UNITED STATES DISTRICT COURT
DISTRICT OF MASSACHUSETTS
EASTERN DIVISION

RED BEND LTD., and
RED BEND SOFTWARE INC.,

Plaintiffs,

v.

GOOGLE INC.,

Defendant.

Civil Action No. 09-cv-11813-DPW

DECLARATION OF MARTIN G. WALKER, PhD, IN SUPPORT OF GOOGLE'S OPPOSITION TO PLAINTIFFS MOTION FOR A PRELIMINARY INJUNCTION

I, Martin G. Walker, Ph.D., provide the following expert report:

I have been retained as an expert by the law firm of Bingham McCutchen LLP, counsel of record for Google, Inc. (hereafter "Google") in the above-captioned matter. This declaration is based on my personal knowledge and experience as well as my investigation in this matter and reflects my expert opinions on certain issues to which I may testify.

I. EDUCATION AND EXPERIENCE.

1. My CV (Exhibit A) contains an overview of my thirty years of experience in the high technology industry in Silicon Valley as well as a list of my recent publications. I received a BSEE from the Massachusetts Institute of Technology in 1973, MSEE from Stanford University in 1976, and a PhD in electrical engineering from Stanford University in 1979. During my career, I founded, served as President, a Member of the Board of Directors, and as Chief Scientist of Analog Design Tools. Later, I founded, served as a Board Member and Executive Vice President of Symmetry Design

Systems, Inc. Next I founded, served as CEO, (later Executive Vice President), and Member of the Board of Directors of Frequency Technology. I have also served as the CTO Knowledge Networks, an internet based consumer market research startup.

2. During my career, I have been responsible for design and development of several complex software programs. Additionally, to support my research activity at Stanford University, I wrote a special purpose program to aide in debugging a computer system that I designed. This program included a component called a "dis-assembler" that the examines executable programs and identifies the individual instructions that make up the program. I have gained direct industry experience understanding and managing software programs of the type and complexity of the software program at issue. As part of my management of software development, I also became familiar with the process of providing software updates to end-users and the process of creating and applying update patches in particular.

3. Recently, I have undertaken numerous consulting assignments that involved evaluating computer software code in the context of litigation. For instance, I analyzed code allegedly created by defendants in matters involving misappropriation of trade secrets to determine whether and to what extent the code incorporated the plaintiff's trade secrets. I have also analyzed software code to determine if the code practiced cited methods of patents. Thus I have a personal understanding of the normal standards in the industry regarding analysis of source code in litigation contexts. Additionally, I have testified in patent litigation contexts.

II. OTHER TESTIMONY.

4. A complete list of recent matters in which I have participated is attached as Exhibit B.

III. OVERVIEW OF ASSIGNMENT AND OPINIONS EXPRESSED.

5. I understand that Plaintiffs in the above matter ("Red Bend") have accused a component of Google Chrome Browser ("Courgette") of infringing certain claims of Red Bends US Patent No. 6,546,552 (the '552 patent). I was asked to investigate the

operation of Courgette, and determine whether Courgette did in fact infringe the properly construed claims of the '552 patent. As part of this investigation, I was also asked to form an opinion regarding the proper construction of certain terms used in the asserted claims of the '552 patent, and to consider questions relating to the validity of the '552 patent. Finally, I was asked to review the declaration of Stephen A Edwards in Support of Plaintiffs' Motion (referred to here in as the "Edwards Decl"), and to comment on his conclusions and methodology in light of all material I reviewed.

6. This declaration presents my findings at this time after having reviewed the issues as raised by Red Bend and its expert, Dr. Edwards, by Motion for Preliminary Injunction and the underlying evidence.

7. In summary, for the reasons discussed below, I have found that the '552 patent not infringed by Google's Courgette program and that the '552 patent is invalid. I also have found that Dr. Edwards used an unreliable methodology to reach his opinions.

8. I am generally familiar with the Claim Construction, Non-Infringement and Invalidity analysis processes through prior engagements I have had in patent infringement actions. I have been informed by Counsel in this action that claim construction involves interpreting claims as a matter of law from the perspective of one of ordinary skill in the art at the time of the invention, and that claim terms are given meaning by their usage in the claims themselves, the patent and the prosecution history. Additionally, other evidence (such as dictionaries) may also be used to determine the meaning of the words to a person of ordinary skill in the art at the time of the invention. I have also been informed that it is not necessary to rely on extrinsic evidence if the meaning of the claims is clear from the intrinsic evidence.

9. I am informed that the infringement analysis involves a comparison of each and every element of the claims, as properly construed, with an accused product. If all elements are found literally in the accused device then the claim is infringed literally. However, if one or more elements are not found in the accused device literally then there can be no literal infringement. I am also informed that infringement can be found under

the doctrine of equivalents if the accused device includes elements that while not literally the same are nonetheless only insubstantially different. I understand that the doctrine of equivalents cannot compensate for functionality that is wholly missing from an accused product or where the element that differs was the subject of argument to the patent office in support of patentability.

10. I am informed that for a finding of invalidity of a patent under 35 U.S.C. §102, which is termed invalidity by anticipation, each and every element of a claim, as properly construed, must be found either explicitly or inherently in a single prior art reference. I have been informed that a claimed invention is unpatentable under 35 U.S.C. §103 if the differences between the invention and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains. Obviousness, as I have been informed, is based on the scope and content of the prior art, the differences between the prior art and the claim, the level of ordinary skill in the art and secondary indicia of obviousness to the extent they exist.

IV. COMPENSATION AND MATERIALS REVIEWED.

11. I am being reimbursed for my time in this matter at the rate of $400 per hour. My compensation is not dependent upon the outcome of this case. In addition to any material expressly mentioned in this report, I considered the material identified in Exhibit C.

V. BACKGROUND.

12. I have reviewed the declaration of Stephen A Edwards in Support of Plaintiffs' Motion (referred to here in as the "Edwards Decl"). Included in the Edwards Decl is a section where he provides some background on the technology at issue. I generally agree with that section (¶ 9-14) and so as to help clarify the issues before the Court, I will not duplicate his discussion. However, I would like to elucidate a few additional details regarding memory references that are important to understand in the context of the operation of the accused device, Courgette.

13. Generally, machine instructions of the type described by Dr. Edwards may include several types of memory references. One type, explicitly discussed by Dr. Edwards is references to other executable instructions. Another type of instruction refers to data. For example a machine instruction may mean "Copy the contents of memory location XXX into memory location YYY." Thus this instruction would have two memory references, XXX and YYY. Further, the computer may use one of two methods to refer to memory locations (be they references to instructions or data): "absolute addresses," and "relative addresses." Absolute addresses are like parcel numbers at the assessor's office: they uniquely identify your property, or the address in memory. Another way to refer to your address is to say it is "two doors down from Carl's house." Such an address is called a relative address. In machine language, a sample reference might be: "take the value of register R, add 2, and use the result as an address." This is called a relative reference since the address in the instruction ("2") is interpreted relative to the value stored in register R. Courgette is adapted for use only on the instructions for the Intel processors, and such instructions use all four types of addresses discussed above.

THE '552 PATENT.

14. I have reviewed the '552 patent and its prosecution history. I have also reviewed the opening Red Bend Brief on Preliminary Injunction as well as the declaration and deposition testimony of Dr. Edwards, Red Bend's expert, on the '552 patent. Based on my review, I find the following aspects of the '552 patent most important for purposes of this case.

15. The '552 patent relates to a very specific process for creating and distributing software upgrades to old programs. When a new program is created updating an old program, the '552 patent teaches using this specific process for capturing the differences between the old and the new program in a compact difference result which can then be distributed as a patch.

16. I believe that the following aspect of the '552 patent is key and fundamental to the process described, and note that the inventors of the '552 patent, Red

Bend's counsel and Dr. Edwards also have flagged this approach to creating difference results within the '552 patent as important:

> The present invention is based on the observation that the relatively large size of the difference result stems from the alterations of reference in reference entries as a result of other newly inserted entries (and/or entries that were deleted).
>
> On the basis of this observation, the invention aims at generating a modified old program and a modified new program, wherein the difference in references in corresponding entries in said new and old programs as explained above, will be reflected as invariant entries in the modified old and new programs. *The net effect is that the invariant reference entries (between the modified old program and the modified new program), will not appear in the difference result, thereby reducing its size as compared to a conventional difference result obtained by using hitherto known techniques.*

This quote is taken from the '552 patent at column 3, lines 31 - 46. It is also found in its entirety at page 6 of Red Bend's opening Motion (emphasis above found in Red Bend's Motion). Dr. Edwards also refers to this language and in particular the problems of insert delete modifications making a patch file needlessly large at paragraph 15 and also in his deposition at 128:19-129:10. See Manning Decl Ex 26.

17. In essence (as set forth above), 'the 552 patent is based on the fact that differences between an old and a new executable program are largely caused by alterations of references (i.e. addresses) that are incidentally changed in reference entries (i.e., instructions that refer to addresses) because of newly inserted (or deleted) instructions forming the transition from the old to the new program. The patent solves this problem by replacing corresponding addresses in the new and old program that changed only due to address shifting from insert/delete modifications with "invariant references." The replacement is made in a modified old and a modified new program. Thereafter, the modified old and new programs are run through a difference generator and the address changes due only to insert/delete are "neutralized" because of the

invariant references assigned and disappear from the difference result. See '552 patent at 1:59-2:9 and 3:27-3:46 and 10:3-15 and 14:65-15:8. Dr. Edwards includes a very similar summary in paragraphs 15-17 of his declaration.

18. In addition to the patent, the prosecution history also emphasizes a process of identifying insert/delete modifications and assigning invariant references to the affected addresses in the old and new program so that they disappear from the difference result:

> A major problem arises when applying these [prior art diff] methods to executable program files . . . . The problem arises from the fact that executable programs are generated from sources and in that process many references are inserted into these executable files. These references do not refer symbolically to other location of the program, as may be the case in source files, but they refer to addresses - sequential locations in the program file. . . .
>
> Consider, for example, an extreme example where a change in the first source of line [*sic.*] may lead to actual change of some first executable file, in which few bytes were added but also all references must change since they refer to locations that now have been moved farther for the amount of bytes added at the beginning. To simply reflect all the changed references when computing a difference, one must include them all. In accordance with the present application such a need is reduced or eliminated, and what is required, is just to send the first few modified bytes . . . . The modification is effected in such as way that the references become "invariant" . . . .

Office Action at 7. RedBend 000150 (Manning Decl. Ex. 2)

19. The patent also is clear that it applies to forming difference results on executable programs. Both Dr. Edwards and I consider executable programs to include "object code," and to involve machine language instructions, as opposed to a higher level language such as source code. See Dr. Edwards' declaration at paragraph 11-12.

20. The prosecution history further clarifies that in extracting the difference between two versions of executable files as defined in independent Claim 1, "there is no source involved, and neither statements, not any textual or other symbolic representation

of the program even exist." See Response to Office Action mailed October 2, at 8, 17-20 RedBend 000151 (Manning Decl. Ex. 2). The applicant makes the same argument relative to all claims of the patent, regardless of whether the claim uses the terms "executable program" or "data table" to describe on what the difference is being generated. See 8, 10-11 and the following quote on page 11 of the Response to the October 2 office action:

> Claims 35-68 are basically similar to claims 1 to 34, respectively, except for the fact that they recite data table instead of executable program. Data table is discussed on page 4, line 9 of the application **and do not embrace source code** as in Okuzumi. It is accordingly submitted that ... 42-44 ... 55-57 are not anticipated by Okozumi for the reasons discussed in detail above with reference to claims 1 to 3, 8-10 . . . .

21. I have worked with counsel on a claim construction chart that embodies definitions that I am comfortable with based on the above principles. The chart lays out Google's constructions next to Dr. Edwards, where applicable, and is attached as Exhibit D.

22. The specification of the '552 Patent also emphasizes the importance of replacing "substantially" reference entry in the old and new programs. For instance, in the section entitled "SUMMARY OF THE INVENTION," the description of the invention includes "scanning the old program and for substantially each reference entry perform" performing certain steps (see '552 Patent at 3:53-54). It is significant that this same or similar language is repeated more than 20 times in the specification. Thus addressing "substantially all" of the reference entries should be thought of as an important part of the invention described in the '552 Patent.

VI. GOOGLE'S COURGETTE PROGRAM.

23. After studying the '552 Patent, and coming to an understanding of the claims, I next began to analyze the operation of the accused program, Courgette. I found that the operation of Courgette was relatively complex, with different parts of the program interacting in subtle ways. Thus I realized that it was important to follow a

careful and thorough methodology to determine the operation. After I understood the operation of Courgette, I compared the operation of Courgette to the asserted claims as properly construed. This section presents the methodology necessary to determine the operation of Courgette, a detailed discussion of its operation, and finally a comparison of the operation of Courgette to the asserted claims of the '552 Patent.

A. Methodology.

24. The detailed operation of computer software programs is notoriously difficult to determine by merely studying the source code. Particularly in the case of object-oriented code such as Courgette, subtle errors of analysis can lead to profound errors in conclusions regarding the operation of the code. While there may be others, there are at least three sources of uncertainty associated with analyzing source code by looking only at the source code: a) Mistaken identity; b) Difficulty in tracking cause and effect across a complex piece of software; and c) difficulty appreciating the structure and content of data objects created by software.

25. Mistaken identity can occur easily in the case of object-oriented code when multiple methods have the same name. In that case, an incorrect assumption can occur regarding which method is actually used by the program during its execution. In other words, some source that is part of a program may never actually be used, while an overlooked portion is in fact used and called by the same or similar name.

26. Difficulty in tracking cause and effect across a complex piece of software arises from a problem of trying to mentally track a very long sequence of operations with a rigorous and accurate understanding of those operations. This is difficult to do for a simple piece of software, for example presented to a college professor for grading. It is nearly impossible for any commercially useful piece of software.

27. Difficulty appreciating the structure and content of data objects created by software is caused again by complexity and the fact that the object oriented programming styles can blur the definition of data structures across multiple files. It is therefore

difficult to appreciate how the data and structures fit together without examining how the program actually creates and uses the data structures themselves and data within those structures.

28. Because analyzing source code alone is problematic, if the complete source code is available, it is standard practice in the industry to compile and run the alleged infringing source code in order to truly understand the operation. The operation can thus be studied through several means including (i) use of a "debugger" (which allows the one to follow each step in the operation of the program and the source code corresponding to each step in the operation of a running program as well as the content and organization of data structures created by the running program), (ii) turning on debugging statements included by the program's authors (that are disabled during normal operation), and/or (iii) adding debugging statements to further document the program's operation. Failure to follow such standard industry practice may lead to significant errors in analysis and result in an unknown and unknowable error rate in opinions offered regarding the operation of the alleged infringing source code.

29. In the instant case, all source code for Courgette is available and has been posted by Google on its website as open source software. Dr. Edwards, Red Bend's technical expert, also notes "A software developer, or anyone with a C++ compiler and little experience, would be able to use the code posted by Google." (See Edwards Decl at It is part of Google's Chrome browser software open source distribution and is found at http://build.chromium.org/buildbot/archives/chromium_tarball.html, The Courgette source code itself is written in C++ programming language which is an industry standard object oriented programming language, and is made available to permit downloading and compiling it into a stand alone executable program.

30. In order to analyze Courgette, I downloaded the source code and followed the directions posted by Google on the dev.chromium.org website to compile the program and operate it using industry standard software development tools, including a debugger. My analysis and findings are based on studying the source code, using a debugger to run

and analyze the source and compiled Courgette code as it runs and reviewing the content of log files created by Courgette. I noted that the Courgette distribution included two programs to be used to confirm proper operation of Courgette and I used those programs as inputs to Courgette for purposes of analysis. I also created my own test programs to further understand and illustrate how Courgette creates patch files used to update programs. Additionally, I added additional debugging statements to the Courgette program so that it would further document data structures as Courgette created its patch files.

31. I also used external documents, and the two documents identified by Dr. Edwards in paragraph 7 of his declaration as background information to help understand where to start my analysis. Based on my experience, I knew that observation of the actual operation of the software in a proper software development environment as described above was necessary to conduct an accurate analysis.

B. Findings

32. I found, as a result of my analysis, that Courgette operates in a very different manner than the '552 patent describes. While Courgette is a program that generates a patch file for updating an old program and can do so on executable programs, the internal operation is completely different and the algorithmic approach is completely different from what is described in the '552 patent.

33. In this regard, the '552 patent is based on the premise that differences (particularly in addresses) between an old and new program due to insertion and/or deletion of instructions can be completely eliminated from a patch file. See discussion above in section VI. The patent requires identification and processing of "substantially all references," relating to insertion or deletion of instructions and states that those references are "reflected as invariant references in the corresponding entries" of the new and old programs.

34. By contrast, Courgette does not identify insertion or deletion of instructions and rather preserves any unique addresses that it does find in the old and the new programs in symbolic data structures and then compresses differences efficiently. Moreover, Courgette does not identify every reference entry or even substantially every reference entry within the old or the new program. In particular, Courgette recognizes all absolute addresses but only certain relative addresses that are used as references. Based on my analysis, Courgette only recognizes a minority of relative addresses associated with reference entries (i.e. instructions). For instance, Courgette does not recognize any instruction having a relative address that refers to data and only three[1] instructions among dozens that use relative addresses to refer to other instructions. These unrecognized instructions are commonly found in executable programs such as the ones that Dr. Edwards and I analyzed. An excerpt from an executable program considered by Dr. Edwards is attached in Exhibit E, and Dr. Edward's testimony regarding the references is included in the Manning Decl., Ex. 26 at 217:9 to 220:19. Dr. Edwards testified that, although he couldn't say what percentage of reference entries were ignored by Courgette, he acknowledged (and I agree) that not all reference entries are analyzed.

35. My findings at this stage address Courgette's generation of a patch, but not the process of applying the patch because none of the asserted claims require applying the patch. I discuss its operation below in the order that program operates.

36. Courgette begins by inputting an old and new program and then parses the programs to create unique data structures apparently developed by Google to represent each program. The data structures themselves are obviously distinct from executable programs. For the purpose of illustrating the operation of Courgette, I created a simple program. Then I made a few changes to the source code. These two versions of the code are attached as Exhibit F. The left column is the "old" program. The right column is the "new" program. The symbols in the middle identify insertions (">"), deletions ("<"), and

[1] These instructions include "JMP," "CALL," and "Jcc." Jcc (conditional jump) includes approximately 30 sub instructions of which Courgette recognizes most but not all. See the method ParseRel32RelocsFromSection in the file disassembler.cc.

changes ("|").

37. Both of these were compiled into executable code (the "old" program, and the "new" program. An excerpt of this code is attached as Exhibit G.[2] Courgette disassembles these executable programs and creates symbolic data structures from the information in the executable programs. Courgette's initial data structures for each of the old and the new program include the following parts:

(i) A designation specifying the function of each byte in a symbolic function list. The designations include "DEF" for bytes Courgette does not analyze further and which Courgette incorporates directly into the function list; "ABS32" for bytes which are 32 bit absolute addresses; "REL32" for bytes which are 32 bit relative addresses; and two additional codes for special addresses. An excerpt of this list is attached as Exhibit H. Bytes that are labeled DEF are not analyzed further by Courgette. Thus if Courgette does not recognize a reference entry, the bytes associated with that reference entry are coded as DEF bytes. If the reference entry changes in the new program due to insert or delete modifications, these reference entries will not be further processed, and will appear in the difference result, as explained below.

(ii) A list of 32-bit absolute addresses and corresponding index values in a table known as a hash array. In short, the table data may be retrieved by value as well as by using a pointer into the table. Exhibit I is a table showing the listing for the reference entries that Courgette recognized. This table is related to the function list above through the numbers in the first column. For instance, the first ABS32 reference in Exhibit J has a value of 0x00D24890. This would refer to the entry in Exhibit I with the matching number (in this case, the last highlighted line). Thus this ABS32 reference entry has an address of 1719c and an index value of 122.

[2] The entire executable listing for even this simple program is over 13,000 lines long, much too detailed to be attached in its entirety.

(iii) A list of 32 bit relative addresses and corresponding index values in a table known as a hash array similarly organized as with the absolute addresses. In short, as above, this table data may be retrieved by value as well as by using a pointer into the table.

38. A simple comparison between Exhibit G (the executable program) and Exhibits H and I show significant differences. On the one hand the executable program is a sequence of bytes in machine language ready for execution by a computer. On the other hand, the data structures used in Courgette reflect a parsing and dissection of each of the old and the new program into symbolic data structures no longer resembling a program.

39. Once the above data structures are created through the disassembly process, an adjustment step is performed on the data structure corresponding to the new program. This step consists of modifying the values of the indexes in the ABS32 and REL32 tables associated with the new program so as to preserve all of addresses in the new program and make efficient reassignments of indices to each address. The goal is to make the list of indexes in the new program resemble more closely the list of indexes in the old program. Exhibit J shows the data structures of the adjusted new program. As can be seen, all addresses are still present in the adjusted program; none have been replaced.

40. After the data structure for the new program is adjusted, Courgette performs a further step of encoding the two data structures into eight streams, one set corresponding to the old program and one set corresponding to the new program. An exemplar is shown in Exhibit K and the eight streams are:

(i) A control stream consisting of the following commands: copy, copy1, abs32, rel32 and origin. Copy means to copy n bytes from the byte stream into the output (n is determined by the next entry in the size stream). Copy1 means to copy 1 byte from the byte stream into the output without reference to the size stream. Abs32 and Rel32 mean to use the next address in the address stream corresponding to next sequential index. The origin means to copy the next address of the special address stream.

(ii) A size stream which specifies the number of bytes in the byte stream corresponding to each Copy command.

(iii) A byte stream which corresponds to DEF bytes that Courgette treats in bulk for purposes of compression. As mentioned above, this stream includes reference addresses that Courgette did not recognize as such.

(iv) A stream of all 32bit absolute addresses used in the old and the new program.

(v) A stream of all 32bit relative addresses used in the old and the new program.

(vi) A stream of index values corresponding to 32 bit absolute addresses as they occur in the old or new program.

(vii) A stream of index values corresponding to 32 bit relative addresses as they occur in the old or new program.

(viii) An origin stream corresponding to a list of special addresses used by the old and new programs.

41.

42. At all times, Courgette preserves all addresses it finds in the new and the old programs by memorializing them in the data structures and the streams corresponding to the old and the new programs. All addresses Courgette finds are also inputs to the difference program used to generate the patch. Courgette preserves and uses all addresses, without analyzing addresses to determine whether they were caused by insert and/or delete modifications. In fact, the Courgette approach does not identify insert and delete modifications at all when processing the old and the new programs.

VII. COURGETTE DOES NOT INFRINGE THE '552 PATENT.

43. Based on my review and findings identified above from analyzing and running Courgette, and looking at the additional documents identified by Dr. Edwards, I can summarize my non-infringement findings by observing that each asserted claim requires substantially all of the references that change due to insert/delete modifications

between the old and the new program to be removed and replaced by invariant references prior to generating the difference result. The invariant references are reflected in the modified programs in executable form in place of the reference addresses which disappear from the difference result. By way of contrast, with Courgette, all addresses are preserved and included in the difference calculation, regardless of whether they changed due to insert/delete modifications or otherwise. I have also found that Courgette does not recognize close to all or substantially all references as claimed, and does not generate modified programs preserving their executable format prior to creating a compact difference result. Courgette preserves and reflects all unique addresses it recognizes and all unique addresses it doesn't recognize in the difference result. The drawing attached as Exhibit L compares salient features of the method claimed in the '552 Patent with the operation of Courgette, as I have described above.

44. Consider a case when a few lines are inserted at the beginning of a large program causing thousands of reference addresses to change in the new program. With the claimed method of the '552 patent, only the added lines will be included in the compact difference result because the reference address changes will not appear in the difference result. By contrast with Courgette, the insertion of the same few lines at the beginning of the same program will cause a difference result that includes not only the newly added lines but also all of the thousands of unique reference addresses that changed that Courgette recognized and all of the reference addresses that Courgette did not recognize in the compact difference result.

45. The following paragraphs elaborate on these non-infringement findings further based on the analysis that I have done. In this regard, the following claim elements (b)(i) are not present in Courgette:

> "substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/insert modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs"

and

> "substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/insert modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables"

46. For this element, which bears these two formulations among the asserted independent claims 8, 21, 42, and 55, Courgette does not meet the element for several independent reasons.

47. First, the above claim element requires looking for any references that are different due to insert/delete modifications. Courgette completely omits this requirement and operates differently. Specifically, Courgette creates symbolic data structures and symbolic encoded streams of data and processes them without any reference to whether addresses are changing due to insert delete modification. There is no suggestion in any evidence that I have looked at, including the Courgette code as it runs and appears, or any ancillary document relied on by Dr. Edwards, including

48. Second, Courgette cannot recognize "substantially all references" as required by this element. Rather, as discussed above, Courgette is only equipped to process addresses associated with certain instructions (or reference entries). It ignores relative addresses associated with the majority of x86 instructions.

49. Third, the claim language above requires "reflecting substantially all

references ... as invariant references" in the modified old and new (programs or data tables). The patent teaches doing this so that these references, which change only due to insert delete modifications, appear to be "invariant" and are therefore not reflected in the difference result. By contrast, I have found with Courgette, by analyzing the code and observing the output that it generates, that all unique address references recognized by Courgette are preserved in symbolic data structures by Courgette (in address streams sent to the difference generator) and are included in the difference result directly, even when changed due to insert/delete modifications. All reference addresses not recognized by Courgette are also included in the symbolic data structures (in the byte stream sent to the difference generator) and also appear in the difference result, even when different due to insert/delete modifications.

50. Dr. Edwards relies on Courgette's use of index values as meeting the "reflecting as invariant reference" language within the claims. But Courgette uses index values to represent unique reference addresses in addition to preserving all unique reference addresses themselves that it recognizes. The indices, however, are not a substitute for the addresses. The indices do not conceal the unique reference addresses from the difference generator or the difference result. Thus, the index values cannot and do not act as invariant references as claimed.

51. I do not believe that the Courgette code itself or the ancillary descriptions of the Courgette program relied on by Dr. Edwards could lead one to conclude that element (b)(i) of the asserted claims is met.

52. The asserted '552 patent claims also require generating a "modified old program" and "modified new program", or "modified old data table" and "modified new data table" and using those modified programs (or data tables) to generate a compact difference result. The old program and data table and are required to be executable and not symbolic, as are their modified forms. This is clear in the prosecution history where the applicants argue that symbolic representations are not used as discussed in paragraph 20 above and where the applicant argued that a modified program does not include the

creation of an index table separate from the program to distinguish the Miller patent identified by the examiner. See Comments on Statement of Reasons for Allowance, at 1,2. RedBend0000173-174 (Manning Decl. Ex. 2).

53. The '552 patent also requires executable programs and modified versions of those programs still in executable form. (See paragraph 20 above). Unlike the '552 patent, Courgette generates ancillary data tables and streams that are no longer in executable form. They symbolically reflect information from the old and new programs, but do not resemble in any way an executable program. The symbolic data structures are first derived by disassembling the old and the new programs and then are further abstracted into symbolic streams that represent separately collect relevant information in streams, such as addresses, indexes bytes, size data and other control information in a format. This symbolic information used by the difference generator cannot be considered in executable format.

54. I have attached as Exhibit M claim charts further summarizing my analysis of the asserted claims in light of the above principles. For purposes of my analysis, I treat asserted independent claims 8, 21, 42 and 55 the same as claim 42. Both Red Bend and Edwards do the same, and I agree with this approach. For all of reasons stated above and in the attached claim charts, I believe that Courgette is very different from the '552 patent and does not infringe any of the asserted independent claims, or claims 9, 10, 23, 24, 43, 44, 56, and 57 depending on them.

55. Finally, I note that in light of the above discussion, there is required functionality of the claims, including "substantially each reference," and/or "due to insert/delete modifications" that is missing from Courgette. In addition, based on comments made to the patent office, the patentee specifically excluded symbolic representations of the program from the scope of the patent. This is inconsistent with Courgette, which does use symbolic code as discussed above. Therefore, it is my opinion that Courgette does not infringe the '552 Patent under the doctrine of equivalents.

VIII. THE EDWARDS DECLARATION.

A. Unreliable Methodology

56.

For instance, his claim chart states the method "AssignOne" of the file adjustment_method.cc performs certain key steps of the method. However, this method is never invoked during the operation of the accused device.

Further, Dr. Edwards opines that the "the difference calculation operates on the modified old program (asm_old) and the modified new program (asm_new_adjusted)." (Edwards Decl Exh C.) Had Dr. Edwards studied the operation of Courgette, he would have realized that the difference calculation actually operates on the "encoded" program data structures, not the "assembly" program data structures as he states in his claim chart. Further, he would have realized, contrary to what is in his claim chart, that there are no objects termed "asm_old" or "asm_new_adjusted" used by Courgette. Similarly, he would have realized that there is no actual program "m" used by Courgette, nor is there is any actual program "p" used by Courgette as he states in his claim chart (Exhibit C) to the Edwards Decl.

57.

It is industry practice that such sources can provide an introduction that can help an expert in gain understanding of the operation of the source code, but it is also standard industry practice that the ultimate basis for an infringement opinion should be the source code itself, not someone else's summary of its operation.

58. By his flawed methodology it is clear that Dr. Edwards failed to meet industry standards for a reliable expert opinion. As a result, his opinions contain numerous and expected factual errors and erroneous conclusions about what code is actually used, what data structures created by the running code actually contain and how the code actually operates.

B. Mistakes and Inconsistencies

59. In his opinion, Dr. Edwards identifies a series of steps performed by Courgette that allegedly constitute meeting the limitations of the asserted claims. For reasons discussed in detail above, it is my opinion that Dr. Edwards is wrong in his analysis. Even more to the point, Courgette does not work in the manner in which he briefly states. In particular, Dr. Edwards states in his claim chart that "the user" is instructed to operate Courgette. For example he states that "the user is instructed to disassemble and adjust using the Courgette executable by invoking it using a different switch -disadj." In fact, using Courgette in this manner would not cause Courgette to perform the elements of the claimed method as he has identified in his opinion. (In this example, Courgette would not produce compact differences, an essential element of the claimed method.) That Dr. Edwards makes this same type of error with respect to the "-dis" and "-disadj" switch 4 times in his declaration simply underlines the unreliability of the methodology employed by Dr. Edwards in his analysis of Courgette.

60. Dr. Edwards makes the following errors in his analysis of the claims of the

'552 Patent:

-

-

- He treated the claim terms as having different meanings for infringement and validity and/or representations made to the patent office to get the patent to issue.

C. Post-Declaration Activity is Not Helpful or Specific

61. Just prior to his deposition, Dr. Edwards produced evidence of additional investigations that he undertook subsequent to signing his declaration. I have reviewed these notes and I listened to deposition testimony and have reviewed the transcript of his deposition testimony regarding this post-declaration activity.

62.

63. In particular, Dr. Edwards' infringement analysis is incomplete in at least the following ways. He has not identified anything in Courgette that looks for and processes references affected by insert/delete modifications. He also appears to agree that the asserted claims require processing all of the references (at least for purposes of invalidity) yet he acknowledges that Courgette does not handle many instructions with

relative addresses. (See for example Manning Decl. Ex. 26 178:14-183:3, 217:9-218:22, and 242:8-243:4.) He also recognizes for invalidity purposes that unless the old and new programs are executable they are not within the scope of the claims, but yet does not address the fact that Courgette dissembles executable files into symbolic data structures before creating difference results on those structures.

64. Dr. Edwards also realizes that his declaration is wrong with respect to alleged infringement of element (b)(i).

However, as of the time of this writing, Dr. Edwards has not supplemented his report to correct these errors. Further, based on my review of his deposition testimony, I still do not understand what if any basis Dr. Edwards has for believing that Courgette infringes element (b)(i) of Claim 42. Dr. Edward's deposition testimony was neither helpful nor specific on this point.

IX. INVALIDITY

65. I am familiar with technology for patching software that predated the '552 patent, including that which is discussed in the background section of the '552 patent. Based on my understanding of the '552 patent, the inventors were able to get the patent allowed over the prior art cited in the file history because the inventors argued that the invention was: (i) limited to creating differences on executable code and not source code, (ii) limited to creating a difference result on modified old and new executable programs, rather than ancillary tables or information; and (iii) limited to using invariant references to make address differences disappear from the difference result. All of these aspects appear to me to be included in one prior art reference that I have reviewed, namely U.S. Patent No. 5,481,713 by Wetmore, et al. (Attached to the Manning Decl. as Ex. 30, here after the "Wetmore '713 Patent.") In my opinion the Wetmore patent meets all of the elements of the asserted claims, including the above aspects of the '552 patent.

66. Like the '552 patent, the Wetmore patent discloses patching or updating

an executable program that would normally reside in read-only memory ("ROM"). While Wetmore refers to the executable code as "object code," both Dr. Edwards and I treat object code and executable code as the same -- both are programs at the lowest level comprised of machine language instructions. See Edwards Decl at paragraph 11, 12.

67. To allow patching, according to Wetmore the code is first modified by "vectorizing" it to replace references with labels that are jumps to modifiable code residing in random access memory ("RAM"). Program patches or updates are then created by generating the difference results between the "vectorized" versions of the old and new executable programs. The difference result is provided to the user's computer to update or patch the executable program, and the user's computer generates the updated executable program based upon the difference results and executable program already present at the user's computer. (Wetmore '713 Patent 10:62 - 11:12)

68. The Wetmore prior art patent discloses all of the steps required by the asserted independent claims of the '552 patent, namely: a) generating a modified old program (a vectorized program that replaces references with invariant values); b) generating a modified new program (a vectorized program where the invariant references are the replaced, vectorized addresses); and c) generating a difference result between the modified old and modified new programs (generating a difference result between the vectorized programs) to generate a difference result.[3] Wetmore thus in my opinion anticipates all of the asserted claims, rendering it invalid under 35 U.S.C. § 102.

69.

70. In my opinion, Wetmore teaches executable code because object code is executable code that includes machine language instructions. Dr. Edwards apparently

[3] Wetmore '713 Patent (a) col.4 ll.38-39; col.5 ll.1-3, 10-17, 18-31; FIG. 4; (b) col.4 ll.38-39; col.5 ll.1-3, 10-17; col.10 ll.6-14, col.11 ll.2-12; Fig 4; (c) col.5 ll.18-56; col.6 l.45 – col.8 l.52; FIGS. 3-5

agreed with this view as well in paragraphs 11 and 12 of his declaration.

71. Additionally, while Wetmore may not explicitly teach vectorizing all references, it does not exclude doing so and there is no reason that someone following the teaching of Wetmore would not apply Wetmore to vectorize all references in a program being patched. I believe this would be obvious to try.

72. I have reviewed the Reexamination papers filed by Google requesting Reexamination of the '552 patent to the extent that they relate to Wetmore and agree with those portions of the Reexamination papers. I have not reviewed other references or arguments contained within the Reexamination papers. I also agree with the claim charts comparing the Wetmore patent to the '552 patent claims and have attached copies of those charts here as Exhibit O.

73. The invalidity charts attached demonstrate that the asserted independent claims of the '552 Patent, including claims 8, 21, 42, and 55 are invalid as anticipated and obvious. I further submit that dependent claims 9, 10, 23, 24, 43, 55, and 57 are also invalid as either anticipated by Wetmore inherently or obvious over Wetmore. These dependent claims merely add that updates can occur over a communications network and specifically the Internet. In my opinion, communication networks including the Internet is generally how most patches were applied at the time of the invention.

X. APPLICABILITY OF COURGETTE TO OTHER OPERATING ENVIRONMENTS

74. It is undisputed that the Courgette code as it exists today creates compact difference results only for executable programs written for the Intel x86 instruction set and stored in the Windows PE executable file format. In his declaration at paragraph 24, Dr. Edwards states that the Courgette software is "easily adaptable to processing executable files for other platforms, such as those found in mobile devices." Dr. Edwards is wrong. He has ignored or is unaware of at least the following factors that tie Courgette to x86 instructions and the PE file format:

- Courgette relies on an optional section of the PE file format to recognize

all absolute addresses;

- Courgette relies on recognizing 3 of the x86 instructions by their signature bits to detect the few types of relative references that it processes;
- Courgette relies on the fact that the Windows C++ compiler used for the Chrome browser project produces PE files that make use of the optional section for specifying absolute addresses; and
- In short, Courgette relies on short cuts that are specific to Windows PE and x86 to simplify the very complex process of parsing executable code and trying to match reference addresses and corresponding instructions. It by no means clear that other platforms, particularly mobile platforms, include the same or even similar structures -- and probably they do not.

75. Since all of the above dependencies in Courgette limit Courgette to producing compact differences of executables that use the x86 instruction set in the Windows PE file format, it is my opinion that adapting the Courgette algorithms to other platforms would be far from easy, and would in fact require substantial engineering resources if it could be accomplished at all. I further understand that Google has not attempted to use Courgette beyond the Chrome project and I am not surprised given my findings in this section. (See Manning Decl. Ex. 9, GOOG00039449)

I declare under penalty of perjury that the foregoing is true and correct.

Executed on: March 1, 2010 in Palo Alto, CA.

MARTIN G. WALKER, PH.D.