

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MICROSOFT CORPORATION,
Petitioner,

v.

X1 Discovery, Inc.,
Patent Owner.

IPR2025-00254
U.S. Patent No. 10,552,490

SUPPLEMENTAL DECLARATION OF LOREN G. TERVEEN, PH.D.

MICROSOFT EXHIBIT 1100
Microsoft v. X1 Discovery
IPR2025-00254

TABLE OF CONTENTS

	Page
I. INTRODUCTION AND ENGAGEMENT	4
II. BACKGROUND AND QUALIFICATIONS	4
III. MATERIALS CONSIDERED AND INFORMATION RELIED UPON REGARDING '490 PATENT	4
IV. UNDERSTANDING OF PATENT LAW	5
V. LEVEL OF ORDINARY SKILL IN THE ART	5
VI. A POSITA WOULD HAVE BEEN MOTIVATED WITH A REASONABLE EXPECTATION OF SUCCESS TO COMBINE THE PRIOR ART REFERENCES TO INCREMENTALLY SEARCH ACROSS FILES	6
A. True's Teaching To Search Text That Is 100% Accurate Would Further Motivate A POSITA With A Reasonable Expectation Of Success To Combine True's Single-Field, Multi-String Teachings With Entourage	18
B. A POSITA Would Have Reasonably Expected Success In Implementing The Ground 1 and Ground 2 Prior Art Combinations Using A Suitable Programming Language	20
1. A POSITA Would Have Reasonably Expected Success In Implementing A User Interface That Supports Incremental Searching	21
2. A POSITA Would Have Reasonably Expected Success In Implementing The Back-End Logic That Executes The Incremental Searching	23
C. The Patent's Lack Of Implementation Details Further Confirms The Reasonable Expectation Of Success	26

VII.	A POSITA WOULD HAVE BEEN MOTIVATED WITH A REASONABLE EXPECTATION OF SUCCESS TO COMBINE THE SIDE-BY-SIDE DISCLOSURES IN THE LOTUS NOTES REFERENCES (GROUND 1) AND THE ENTOURAGE REFERENCES (GROUND 2) ..	27
A.	A POSITA Would Have Found It Obvious To Combine The Functionality From Lotus Notes' Search Tools, And The Query Builder Tool Alone Teaches The Disputed Claim Elements...	28
B.	A POSITA Would Have Found It Obvious To Combine The Functionality From The Entourage Search Tools	30
VIII.	THE PRIOR ART TEACHES EVERY ELEMENT OF THE INDEPENDENT CLAIMS	31
A.	The Ground 1 References Teach Element [1.5].....	31
IX.	THE PRIOR ART RENDERS THE DEPENDENT CLAIMS OBVIOUS..	32

I, Loren G. Terveen, Ph.D., do hereby declare as follows:

I. INTRODUCTION AND ENGAGEMENT

1. I have been retained as an independent expert on behalf of Microsoft Corporation in connection with the above-captioned Petition for *Inter Partes* Review (“IPR”) to provide my analyses and opinions on certain technical issues related to U.S. Patent No. 10,552,490 (hereinafter “the ’490 patent”).

2. This Reply Declaration responds to certain points raised in the Patent Owner Response (Paper 21) and the Declarations of Martin Haeberli (EX2001, EX2020).

3. I filed a previous declaration (the “Initial Declaration,” EX1003), dated December 19, 2024, in this matter.

4. I am being compensated at my usual and customary rate for my time with this IPR. My compensation is not affected by the outcome of this IPR.

II. BACKGROUND AND QUALIFICATIONS

5. I incorporate by reference the description of my background and qualifications presented in my Initial Declaration and my curriculum vitae appended thereto as Appendix A. *See* EX1003, ¶¶4-11, Appendix A.

III. MATERIALS CONSIDERED AND INFORMATION RELIED UPON REGARDING ’490 PATENT

6. In preparing this declaration and arriving at my opinions, I have reviewed the materials cited in my Initial Declaration, the additional exhibits listed

below, and the POR and Mr. Haeberli's declarations and the documents and exhibit cited therein.

LIST OF ADDITIONAL EXHIBITS

No.	Description
1095	<i>Yahoo Raises Stakes in E-mail War with Google</i> , NBC News (June 15, 2004)
1096	<i>Yahoo Mail Review: Yahoo Mail</i> , CNET (July 15, 2002)
1097	Charles Petzold, <i>Programming Windows</i> , Microsoft Press (excerpts) (5 th ed.) (1998)
1098	Deposition Transcript of Martin Haeberli, <i>Microsoft Corp. v. XI Discovery, Inc.</i> , IPR2025-00255 (Jan. 8, 2026)
1099	John Cole et al., <i>The Implementation of Electronic Services: Planned or Organic Growth?</i> , Informatics in Primary Care, Vol. 13, No. 3, 187-193 (2005)
1101	<i>Microsoft Press Computer Dictionary</i> , Microsoft Press (excerpts) (1 st ed.) (1991)

IV. UNDERSTANDING OF PATENT LAW

7. I incorporate by reference the description of my understanding of patent law presented in my Initial Declaration. *See* EX1003, ¶¶13-25.

V. LEVEL OF ORDINARY SKILL IN THE ART

8. I have reviewed the level of ordinary skill in the art proposed by Mr. Haeberli in his declaration. *See* EX2001, ¶¶26-31. For the reasons set forth in my Initial Declaration, I was a person of ordinary skill in the art in 2002 according to the level of ordinary skill proposed by Mr. Haeberli.

9. In this Declaration I apply the definition of a POSITA set forth in my Initial Declaration. EX1003, ¶¶30-31. However, the opinions I expressed in my Initial Declaration or the opinions expressed herein would not change if I applied Mr. Haeberli's proposed level of ordinary skill.

**VI. A POSITA WOULD
HAVE BEEN MOTIVATED WITH A
REASONABLE EXPECTATION OF SUCCESS
TO COMBINE THE PRIOR ART REFERENCES
TO INCREMENTALLY SEARCH ACROSS FILES**

10. Mr. Haeberli agrees that it was known to incrementally search within a single file but disputes whether it would have been obvious to do so across files, allegedly because of the limited memory capacity of 2002-era computers. EX2020, ¶¶8-9, 47-62, 84-88. I disagree for several reasons.

11. *First*, searching across files is not so different from searching within a single file that a POSITA would reasonably expect success with an intra-file search but not an inter-file search. The minor technical difference between the two is that an inter-file search must keep track of the files that contain the search string, whereas an intra-file search cares only about the string's location in the file. But both approaches employ the same general strategy—they locate matching strings in underlying text—and may use indexes to speed up the process (an index for a single document may point to the search string location in the document; an index for multiple documents may point to the document(s) that contain the search string).

12. *Second*, as I explained in my Initial Declaration, searching across files was well known, which Mr. Haeberli does not dispute. EX1003, ¶¶33-44.

13. *Third*, 2002-era computers were entirely capable of incrementally searching across a plurality of (i.e., two or more) files, especially with the help of indexes. For example, the median size of two emails in 2002 was approximately 4.4 kilobytes—0.0044% of the alleged 128-megabyte RAM capacity of 2002-era computers. EX1099, 190; EX2020, ¶48. Mr. Haeberli's assertion that 2002-era computers did not have the volatile memory capacity to perform an incremental search across files is flawed because he assumed an inappropriately large database size. Specifically, he assumed a database size of one gigabyte, without providing any explanation why he selected this particular size. The reality is that then-existing databases were much smaller. For example, email databases were no more than ten megabytes and were often between two and four megabytes. EX1061, 136; EX1095, 1 (Yahoo's pre-2004 email database was four megabytes), 3 (Hotmail's database was only two megabytes); EX1096, 1 (same).

14. Re-running Mr. Haeberli's same analysis (which assumes the index is 5% the size of the database), an index for a four-megabyte database would have been only 200 kilobytes—0.16% the 128-megabyte RAM capacity of 2002-era computers. Thus, 2002-era computers would not have had any issues performing an incremental search across files.

15. Moreover, a POSITA would have expected then-existing index data structures to improve search speed since, as I explained in my Initial Declaration, it was well known that indexes improve search speed and were at the core of virtually every search tool. EX1003, ¶¶45-47 (citing EX1013, EX1023, EX1032, EX1053, EX1054, EX1055, EX1057, EX1059, EX1061).

16. Green, Baeza-Yates, Raskin, and Lotus Notes describe various index data structures that would improve search speed (with the optimal structure depending, of course, on the number and size of the files to be searched, the desired complexity of the search, the type of search, etc.), including the inverted indexes described by the challenged patent (EX1001, 5:29-31), as well as binary trees, B-trees, hash tables, and tries. *E.g.*, EX1018, 1:33-2:48, 5:1-10:57; EX1021, 129, 132; EX1023, 191-227; EX1061, 144. Further, Baeza-Yates, Lotus Notes, and Green disclose indexing and/or searching stems/prefixes, just like the challenged patent (EX1001, 7:37-48). EX1018, 1:33-52 (“... A complete indexing of characters in each word permits very fast exact match and prefix searches ...”), 5:16-6:40, 7:35-8:18; EX1023, 105 (“Prefixes[:] A string which must form the beginning of a text word. For instance, given the prefix ‘comput’ all the documents containing words such as ‘computer,’ ‘computation,’ ‘computing,’ etc. are retrieved.”), 165 (“Selection of index terms to determine which words/stems (or groups of words) will be used as an indexing elements.”), 166, 195 (“Prefix and range queries can also be

solved with binary search, tries, or B-trees”), 202 (“If a suffix tree ... can be afforded, many basic patterns such as [] prefixes ... can be searched ... by a simple trie search.”), 221; EX1061, 144 (“When you have a full-text-indexed database, the following options are available to you: Use word variants: This option tells Notes that you want to look for not only the base word you typed, but also the base word with any possible prefixes”).

17. Green further notes that “[n]umerous techniques are known for partial indexing of word lists on prefix values (starting characters) or word length with a view to reducing the number of words that must actually be compared with a search string. A very well known search technique involves use of a binary tree. The search algorithm associated with a binary tree very quickly reduces the number of lexical nodes that must be compared with a search string.” EX1018, 1:40-47. Green goes on to describe an index data structure that indexes the first letter of each word and the word length, allowing for prefix searches with less memory than a full-text index that indexes every prefix. *Id.*, 1:56-2:48, 5:1-10:57.

18. Mr. Haeberli’s argument that a larger index would allegedly be needed to perform an incremental search misses the point. EX2020, ¶¶16, 33, 47, 59, 75, 88. Even if the index’s size increases, an index generally still increases the speed of the search because it contains less data than the underlying files. EX1003, ¶¶45-47. This

is the entire point of an index. The POR also mischaracterizes my deposition testimony on this point.

19. Specifically, the POR alleges that I agreed that implementing an incremental search requires a much larger index that could slow down the search to a point where the search may no longer be incremental. Paper 21, 63 (citing EX2021, 49:13-52:19); *see also id.*, 19, 23-24 (citing EX2021, 37:7-39:17). Not so. The cited portions of my transcript (EX2021) explain that “assum[ing] that you hold everything else equal ... [if] you simply make that index larger, it could increase the time it took to search it, but it would also depend on how that index is structured. So there could be increases in size ... that would lead to little or no increase in search time.” EX2021, 38:16-39:1. Nowhere in the cited portions of my transcript did I admit that incremental searching requires a “much larger index” as alleged by the POR. Moreover, I did not admit that the index needed to perform an incremental search would slow down the search to a point where the search is no longer incremental. Instead, I responded to questions about whether a system that is unable to provide character-at-a-time results is incremental and I said these questions were “hard to answer” and that “I would have to be careful and look at how the patent describes incremental search.” EX2021, 49:13-52:19. As I explained above in paragraphs 15-17, a POSITA would have expected then-existing indexes to support character-at-a-time results. Moreover, the patent claims do not require character-at-

a-time results because a system that practices the claims does not return results until the user enters a second search string (i.e., the user could enter an entire word without receiving any search results).

20. *Fourth*, as I explained in my Initial Declaration, multiple prior art references, including the Entourage references (EX1004-1006, EX1043), Eudora (EX1039), Nguyen (EX1010), Siitonen (EX1016), Spaey (EX1012), Raskin (EX1021), and True (EX1009) teach incrementally searching across files, confirming that a POSITA would have been motivated with a reasonable expectation of success to implement this functionality.

21. Raskin, for example, teaches incrementally searching across files, not just within a single file, with indexing providing millisecond response times. EX1021, 129, 132. Indeed, when discussing incremental searches, Raskin notes “a few implementation tricks” that can be used to increase the search speed. *Id.*, 129. For example, Raskin explains that algorithms (e.g., “Boyer-Moore algorithm”) and indexes speed up the search. *Id.* “Indexing all local mass-storage devices can make searching over a local system or network take only milliseconds.” *Id.* And when describing how the “search can proceed circularly not only through the local document but also afterward in automatically expanding domains, up to and including the entire Internet,” Raskin explains in a footnote that “[i]t is entirely possible, by such techniques as lookahead and indexing, to maintain the 250 msec

response time” for much larger databases than the average 2002-era email database. *Id.*, 132. *See also* EX1023, 196 (searching for a word on a 250-megabyte database takes 80 milliseconds (0.08 seconds) when using a full inverted index), 220 (a complex search on a 250-megabyte database that allows for errors takes only 0.6 to 0.85 seconds when using a full inverted index), 225 (searching for text on 250-megabyte database using suffix trees takes 0.15-0.3 seconds), 383-384 (confirming that, on a five-megabyte database, even more complex searches (i.e., ones that account for errors, wildcards, etc.) would take only about 250 milliseconds when using an inverted index, given that such a complex search takes only one second on a 20-megabyte database).

22. Mr. Haeberli suggests that Raskin’s incremental search is limited to searching within a file based on Raskin’s discussion of Emacs. EX2020, ¶¶35-37, 66-67. I disagree. Raskin uses Emacs as just one example of a system that implements an incremental search. Mr. Haeberli ignores Raskin’s broader teachings that an incremental search can also be implemented to search across files.

23. Entourage discloses searching across files, such as emails, “[a]s you type the filter text.” EX1004, 9; *see also, e.g., id.*, 8 (“[F]iltering is the fastest way to perform a simple search.”), 38, 161; EX1005, 173; EX1006, 384-386; EX1043, 385 (“The search is conducted as you type”).

24. Eudora similarly discloses displaying email messages as the user types. EX1039, 102-103 (“If you type j in your inbox, Eudora will highlight a message from Jeff Chee. If you type ja (quickly), Eudora will highlight a message from Andy Jain.”); *see also* EX1003 ¶37.

25. Mr. Haeberli’s original declaration asserts that Entourage and Eudora disclose filtering, not searching, but this is a distinction without a difference. EX2001, ¶¶40, 46.

26. First, it contradicts Entourage’s express disclosure, which refers to filtering as a type of search. *E.g.*, EX1004, 8 (“[F]iltering is the fastest way to perform a simple search.”); EX1043, 385 (“The search is conducted as you type”); *see also* EX2021, 127:21-128:17. Second, from a technical perspective, filtering and searching are not meaningfully different. A search across files, for example, filters out the files that do not contain the search string(s) and presents only the files that contain matching strings. Whether the system initially displays (e.g., in a list) the entire set of files before the user initiates the search is immaterial. The system still performs the same search/filter operation. Either way, the system must reference the index to determine which files contain matching strings. Thus, although Entourage displays a list of all emails before the user initiates a search, a POSITA would understand that the search tool (e.g., the Quick Find tool) still performs the same search that it would if it did not display this initial list.

27. The patent itself repeatedly refers to searching and filtering interchangeably. EX1001, Abstract, 2:24-27, 2:51-54, 3:54-58, 7:37-47, 15:44-22:14, 23:3-21, 26:24-36, 28:36-45, FIGS. 3G-3H; EX1098, 60:20-61:14. For example, the patent describes the search process as a filtering process, whereby the results are “incrementally narrowed down” as the user adds characters to their query. EX1001, 15:44-22:14 (describing how “the list pane ... displays incrementally narrowed down search results” as the user enters additional characters to one or more of the search strings); *see also id.*, 2:24-27, 3:54-58 (“To further refine or filter the search results, the user can enter other search terms into other fields. In one embodiment, the filtering process is performed substantially immediately upon the user entering addition[al] search characters or terms.”); 23:3-21, 26:24-36, 28:36-45 (explaining how “with each deletion of a search string character, the search engine [] will reduce the amount of search filtering to broaden the search.”); *see also id.*, FIGS. 3G-3H (illustrating how the search results in an email search interface are filtered from twelve emails to four emails after the addition of the character “7” to the end of the search string “2003-0”). Indeed, the patent’s specification describes the claimed process of locating files containing both the first and second search strings as a filtering process. *Id.*, Abstract (“incrementally filter search results to locate at least a first document that has at least a first word that begins with the first string and a second word that begins with the second string”), 2:51-54 (same). The

patent also implies that “searching” and “filtering” are synonymous because both their speeds are enhanced by indexing prefixes. *Id.*, 7:37-39 (storing prefixes “enhances the speed of incremental filtering or searching”); *see also id.*, 7:40-47.

28. Nguyen discloses incrementally updating search results for “discrete cases” (i.e., “discrete record entities”) “each relating to a separate topic,” such as “solution[s] to [] problem[s] commonly occurring in network printers.” EX1010, 1:60-61, 5:39-47; *see also id.*, 7:14-8:66, FIGS. 2-4B. Nguyen “initiates a search” “if a keystroke ... is detected” (*id.*, 7:33-36) and “[i]f additional keystrokes are detected, the method again conducts ... searches ... as modified by the additional keystroke” (*id.*, 8:34-39). For example, “[t]he first search would be performed on the keystroke ‘p’ and would produce a voluminous number of matches ... [T]he next keystroke [] modify[ing] the search field to ‘pr’ ... would significantly lower the ranking of the POST procedure topic” and a “third keystroke [] modify[ing] the search field to ‘pri’ [would] lower[] the ranking of the various ‘processor’ topics.” *Id.*, 7:68-8:47. “In this manner, by continuing to input additional keystrokes ... the results of each search would be narrowed for each keystroke” *Id.*, 8:51-57. Mr. Haeberli does not dispute that Nguyen teaches incrementally searching across files in at least some instances.

29. Siitonen discloses incrementally displaying search results for “one or more file records containing personal and business records containing names and

calling numbers” (EX1016, 3:24-26), among other contact information such as “addresses, ... facsimile numbers, E-mail and paging addresses” (*id.*, 2:41-43), etc., “similar to a calling card” (*id.*, 1:55). *Id.*, 1:52-56, 2:38-45, 2:51-3:39. “As the user types in the search key, the PDA virtually instantly displays the items matching the search found in the contact data base. ... For example, if the user types the letter ‘j’ all records having names beginning with the letter ‘j’ appear. The user may continue to type additional letters defining a name, for example, the pair of letters ‘on’ chooses records such as ‘Jones’ but not records such as ‘Jackson’. Further typing the letter ‘a’ would eliminate ‘Jones’ as a possibility and display any names having as their first four letters ‘jona’ such as ‘Jonathan’.” *Id.*, 2:51-63.

30. Mr. Haeberli does not dispute that Siitonen discloses incrementally searching for a contact and instead argues that a contact is not a file without providing any supporting explanation. EX2001, ¶44; EX2020, ¶23. Siitonen, however, expressly discloses that the names and calling numbers (the contacts) are stored as “file records” in a database and that the search process involves selecting a “file record” that matches the search query. *E.g.*, EX1016, 3:18-39. Moreover, the challenged patent itself refers to such contact information as a type of file and Mr. Haeberli acknowledged during his deposition that it was known to store each contact record as a file. EX1001, 31:18-21 (claim 13) (“[T]he files comprise ... contact information”), 32:7-9 (claim 17) (“... contacts files”); EX1098, 12:16-13:21.

31. Spaey discloses performing an incremental search across Internet-based files, such as files containing information about CDs, which Mr. Haeberli does not dispute. *E.g.*, EX1012, [0026], [0051-53], [0167] (“Incremental search, i.e. the search process on the server 14 is triggered as soon as a character 68 is typed and narrows down synchronously with each additional character 70(see FIGS. 8A, 8B & 8C).”), [0247], FIGS. 3, 8A-9. For example, typing in the letter “r” yields a first set of search results (*id.*, FIG. 8A), “re” yields a smaller second set of search results (*id.*, FIG. 8B), and “ree” yields an even smaller third set of results (*id.*, FIG. 8C).

32. True discloses incrementally searching across transcript windows, which a POSITA would have understood or at least found obvious to be stored as separate files. Specifically, because True breaks up the transcript into different transcript windows and returns results for each transcript window separately, a POSITA would have understood or at least found it obvious to treat and store these transcript windows as separate files. EX1009, 2:10-58, 4:38-6:23, FIG. 2; EX2021, 141:16-147:6. Indeed, during his deposition, Mr. Haeberli implicitly acknowledged that True’s transcript windows are files since they meet his own definition of a file: they include a sequence of bytes of data that the computer can uniquely identify. EX1098, 10:9-11:8, 51:9-53:22.

A. True's Teaching To Search Text That Is 100% Accurate Would Further Motivate A POSITA With A Reasonable Expectation Of Success To Combine True's Single-Field, Multi-String Teachings With Entourage

33. As I explained in my Initial Declaration, True teaches performing an incremental search for multiple search strings entered in a search field. EX1003, ¶¶123-134, 137-141, 153-175. Mr. Haeberli argues that True is not compatible with Entourage because True searches within a single transcribed audio file, which is not 100% accurate like the pure text of Entourage's files. EX2001, ¶¶132-140; EX2020 ¶¶78-85. I disagree for several reasons.

34. *First*, my Initial Declaration relies on True's teachings to perform an incremental search for multiple search strings entered in a search field, not its alleged teachings to assign confidence levels to a single transcribed audio file. EX1003, ¶¶123-134, 137-141, 153-175.

35. *Second*, True discloses searching text files that are 100% accurate, which Mr. Haeberli acknowledged during his deposition. EX1009, 7:63-8:2; EX1098, 48:5-49:19. Specifically, True describes how the searchable information could be a "text file" where the confidence levels are all "either zero or one rather than continuous values ranging between zero and one." *Id.* As acknowledged by Mr. Haeberli during his deposition, a value of one means 100% accuracy. EX1098, 48:5-49:19.

36. **Third**, as I explain above in paragraph 32, True teaches searching across files since it treats the transcript windows separately and returns individual results for each transcript window. EX1009, 2:10-58, 4:38-6:23, FIG. 2; EX2021, 141:16-142:6, 144:1-7. Indeed, during his deposition, Mr. Haeberli implicitly acknowledged that True's transcript windows are files since they meet his own definition of a file: they include a sequence of bytes of data that the computer can uniquely identify. EX1098, 10:9-11:8, 51:9-53:22.

37. **Fourth**, even if True did not teach searching across files, intra-file search teachings are applicable to inter-file searches because intra-file searches are not meaningfully different than inter-file searches, as I explained above in paragraph 11, and modifying an intra-file search to search across files would be trivial. Specifically, searching for two search strings across multiple portions of a file (e.g., "transcript windows") as described in True is conceptually and functionally equivalent to searching for two search strings across multiple files. In each case, the strings are checked against multiple portions of data (e.g., data in a time range of a transcript file or data within files) that are distinguishable from one another (e.g., by the time ranges of the transcript or by the names of the files), and matches are found and returned. Accordingly, even if True did not store its transcript windows as separate files, a POSITA would understand that modifying True to search across

files would require minimal alterations to the programming, such as the substitution of file name identifiers in the place of time window identifiers.

38. Thus, a POSITA would have had additional reasons to combine True's teachings with Entourage's teachings, including that, like Entourage, True teaches searching text that is 100% accurate, contrary to Mr. Haeberli's assertions.

39. Moreover, True's incremental search teachings are directly relevant to the problem addressed by the '490 patent: slow, cumbersome delimited searches. EX1001, 1:43-2:16; EX1009, 2:25-48, 4:66-5:15.

**B. A POSITA Would Have Reasonably Expected Success
In Implementing The Ground 1 and Ground 2 Prior Art
Combinations Using A Suitable Programming Language**

40. As I explained in my Initial Declaration, a POSITA would have reasonably expected success in implementing the Ground 1 and Ground 2 combinations because the combinations would have required only "simple and straightforward software modifications." EX1003, ¶¶80, 96, 99, 108, 140, 149, 155, 163.

41. Specifically, a POSITA would have reasonably expected success in implementing the grounds' prior art combinations using a suitable programming language such as C or C++, and/or the language(s) that the Lotus Notes and Entourage programs were written in (likely C or C++). As acknowledged by Mr. Haeberli during his deposition, POSITAs were familiar with C, C++, and other

programming languages needed to implement an incremental search tool, as evidenced by the existence of several prior art incremental search tools. EX1098, 21:9-15, 24:17-25:10. A POSITA would have reasonably expected success in using one or more of these programming languages to implement/modify the front-end UI for accepting user's search strings, as well as the back-end logic for building indexes and executing the incremental search using data structures such as those I mentioned in paragraph 16 above.

1. A POSITA Would Have Reasonably Expected Success In Implementing A User Interface That Supports Incremental Searching

42. For Ground 2, Entourage's UI already supports an incremental search. For Ground 1, a POSITA would have reasonably expected success implementing Lotus Notes' user interface (UI) to support incremental searching using an appropriate programming language such as C or C++, or whichever programming language Lotus Notes was built on at the time (likely C or C++). Mr. Haeberli's reference to Raskin's discussion of incremental search being difficult to implement in Visual Basic is a red herring because a POSITA would not use Visual Basic for this task. EX2020, ¶¶8, 60, 65, 70. While Visual Basic allowed a user to build a UI quickly using predefined elements, its capability was limited to these predefined elements, which did not include a dynamic text box or similar UI element needed to implement an incremental search. EX1098, 22:6-24:16.

43. Robust programming languages such as C and C++, together with well-developed operating system and UI frameworks, however, allowed an ordinary programmer to build a UI that supported incremental searching, as evidenced by the numerous prior art systems that implemented such functionality, including Entourage, Eudora, Emacs, Canon Cat, SwyftWare, and others. *E.g.*, EX1004, 8-9, 161; EX1021 at 125, 127, 129; EX1039, 102-103; EX1043, 385-389. Indeed, Mr. Haeberli acknowledged as much during his deposition. EX1098, 21:9-15, 24:17-25:10 (agreeing that a POSITA would have been familiar with C and C++ in 2002 and that a POSITA would have been able to implement a dynamic text box in C or C++, as evidenced by at least Emacs).

44. Implementing such functionality would have required little effort from a programmer. Much of the input-handling logic needed to trigger the computer to update search results was already provided by the operating system and user-interface frameworks; the programmer did not need to write the code from scratch. For example, Windows supplied predefined logic for detecting character-by-character input through the WM_CHAR message, which an application could handle to trigger incremental search and updating of search results. EX1097, 232 (WM_CHAR and WM_SYSCHAR are messages “that deliver text to the window procedure.”), 234 (“If you need to read keyboard character input in your window, you process the WM_CHAR message.”).

45. Moreover, the '490 patent does not provide any details on how to build a user interface that supports an incremental search, further confirming that a POSITA would have had the knowledge to do so and would have reasonably expected success in building such an interface.

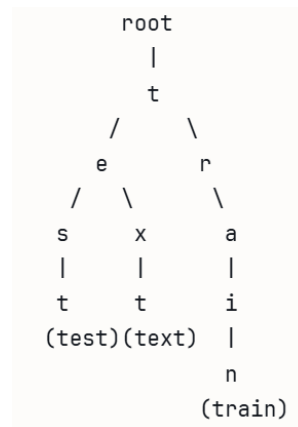
2. A POSITA Would Have Reasonably Expected Success In Implementing The Back-End Logic That Executes The Incremental Searching

46. A POSITA also would have reasonably expected success implementing the back-end logic for building indexes and executing the incremental search using data structures such as those I mentioned in paragraph 16 above and programming in a suitable programming language such as C or C++.

47. For Ground 1, a POSITA would have reasonably expected success in using a suitable programming language (e.g., the language the Lotus Notes was written in) to modify Lotus Notes to initiate the search (and return results) as soon as the user enters two search strings, and to update the results as the user adds additional characters to either of the strings. As I explained above in paragraphs 15-17, a POSITA would have reasonably expected Lotus Notes' existing full-text index and/or another well-known index data structure (e.g., inverted indexes, tries, binary trees) to be able to deliver character-at-a-time results (to the extent the claims require this), especially when searching across smaller file sizes and smaller sets of files. Specifically, the programmer would reasonably expect to use a suitable

programming language to look up a set of matches for the first search string in the index data structure, look up a set of matches for the second search string in the index data structure, and then intersect the results to find the list of files that contain both search strings, as taught by Wu and explained in my Initial Declaration. EX1019, 6:51-66, 8:66-11:32, FIG. 6; EX1003, ¶¶74, 79, 92, 97. When the user adds a character to one of the search strings, the tool would only need to narrow the already-found results for that search string (i.e., locates a set of matches within the first set of matches for the search string), and then intersect the results again with the already-found matches for the other search string. For example, when implementing Lotus Notes' full-text index to perform an incremental search, the index data structure could include associations for each prefix or only whole words. If the data structure indexes only whole words, the search tool would simply join the records of the words that have the partial search string. For example, if Lotus Notes' full-text index is represented as a trie data structure (as discussed in Baeza-Yates and an example of which is shown below), each leaf node would include an indication of the files containing the word. *See, e.g.*, EX1023, 195, 199-200. Thus, in the below exemplary trie, the leaf node for "test" would include a list of the files containing the word "test," the leaf node for "text" would include the list of files containing the word "text," and the leaf node for "train" would include the list of files containing the word "train." If a user enters the single character "t," the computer simply joins

the lists of files associated with these three words. When an “e” is received after the “t” (such that the search string is “te”), the computer joins the lists of files associated with “test” and “text,” and so on and so forth. Alternatively, each node/position in the trie could include a list of the associated files containing the prefix so that the computer does not have to join any lists.



48. For Ground 2, a POSITA would have reasonably expected success in using a suitable programming language (e.g., the language Entourage was written in) to implement an index data structure, as taught by Baeza-Yates, to locate files (e.g., emails) containing two search strings. As I explained above in paragraphs 15-17, a POSITA would have reasonably expected an inverted index or other well-known index data structure (e.g., tries, binary trees) described by Baeza-Yates to store prefixes and/or allow for quick prefix results to be able to deliver character-at-a-time results (to the extent the claims require this), especially when searching across smaller file sizes and smaller sets of files.

49. Specifically, the programmer would reasonably expect to use a suitable programming language to look up a set of matches for the first search string in the index data structure, look up a set of matches for the second search string in the index data structure, and then intersect the results to find the list of files that contain both search strings, as taught by Baeza-Yates and explained in my Initial Declaration. EX1023, 5-27, 101-105, 165-166, 191-228; EX1003, ¶¶136, 139, 146-148, 167-168. See my analysis above (paragraph 47) for Ground 1, which applies equally here for Ground 2.

**C. The Patent's Lack Of Implementation Details
Further Confirms The Reasonable Expectation Of Success**

50. As explained above and in my Initial Declaration, the claimed functionality amounts to nothing more than the combination of well-known, predictable software functionality that a POSITA would have reasonably expected to be able to combine with minimal, straightforward modifications.

51. Moreover, I see nothing in the challenged patent that describes how to scale up its incremental search to the large volumes that Mr. Haeberli seems to allege are required. EX2001, ¶105; EX2020, ¶52. For example, the patent describes how the initial search is in the 'words' file, which stores all the words that occur in the target files. EX1001, 7:1-4. There is no discussion as to how to prevent that file from becoming too large to provide results fast enough to perform the incremental search when searching across a large volume of files.

52. The patent description of storing prefixes suffers from the same problem. EX1001, 7:37-48. This would result in an even larger index, there is no discussion of how exactly that index would be stored, and there certainly is no explanation as to how that index would provide results quick enough to perform an incremental search when searching across a large volume of files. Further, searching the index incrementally would result in a massive set of initial results. For example, because many common words start with ‘a’ (e.g., an, and, etc.) a search of ‘a’ would result in a massive set of results when searching across a large volume of files (i.e., every or nearly every file in the database would be a match), and the challenged patent does not include any discussion of how the system would handle that initial massive set of results.

**VII. A POSITA WOULD HAVE
BEEN MOTIVATED WITH
A REASONABLE EXPECTATION
OF SUCCESS TO COMBINE THE SIDE-BY-SIDE
DISCLOSURES IN THE LOTUS NOTES REFERENCES
(GROUND 1) AND THE ENTOURAGE REFERENCES (GROUND 2)**

53. A POSITA would have been motivated with a reasonable expectation of success to combine the Lotus Notes search tool functionalities relied on in my Initial Declaration, as well as the search tool functionalities described in the Entourage references relied on in my Initial Declaration. EX1003, ¶¶32-53, 82-124, 142-202. Combining these functionalities in a single tool was desirable and/or obvious to try.

A. A POSITA Would Have Found It Obvious To Combine The Functionality From Lotus Notes' Search Tools, And The Query Builder Tool Alone Teaches The Disputed Claim Elements

54. As explained in my Initial Declaration, the Lotus Notes references (Ground 1) teach many of the claim elements within a single search tool (the Query Builder tool) as well as across different search tools. EX1003, ¶¶82-102. That the Query Builder disclosure alone satisfies the mapped claim elements is dispositive.¹

55. Moreover, combining these functionalities in a single tool was desirable and/or obvious to try, as suggested in my Initial Declaration. EX1003, ¶¶40-44, 89.

56. For example, as was known in the art, described by Lotus Notes, and acknowledged by Mr. Haeberli, a user had two options for entering search strings: either they could enter the strings in a single search field or separate search fields. EX1003, ¶¶40-41; EX1059, 161-164; EX1061, 140, 142; EX2020, ¶9. As taught by the Lotus Notes references, the single search field is preferred when searching for just a few search strings, while multiple search fields are preferred when searching

¹ The Search Bar also satisfies the mapped claims at least because it would have been obvious to a POSITA to use a single character for the string separator instead of a Boolean AND, as explained in my Initial Declaration. EX1003, ¶89. Indeed, Mr. Haeberli acknowledged during his deposition that single-character string separator characters were well known. EX1098, 40:10-17.

for many strings. EX1061, 140 (explaining that the single-field Search Bar tool allows “you to find all the occurrences of a word (or a couple of words) very, very quickly”), 142 (explaining that the multi-field Search Builder is a better “option when you want to look for more than just a couple of words or phrases”); *see also* EX1003, ¶¶40-44. Thus, when performing a search on only two search strings, as recited in the claims, a POSITA would have been motivated to use a single search field, as in the Search Bar and Query Builder.

57. As was also known in the art, described by Lotus Notes, and acknowledged by Mr. Haeberli, a user had two options for separating search strings when entering multiple search strings in a single search field: either they could separate the string with a single character (e.g., a comma as in the Query Builder) or a Boolean operator (e.g., an AND as in the Search Bar). EX1059, 162-163; EX1061, 140; EX2020, ¶9; *see also* EX1003, ¶89. As explained in my Initial Declaration, a POSITA would have understood that when performing an incremental search, a single character would be preferred at least because an incremental search is designed to minimize the number of characters the user has to enter before obtaining the desired result. EX1003, ¶89.

58. For at least these reasons, even if Lotus Notes did not expressly disclose the disputed claim elements within a single search tool (which it does), it is my opinion that a POSITA would have been motivated with a reasonable expectation of

success to combine the well-known functionality described in Lotus Notes' cited search tools at least because they represented a limited number of well-known implementation options, each with its own recognized advantages.

**B. A POSITA Would Have Found It Obvious To
Combine The Functionality From The Entourage Search Tools**

59. I disagree with Mr. Haeberli's assertion that a POSITA would not have combined the search tools' functionality simply because they are described in separate search tools. A POSITA would not read a reference so rigidly and myopically, especially for flexible and predictable software like in Entourage and Lotus Notes.

60. As just one example, a POSITA would not read Entourage's Quick Find tool to be incapable of handling multiple search strings simply because Lotus Notes describes this functionality in the Advanced Find tool. If anything, a POSITA would understand that such multi-string functionality would be inherent and/or easy to include in the Quick Find tool because such functionality utilizes the same or similar approach for performing a single-string search and would not expect it to require significant extra computing power or time. For example, if the user types "c," the Quick Find tool returns results containing the words that start with "c." If the user then types a second string "h" (so that the query is "c h"), the Quick Find tool would locate emails containing words that start with "h" using the same approach that it did to find emails containing words that start with "c." Then, as was well known in

the art, the tool would simply intersect the results (e.g., perform a Boolean AND operation). EX1003, ¶¶42-43, 71, 74-75, 79, 81, 92, 97-98, 139, 167-171.

61. As another example, a POSITA would not read Entourage's Quick Find tool to be incapable of highlighting matching search strings simply because Lotus Notes describes this functionality in the Find tool. If anything, a POSITA would have found it trivial to include the Find tool's highlighting functionality in the Quick Find tool. There would be little to no modifications needed. Specifically, the Quick Find tool already finds results with matching strings. Thus, the part of the Find tool's code that instructs the program to highlight the matching string could likely be copied over with little to no modifications.

VIII. THE PRIOR ART TEACHES EVERY ELEMENT OF THE INDEPENDENT CLAIMS

62. As explained in my Initial Declaration, the prior art in each ground teaches every element of the independent claims. EX1003, ¶¶82-124, 142-202.

A. The Ground 1 References Teach Element [1.5]

63. As I explained in my Initial Declaration, the Lotus Notes references teach element [1.5]. *E.g.*, EX1003, ¶84.

64. Moreover, the claims do not recite indexing prefixes as suggested by the POR, and even if they did, indexing prefixes was known in the art, as I explained above in paragraphs 16-17. Because an incremental search returns results as the user types, it would have been obvious when implementing an incremental search tool to

use any of the known index data structures that supported prefix searching (e.g., inverted indices, Boolean trees, tries, etc.). EX1018, 1:33-2:48, 5:1-10:57; EX1021, 191-227; EX1061, 144. Returning results before a user has entered a full word necessarily requires this capability. EX2021, 105:6-11. Further, a system that indexes only whole words inherently indexes prefixes because common words that would be indexed in any system (e.g., the word “a”) are prefixes for other words (e.g., “an,” “and,” “Andrew,” etc.), as acknowledged by Mr. Haerberli during his deposition. EX1098, 25:11-27:1.

IX. THE PRIOR ART RENDERS THE DEPENDENT CLAIMS OBVIOUS

65. The disputed dependent claims relate to highlighting search strings in the results (31:3-14 (claims 10-11)) and searching within specific fields and/or file types (30:35-59, 31:22-32:18 (claims 4-7, 14-19)). As I explained in my Initial Declaration, these functionalities and their benefits were known to POSITAs. Specifically, search term highlighting directs the user to the relevant portion(s) of the results and gives the user more confidence in the results. EX1003, ¶¶48-52, 138, 155, 188-194. Searching within specific fields and/or file types helps narrow the search results and provide quicker responses. EX1003, ¶¶31, 53, 105, 108, 116, 180, 183-185, 197-198, 200-201. Indeed, it was self-evident to a POSITA that searching within a specific field of the files (e.g., the “To” field of emails) and/or searching within a specific file type will return results more quickly than searching the entire

contents of all files because there is less data to search through. *E.g.*, EX1004, 8-9 (field-specific search), 11 (file-type specific search), 161-162 (field-specific search and file type-specific search); EX1006, 18 (“Find File ... lets you search for a document based on its name file type, and location”), 384-387 (field-specific search and file type-specific search); EX1008, 471-473 (field-specific search); EX1023, 108, 391 (“If the user can restrict the search to a field (for example, the page title) or limit some attribute (date, country), this will certainly reduce the size of the answer.”); EX1043, 385-389 (field-specific searches and file type-specific searches); EX1057, 60, 66 (field-specific search), 100 (file-type specific search), 110-111 (same), 210; EX1059, 167-168 (field-specific search); EX1061, 141-142 (field-specific search). Thus, it is still my opinion that these claims would have been obvious at least because they recite well-known functionality that had established advantages.

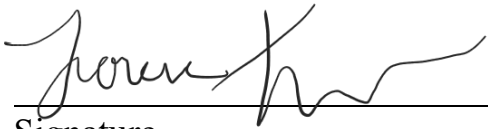
66. The highlighting claims recite highlighting search strings in the same user interface as the search results, which a POSITA would have understood can include multiple windows, not just one, as Mr. Haeberli acknowledged at his deposition. EX1001, 29:55-30:28 (claim 1), 30:66-31:13 (claims 9-11); EX1098, 18:19-20:5; EX1101, 162-163, 358. Further, even if the claims did recite highlighting search strings in the same window as the search results, this too would have been obvious at least because it was known to do so, was one of only a limited

number of known options (either the highlighting is shown in a separate window or the same window as the search results), and was nothing more than a simple design choice. EX1003, ¶¶48-51, 193.

67. For the field and file type claims, I also disagree that it would not have been obvious to use Entourage's incremental search techniques to search across other text-based files. Searching the text of emails is not meaningfully different than searching the text of contacts, calendar events, or word processing documents. They all involve the same underlying approach, as evidenced by the fact that Entourage's Advanced Find tool "enables you to search multiple item types simultaneously." EX1004, 11; *see also id.*, 161-162; EX1006, 18, 384-387; EX1043, 385-389; EX1057, 100, 110-111. Moreover, it would have been obvious and/or common sense to use the attribute fields associated with a specific type of document when searching within that document type (e.g., to use email attribute fields such as "To," "From," etc., when searching across emails).

I, Loren G. Terveen, do hereby declare and state, that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: February 12, 2026


Signature