UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
14/451,109	08/04/2014	David F. WILEY	111563-0110 (STR-002-DIV)	7974

38706 7590 03/06/2020
FOLEY & LARDNER LLP
3000 K STREET N.W.
SUITE 600
WASHINGTON, DC 20007-5109

EXAMINER	
SHECHTMAN, CHERYL MARIA	

ART UNIT	PAPER NUMBER
2157	

NOTIFICATION DATE	DELIVERY MODE
03/06/2020	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

ipdocketing@foley.com

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

Ex parte DAVID F. WILEY

Appeal 2018-008013
Application 14/451,109
Technology Center 2100

Before BRADLEY W. BAUMEISTER, JASON V. MORGAN, and DAVID J. CUTITTA II, *Administrative Patent Judges*.

MORGAN, *Administrative Patent Judge*.

DECISION ON APPEAL

STATEMENT OF THE CASE

Introduction

Pursuant to 35 U.S.C. § 134(a), Appellant[1] appeals from the Examiner's decision to reject claims 1–3, 5–12, and 14–20. Claims 4 and 13, having been incorporated in claims 1 and 10 respectively, are canceled. Amend. After Final 6 (filed Dec. 4, 2017; entered Jan. 9, 2018). An oral hearing was held February 26, 2020. A transcript will be filed in due course. We have jurisdiction under 35 U.S.C. § 6(b). We REVERSE.

[1] We use the word "Appellant" to refer to "applicant" as defined in 37 C.F.R. § 1.42. Appellant identifies the real party-in-interest as Stratovan Corporation. Appeal Br. 2.

Summary of the disclosure

Appellant's claimed subject matter relates to automated file format capturing. Spec. ¶ 76. Specifically, "the file format specification is automatically derived from the source code so that there is only one point of origination and maintenance." *Id.* ¶ 81.

Illustrative claim (*key limitations emphasized*)

> 1. A method implemented by a computer for automatically capturing file format information directly from the file input/output, comprising:
>
> identifying source code corresponding to certain file operations that perform the file input/output during execution thereof;
>
> *modifying the source code corresponding to the certain file operations so as to change the operation of the certain file operations to cause them to also capture file format information during execution thereof*, wherein without modifying the corresponding source code, the certain file operations would not capture the file format information during execution thereof, *wherein modifying source code includes instrumenting write operations such that the file format information is captured by a dummy file capturing object*;
>
> collecting, after modifying the corresponding source code, the file format information during execution of the certain file operations in a captured file object; and
>
> processing the captured file object to obtain the file structure specification.

The Examiner's rejection and cited references

The Examiner rejects claims 1–3, 5–12, and 14–20 under 35 U.S.C. § 103(a) as obvious over Venkatapathy (US 2004/0117771 A1; published June 17, 2004), Panchenko et al. (US 8,060,869 B1; issued Nov. 15, 2011) ("Panchenko"), Birum et al. (US 7,100,152 B1; issued Aug. 29, 2006)

(“Birum”), and Arora et al. (US 6,662,362 B1; issued Dec. 9, 2003) (“Arora”). Final Act. 9–18.

ANALYSIS

In rejecting claim 1 as obvious, the Examiner finds that Venkatapathy’s insertion of instrumentalization into source code teaches or suggests “**modifying the source code corresponding to the certain file operations so as to change the operation of the certain file operations during execution thereof**.” Final Act. 10 (citing Venkatapathy ¶ 21, Fig. 2, Abstract). The Examiner finds that Panchenkos’s identification of a function that accesses memory (i.e., performs either a read or write access) teaches or suggests “source code including **file operations performing file input/output**.” *Id.* (citing Panchenko Fig. 4); *see also* Ans. 13 (citing Panchenko col. 11, l. 63–col. 12, l. 4).

The Examiner finds Birum’s use of filter tags instrumented into source code teaches or suggests “**modifying of source code causing file operations to also capture file format information during execution thereof**.” Final Act. 10–11 (citing Birum Abstract, Fig. 4, col. 8). The Examiner finds Panchenko’s identified write access functions and Arora’s formatted trace data teaches or suggests “**modifying source code includes instrumenting write operations such that the file format specification is captured by a dummy file capturing object**.” *Id.* at 14–15 (citing Panchenko Fig. 4; Arora col. 8, ll. 5–30); *see also* Adv. Act. 2 (citing Arora col. 7, ll. 54–61, col. 8, ll. 5–14, 27–30, col. 10, ll. 7–32, Figs. 2E, 3) (Jan. 9, 2018); Ans. 15.

The Examiner proffers that it would have been obvious to an artisan of ordinary skill to combine these references for the purposes of "detecting memory problems in user programs" (Final Act. 11 (citing Panchenko col. 1, ll. 6–10)), "providing greater flexibility in analyzing software (*id.* at 12 (citing Birum col. 9, ll. 13–15)), and "identifying locations in a source code of inefficient coding constructs so that an application developer can transform the inefficient source code to more efficient coding constructs" (*id.* at 13 (citing Arora col. 2, ll. 29–34)). Therefore, the Examiner finds that the cited references teach or suggest the claim 1 recitations of: (1) modifying the source code corresponding to the certain file operations so as to change the operation of the certain file operations to cause them to also capture file format information during execution thereof and (2) wherein modifying source code includes instrumenting write operations such that the file format information is captured by a dummy file capturing object. *Id.* at 10–15.

Appellant contends the references fail to teach or suggest capturing *file format information* in the manner recited for several reasons. First, "Venkatapathy merely analyzes source code to identify code segments that could possibly contain errors." Appeal Br. 7. Second, "Panchenko identifies functions in binary code . . . including instructions in the functions that read or write memory." *Id.* at 6. Third, Birum's tags perform dedicated writes for "program tracing, data tracing and memory allocation, and can be filtered as such." *Id.* at 7. And fourth, "Arora merely teaches inserting completely new 'trace hooks' into source code at places of interest." *Id.* at 8.

In particular, Appellant argues that Arora's description of "how the trace file can [be] used to determine patterns of how/when methods start and end" does not teach or suggest "capturing file formation information from file write operations." *Id.* at 9; *see also* Reply Br. 4. Appellant's arguments are persuasive.

Venkatapathy "relates to defect analysis in software." Venkatapathy ¶ 1. Panchenko's instrumentation of memory access instructions catches and reports "memory access errors such as array bounds read/write, allocating zero size, freeing wrong memory block, bad realloc parameter, double freeing memory, freed memory read/write, freed realloc pointer, invalid memory read/write, partially initialized read, unallocated read/write, uninitialized memory read." Panchenko col. 14, ll. 27–32. Birum uses tags to allow filtering of data collected in applications such as "program tracing, data tracing, and memory allocation." Birum col. 8, ll. 1–6. And Arora's "[t]race hooks are typically inserted for the purpose of debugging, performance analysis, or enhancing functionality." Arora col. 8, ll. 9–11.

None of the uses described in the cited portions of the references pertain directly to the recited application of modifying source code to capture *file format* information. That is, although the cited references are pertinent to software instrumentation in general (either through modification of source code or binary code), the Examiner's findings fail to show that the references teach or suggest applying such instrumentation in particular to identified *certain file operations* such that the *file format* information is captured.

The Examiner emphasizes that "Arora further discloses that the post-processing of the trace file includes analyzing the data collected in records in the trace file . . . to detect patterns of paired entry-exit records for method types including 'native method B', 'JVM_Recv', and 'JNI'." Adv. Act. 2 (Jan. 9, 2018) (citing Arora col. 8, ll. 27–30, col. 10, lines 7–32, Figs. 2E, 3).

The Examiner finds this disclosure teaches or suggests "file format information [being] captured by the dummy file capturing object." *Id.*; *see also* Ans. 15.

But Arora's only mention of a file format relates to trace record information that "may be formatted for output in the form of a report." Arora col. 8, ll. 28–29. The information Arora captures on paired entry-exit records is used to identify "inefficient use of a cross-language boundary, such as the interface between Java code and native code, in which data is inefficiently transferred across the interface through a local array in the native code." *Id.* col. 2, ll. 52–55.

That is, Arora "is directed to a methodology of detecting and remedying poor use of a cross-language interface by identifying inefficient, cross-language, coding constructs for transformation to more efficient constructs." *Id.* col. 7, ll. 43–47. Thus, merely creating a report based on such records to identify inefficient use of a cross-language boundary does not represent the instrumentation of "write operations such that the file format information is captured" in the manner recited.

Because the Examiner's findings fail to show the cited references teach or suggest modifying source code to capture file format information in the manner recited, we do not sustain the Examiner's 35 U.S.C. § 103(a) rejection of claim 1, or of claims 2, 3, 5–12, and 14–20, which contain similar recitations.

CONCLUSION

Claims Rejected	**35 U.S.C. §**	**References**	**Affirmed**	**Reversed**
1–3, 5–12, 14–20	103(a)	Venkatapathy, Panchenko, Birum, Arora		1–3, 5–12, 14–20

REVERSED