc Perl script reports that this directory contains 2088 lines of code and 127 comment lines. According to Leigh Jamgochian, approximately 80 hours of time was spent by Liquid on this project and billed to PGH. The git log entries related to this project show that the following checkins were made between June 9, 2016 and June 27, 2016:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE
182. By itself, the fact that the Liquid developers used a version control system for code that would have to be used only once is evidence of a reasonable amount of care. The git checkin comments above show that bug-tracking was in place for this project (see the 6/17/2016 checkin noting that "bugs #52 through #56" were fixed). This is confirmed by the data at DoneDone, showing that a total of 77 issues were logged on this project between May 5, 2016 and June 30, 2016. Here is one example:

TABULAR OR GRAPHIC MATERIAL SET FORTH AT THIS POINT IS NOT DISPLAYABLE

183. The scripts to be run are in a subfolder called “steps” and the files are prefixed with the step number, e.g., “00_load” and “50_AddMissingStudents”.

184. The task of”50_AddMissingStudents” is copying information from a temporary “import” table to a permanent “osi_students” table. The column or “field” names are not the same in these two tables, however. It would have been possible to “hard-code” the respective column names into a 50-line or 100-line SQL statement. However, this would have been ugly and fragile, notably because the “source” and “target” column names would not have been next to each other. The Liquid programmers solved this problem by writing a computer program in PHP to write the computer program in SQL. The PHP program contains an array called “fieldMapping” that contains the source and target items, each pair on a single line, and then a final 7-line SQL INSERT.

185. The data import project demonstrates that (a) Liquid was using reasonably industry-standard methods, such as version control and bug-tracking, (b) Liquid was reasonably diligent and productive, finishing the project in both a fairly short calendar time and without billing an excessive number of hours, and (c) that Liquid programmers had sufficient pride in their work that they applied a certain degree of craftsmanship even to software that was not intended for long-term operation (the actual import was to be run only once).

XXVI. WHAT WAS THE QUALITY OF WORK DONE BY LIQUID? (V2.0)

186. Based on my review of the source code in the /fra directory, the work done by Liquid on v2.0 was all engineered to the WordPress Plugin standard. This is consistent with the PowerPoint presentation, “July PPT,” that was mailed on July 31, 2015 by Leigh Jamgochian to Justin Stroh and Tim Reeder.

187. As noted above, Liquid achieved the goal set forth in the July PPT of separating **HIPAA**-covered PHI from ordinary business data so as to comply with the Federal “Minimum Necessary Requirement.”

188. As noted above, in the “Did the v2.0 system work?” section, the structure of the v2.0 site enabled non-programmers with administrator privileges to change business-related information, such as policy rates. This is an example of a good design practice as it cuts down on ongoing programming costs (i.e., Liquid was setting up the v2.0 site so that PGH wouldn't have to pay Liquid for making minor alterations to the site; PGH would have been able to make these alterations itself).

189. As noted above, Liquid seems to have used reasonable efforts to avoid an explosion of duplicative source code and to have divided up code in a modular fashion so as to facilitate maintenance.

190. As noted above, the v2.0 was “responsive” and, during my own testing, displayed different layouts on a desktop web browser versus on a mobile phone.

191. As noted above, Liquid included error checking and error handling code in the v2.0 source.

192. As noted above, Liquid used stored procedures in MySQL to centralize business logic where appropriate.

193. As noted above, Liquid modularized the code in such a way that it would be possible to swap out the credit card processor for a new vendor without the risk of bugs popping up in unexpected places around the code base.

194. The Golbeck Report, page 50, attacks the v2.0 system for not coming packaged with documentation. But, the project was yet to be launched and, from the Liquid developers' perspective, the business requirements were shifting continuously. It was a reasonable decision to defer formal documentation until the v2.0 system was ready to launch. That the lack of documentation rendered the system confusing to Dr. Golbeck (page 50) or Judge's programmers is not a legitimate criticism. At the moment that the project was discontinued and the snapshot that we have was taken, the only people who were looking at v2.0 source

code were the Liquid developers who had created it. Had PGH told Liquid that they wanted to hand off the v2.0 to another team and that they wanted to pay Liquid to document the v2.0 system to facilitate adoption by another team of programmers, the lack of documentation might be significant. But those are not the facts of this case.

195. [The Golbeck Report highlights the Agile method of software development. "Is 'agile documentation' an oxymoron?" [25] points out the following:

For many agile development teams, "documentation" is considered a dirty word. After all, agile teams work under the premise that uncertainty is so common and change so rapid that spending a lot of time on upfront architecture or design will be largely incorrect or irrelevant further down the line.... As agile teams scale and fit into larger enterprise environments, the agilest must devise more mature, but equally agile, documentation strategies.

The Liquid programming team never did "scale and fit into larger enterprise environments" (hundreds or thousands of programmers). Therefore it is unreasonable to expect to find the documentation that would be characteristic of v2.0 being adopted by an S&P 500 IT department.]

196. That the v2.0 site did not launch publicly is not necessarily a reflection on the v2.0 code quality. Data migration, cutover, and launch of a replacement site for a working legacy site is a separate project from development of a candidate replacement software package. For an ecommerce business that operates 24/7, cutting over to a new server or new software can be a challenging engineering project by itself. When the legacy site is producing revenue every day, the business risks to a merchant are obvious and the benefits must be significant for a decision to be made to proceed. In my experience, it is not uncommon for an enterprise to invest substantial sums in a candidate replacement for a legacy system and, despite the fact that the replacement works reasonably well and offers at least some advantages, decide to stick with the legacy system and discard their investment in the candidate replacement.

197. That Judge did not want to adopt Liquid's v2.0 code is not surprising and is not necessarily a reflection on the v2.0 code quality. Programming teams often establish their own conventions for software projects and it is painful to incorporate another team's structures. Two equally good programmers will likely find it painful to swap code bases.

XXVII. WAS PGH IN ANY WAY RESPONSIBLE FOR THE V2.0 DISAPPOINTMENT?

198. Most software development projects are collaborations between business people, who understand what needs to be delivered in order to make money or serve users, and programmers, who understand how to write instructions that will change a computer's behavior.

199. It would be a rare project in which responsibility for success or failure could be laid entirely at the feet of either the programmers or the business people involved. Yet the Golbeck Report, with only conclusory statements about tens of thousands of lines of source code, states confidently that all blame can be assigned to the technical team.

200. I don't want to fall into the same trap of Monday Morning Quarterbacking that Dr. Golbeck did. Nor do I want to pretend that I have the knowledge of a fact witness regarding who said what and who did what over a multi-month period that involved hundreds of bug/issue tickets, phone calls, email exchanges, and face-to-face meetings.

201. However, I will point out that the schedule outlined by PGH essentially precluded textbook waterfall-style project scoping. PGH wanted changes to their software to be implemented, tested, and deployed within about a month after hiring Liquid. A traditional scope definition phase in the corporate IT world lasts roughly six months and results in no software at all, only a set of binders. (McKinsey highlights a success story with "One public-sector organization established strong project control by defining an initiative's scope in an initial six-month phase and making sure all stakeholders signed off on the plan." [26]) When

I managed ArsDigita we took business away from IBM Global Services because we were able to build and launch a service in less time than IBM would have taken to develop a scope, a plan, and a quote for software development. I haven't seen any evidence that PGH was willing to wait for or pay for a formal scoping process.

202. Additionally, consistent with the contracts in which PGH was purchasing hours and could use them as PGH managers thought best, PGH regularly emailed Liquid developers, such as Leigh Jamgochian, asked for specific tasks to be accomplished as soon as practical (see above). Common sense suggests that if Liquid programmers' time is being allocated by PGH via email then PGH is at least partly responsible for what gets accomplished and on what schedule. For instance, one email specifically states the following: “Due to a series of disruptive and time-consuming maintenance issues requested by First Risk Advisors [PGH], much of the budget was reallocated to other initiatives an tasks.” (Def. Discoverable Documents, Bates #000126).

203. PGH was imprecise in the specifications it delivered to Liquid's programmers. For example, in a January 9, 2016 email, David Opperman wrote “Ability on an insured by insured basis to override any rules built in: We would like this capability to to really be able to go in and approve an exception but then send an email to the insured that their request has been approved and have them log back into their accoutn and self serve the change so no changes or charges are made outside the system and the system and all changes stay in sync.” Compare to a spreadsheet sent by a UnitedHealthcare employee, attached to a June 17, 2015 email: “Academic Year: since PGH is not collecting enrollment for the 14/15 policy year, can this be removed as an option? Going forward, will policy years be removed from this drop-down when they are no longer available for purchase?” A programmer can be much more productive, and spend less time on rework, when working with a customer who provides precise guidance such as that delivered by UnitedHealthcare.

204. Based on the discoverable documents I reviewed, it seems reasonable to infer that PGH had been, at least at times, unresponsive to Liquid's requests for review and feedback. Why else would at least one agreement note that “Responses to requests for review and feedback will be, and must be, provided within 2 business days.” (Def. Discoverable Documents, Bates #000127).

205. A concrete example of this unresponsiveness is in an email exchange from late April 2015 (Def Discoverable Documents, Bates #000204-207). PGH has set forth an urgent schedule. Liquid has a programmer working. The project manager, Colleen George, has set up DoneDone and Dropbox for collaboration. She has created tickets in DoneDone for the items that PGH has already requested and noted that some tickets will be flagged “need more information.” More than 24 hours later, Shane Opperman responds that “It has been a crazy day at the office for me the pas[t] couple of days[,]” and promises to supply the rules no later than “early” the next day. Colleen George follows up the next day at 1:01 pm: “Shane - Just checking up on this. Can we get the rules today?” Apparently, the rules had not been supplied by 5:06 pm, because Ms. George writes, “I tried to call you this afternoon, but didn't get connected to you voicemail. It is critical that we get both the business rules for coverage and the authorize.net access ASAP, so that we can proceed with the prioritized list in a timely manner.” Thus, the project that was supposed to last roughly 20 working days slipped at least 2.5 days due to PGH's unresponsiveness.

206. Because PGH employees were not skilled at writing precise business rules, my understanding is that much of the requirements-gathering and suggesting for refinement of the evolving v2.0 system was done via telephonic and face-to-face meetings between Liquid and PGH employees. As there are, to my knowledge, no audio or video recordings of these meetings, it would not be possible for any expert to determine, to a reasonable degree of certainty, precisely who was at fault for the fact that v2.0, like so many other software projects since the inception of the digital computer, took longer to develop than originally hoped. It is not a breach of professional responsibility for a programmer to use telephone and in-person meetings to gather requirements and feedback, especially when a customer is unable to deliver clear written specifications and feedback. Nor is it a breach of professional responsibility for a solo programmer or small team of programmers to use the source code itself as the primary written record of these meetings.

207. Given that the contracts between PGH and Liquid were clearly for the sale of work by the hour, and the risk of those hours not being perfectly productive was assigned by those contracts to PGH, I don't understand the relevance of looking for blame. However, it is a big part of the Golbeck Report and therefore I included my thoughts on the subject.

XXVIII. OPINIONS

208. Computer programming and software development are not professions and there is no "professional standard" for software development process to which all programmers are expected to adhere. This is partly due to the fact that there is not a one-size-fits-all standard that would make sense for the variety of projects that programmers are hired to undertake.

209. Software development is inherently risky and software project delays, cost overruns, and outright failures are common.

210. Even when a new system has been developed and is working, switching over to that new system can be a challenging and risky engineering project in its own right, separate from the development of the new software.

211. As with Monday Morning Quarterbacking, with the benefit of hindsight it is easy to hypothesize regarding why an IT project did not proceed as hoped. However, there is no way to test such a hypothesis and the experts at making such hypotheses are not highly sought after by companies that actually develop software.

212. The risks inherent to software development projects are common knowledge among business managers.

213. It is common for software development contracts to allocate the risks of delays, cost overruns, and failure between the parties.

214. In a contract for hourly work, absent special explicit terms, the conventional understanding is that the customer bears the risks and that, if the customer is unhappy with the work being done, the customer's primary remedy is to terminate the contract and not pay for any unused hours. (Not dissimilar to what would happen with a programmer hired on a W-2 basis.)

215. The level of requirements, planning, architecture, and code documentation done by Liquid were not outside of industry norms considering the type of work being done, the business needs of PGH, and the rate at which PGH's business needs were evolving.

216. In those cases where it is straightforward, with hindsight, to evaluate the quality of Liquid's work, it is plain that Liquid performed at least to industry standards and, certainly, to a higher standard than PGH's previous software development contractor.

217. In those cases where it is straightforward, with hindsight, to evaluate the timeliness and time consumed by Liquid to perform a task, it is plain that Liquid completed tasks in a reasonable number of billed hours and in a reasonable amount of calendar time.

218. PGH presented Liquid's programmers with a moving target of requirements that evolved within the contract periods.

219. PGH periodically raised urgent issues, sometimes directly with the programmers rather than with the project manager. PGH periodically requested that Liquid's programmers stop work on the bigger picture items and address the urgent feature request or bug fix immediately. Liquid complied with these requests, which was reasonable in light of contract terms in which Liquid was supplying hours and PGH had discretion as to how those hours would be used.

220. The issues raised by the Golbeck Report can generally be summarized as "the work done jointly by Liquid and PGH did not go as well as hoped." Due to the contractual relationship between the parties, however, even if this were true and even if

all of the fault could be ascribed to Liquid, this is not relevant. Liquid was hired to work for a certain number of hours at a certain hourly rate.

XXIX. RESERVATION OF RIGHTS

221. My opinions are based upon the information that I have considered to date. I reserve the right to supplement or amend my opinions in response to additional opinions expressed by PGH's experts, and in light of any additional evidence, testimony, or other information that may be provided to me after the date of this report, including at trial. In addition, I expect that I may be asked to testify in rebuttal as to issues that may be raised in any supplemental reports of PGH's experts, or to issues that may be raised by PGH's fact witnesses and technical experts at trial. Additionally, while I have generally organized my opinions into sections with headings and sub-headings in this report, my opinions and the underlying information herein may be pertinent to numerous issues and I have not generally repeated that information. I therefore reserve the right to rely upon any information cited herein for any of my opinions, irrespective of the section headings.

Appendix not available.

Footnotes

1 https://philip.greenspun.com/ancient-history/professionalism-for-software-engineers

2 https://w3techs.com/technologies/overview/content management/all/3

3 https://codex.wordpress.org/Writing_a_Plugin

4 https://www.gao.gov/assets/110/104861.pdf

5 https://www.gao.gov/assets/110/107242.pdf

6 http://www.zdnet.com/article/worldwide-cost-of-it-failure-revisited-3-trillion/

7 https://www.getdonedone.com/a-feature-request-is-a-bug-is-a-task-is-a-to-do-is-a-dumb-detail/28

8 http://philip.greenspun.com/doc/ticket

9 http://calleam.com/WTPF/?page_id=1445

10 https://www.zurich.com/en/knowledge/articles/2015/09/cyber-costs-threaten-to-exceed-benefits

11 https://www.cs.usc.edu/masters/computer-security/

12 https://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-49.pdf

13 https://www.cvedetails.com/vulnerability-list/year-20 1 7/vulnerabilities.html

14 https://www.nytimes.com/2018/01/03/business/computer-flaws.html

15 available from https://github.com/A1Danial/cloc

16 https://wordpress.org/news/2013/01/wordpress-3-5-1/

17 https://wordpress.org/news/2015/02/wordpress-4-1-1/

18 https://www.hhs.gov/**hipaa**/for-professionals/privacy/guidance/minimum-necessary-requirement/index.html

19 https://en.wikipedia.org/wiki/2009_Sidekick_data_loss

20 http://www.datacenterknowledge.com/archives/2009/10/10/t-mobile-microsoft-lost-all-the-sidekick-backups

21 http://www.zdnet.com/article/key-questions-on-the-massive-rbs-natwest-it-failure/22

22 https://wordpress.org/news/2016/02/wordpress-4-4-2-security-and-maintenance-release/

23 https://developer.wordpress.org/plugins/53

24 http://www.telegraph.co.uk/travel/comment/Boeing-787-Dreamliner-a-timeline-of-problems/71

25 *https://www.ibm.com/developerworks/rational/agile/agile-documentation-* oxymoron/index.html

26 https ://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value

End of Document