# IN THE UNITED STATES PATENT TRIAL AND APPEAL BOARD

In the *Inter Partes* Review of U.S. Patent No. 7,110,936

Trial No.: Not Yet Assigned

Issued: September 19, 2006

Filed: November 19, 2001

Inventors: Fen Hiew, *et al.*

Assignee: Complementsoft LLC

Title: SYSTEM AND METHOD FOR GENERATING AND MAINTAINING SOFTWARE CODE

**MAIL STOP *PATENT BOARD***
Patent Trial and Appeal Board
United States Patent & Trademark Office
P.O. Box 1450
Alexandria, Virginia 22313-1450

## PETITION FOR *INTER PARTES* REVIEW UNDER 37 C.F.R. § 42.100

On behalf of SAS Institute, Inc. ("SAS" or "Petitioner") and in accordance with 35 U.S.C. § 311 and 37 C.F.R. § 42.100, *inter partes* review is respectfully requested for claims 1-16 of U.S. Patent No. 7,110,936 ("the '936 patent"), attached hereto as Exhibit 1001.

The undersigned representative of Petitioner authorizes the Patent Office to charge the $27,200 Petition Fee, along with any additional fees, to Deposit Account 501432, ref: 343355-610009. Sixteen claims are being reviewed; accordingly no excess claim fees are required.

## Table of Contents

## I. Introduction

The ‘936 patent is currently being wielded by the patent owner, Complementsoft LLC (“Complementsoft”), in an attempt to cover long-known systems and methods for developing software. (*ComplementSoft, LLC v. SAS Institute Inc.*, Docket No. 1:12-cv-07372 (N.D. Ill. Sept 14, 2012).) The ‘936 patent is directed to an integrated development environment for generating and maintaining source code and discloses four elements that work in concert – a document manager, an editor, a parser layer, and a visualizer. (*See, e.g.*, ‘936 patent, claim 1.) The subject matter claimed in the ‘936 patent includes standard elements that were well known in the prior art before the filing date of the ‘936 patent, and this was acknowledged by the Office.[1] Complementsoft itself has conceded that numerous claim elements of the ‘936 patent were disclosed in

[1] In a Supplemental Notice of Allowability for the ‘936 patent, the Office stated:

> “Applicants are disclosing a software development environment where a user can simultaneously view a graphical representation and a text representation of source code including views that are synchronized such that modifications to one view are automatically reflected in the other view, including the ability to detect the particular language of the source code by applying rules. This has been disclosed in the prior art of record.” (Ex. 1004, Supplemental Notice of Allowability dated August 1, 2006, at 3-4.)

the prior art, as evidenced in documents filed in the related litigation. (*See* Ex. 1016, Complementsoft's LPR 2.5 Initial Response to SAS Institute's Invalidity Contentions.)

In fact, during prosecution of the '936 patent, the Office found that *all* four elements of claim 1 – the document manager, editor, parser layer, and visualizer – were disclosed in a single reference and in the same arrangement as claimed in the '936 patent. (Ex. 1002, Office Action dated May 31, 2005, at 3, noting "[The Coad reference] clearly anticipates the claimed limitations of independent claims 1 and 18.")

Despite this finding, however, Complementsoft was able to gain allowance based on a single aspect that allegedly distinguished the prior art from the pending claims. During prosecution, Complementsoft attempted to distinguish the prior art, which was applicable to object-oriented languages, from the amended claims, that were modified to recite data manipulation languages.[2] Complementsoft argued that the object-oriented languages used in the prior art were "structured" programming languages, while the data manipulation languages of the amended claims were "unstructured" programming languages, and that the claims were thus distinguishable from the prior art on this basis.

Complementsoft's argument, however, is technically misleading because a person of ordinary skill in the art would understand that most data manipulation

[2] As filed, the claims were directed to any form of "source code."

languages are in fact structured programming languages, and the claims should not have been allowed on this basis. (*See* Ex. 1015, ¶¶ 24-60.) Further, the scope of claim 1, covering both program flows *and* data flows, is inconsistent with the argument set forth by Complementsoft. This is because the distinction argued by Complementsoft (*i.e.*, that displaying flow in unstructured programming languages is distinguishable from displaying flow in structured programming languages) applies, if at all, to displaying *program flows* only. (*Id.* at ¶¶ 61-68.)

Moreover, as described below, the claimed software development environment was not new when the '936 patent was filed. As noted in the background of the '936 patent, GUI-based software editor packages are very old, and the problems inherent with such software were common knowledge in the field: "There are also a few GUI based editor software packages to make editing code easier. . . . What is lacking, however, is [an] . . . editor integrated with [a] diagramming package so the user does not have to manually generate and update program flow or data flow diagrams." ('936 patent at col. 1, lines 52-63.) The '936 patent purports to address these problems by dynamically linking an editor and a visualizer, such that edits made to source code using the editor are automatically reflected in graphical representations of flows displayed by the visualizer, and vice versa. (*See, e.g.*, *id.* at claim 1.) But the linking of an editor and a visualizer was a common component of software development environments at the time that the '936 patent was filed.

Accordingly, the lack of inventiveness is evidenced herein by the refutation of the argument presented during prosecution with respect to Coad. The lack of inventiveness is further shown, *inter alia*, by the submission of U.S. Patent No. 5,572,650 (Ex. 1005, "Antis"), which issued more than five years prior to the filing of the '936 patent in November 2001. Antis discloses all of the limitations of the independent claim of the '936 patent, including the visualization of flows within source code programmed using a data manipulation language. Specifically, Antis discloses a "method and apparatus for visually displaying structural characteristics of a large database." (Antis at Abstract.) Antis discloses the display of graphical representations of flows in source code, and, in particular, flows in *data manipulation language* source code. In fact, the types of flows disclosed in the Antis patent are *nearly identical* to those described and claimed in the '936 patent. (*See*, *e.g.*, Fig. 17 of the '936 patent, disclosing flow by using arrows to display the data that is accessed by retrieved pieces of source code, and Antis at col. 7, line 31 to col. 8, line 4, disclosing a "code view" that provides a visual representation of the database relations that are accessed by a selected piece of source code.)

The dependent claims of the '936 patent add nothing more than well known software editing concepts that are explicitly disclosed in the Coad reference applied during prosecution or in one or more additional prior art references that are analogous art to Antis and that one of ordinary skill in the art would be motivated to combine with Antis in generating an integrated software development environment.

Petitioner submits that had the references presented herein been properly considered by the Patent Office during prosecution, claims 1-16 of the '936 patent would not have issued, and therefore this petition for *inter partes* review should be granted.

## II. Grounds for Standing Pursuant to 37 C.F.R. § 42.104(a)

SAS certifies that the '936 patent is available for *inter partes* review and that SAS is not barred or estopped from requesting *inter partes* review challenging the patent claims on the grounds identified herein.

## III. The '936 Patent

### A. Summary of the '936 Patent

The '936 patent was filed on November 19, 2001, and issued on September 19, 2006. The '936 patent claims priority to U.S. Provisional Application Nos. 60/270,950 and 60/293,854, filed on February 23, 2001, and May 25, 2001, respectively.

The '936 patent is directed to an integrated software development environment for generating and editing computer code. ('936 patent at Abstract.) The integrated software development environment includes four modules: (1) a document manager, (2) an editor, (3) a parser layer, and (4) a visualizer. (*See id.* at col. 5, lines 4-25.) The document manager is a file management program used to retrieve source code for editing. (*Id*. at col. 6, lines 22-42.) The retrieved source code is edited using the editor module, which "allows the user to perform standard text editing functions, including, mouse placement of the cursor, click-and-drag text selection and standard Windows key combinations for cutting, copying and pasting data." (*Id.* at col. 7, lines 4-7.)

The parser layer and the visualizer are used to display graphical representations of flows within the retrieved source code. The parser layer is "language aware," such that the integrated software development environment of the '936 patent is able to develop and edit code of multiple different programming languages. (*Id.* at col. 17, lines 17-32; *see also id.* at col. 9, lines 49-53, disclosing use with SAS, SQL, SPSS, DB2 UDB, Oracle, and PL/SQL languages.) Using the rules and logic that correspond to the detected programming language, the file parser breaks the retrieved source code into individual words or tokens. (*Id.* at col. 17, lines 46-49.) The file parser also tags each token to associate it with a particular class. (*Id.* at col. 17, lines 48-54.)

Tagging of the tokens enables the visualizer module to recognize and display the flow of the retrieved source code. (*Id.* at col. 17, lines 54-56.) Fig. 17, for example, is an exemplary screenshot from the visualizer module depicting a flow of retrieved source code. In Fig. 17, the visualizer module displays at 120 icons 126 with arrows that illustrate data flow in the retrieved source code. The flow thus provides an indication of what data (*e.g.*, relations smpl.smpl_1, smpl.smpl_2) are accessed by which modules of source code (*e.g.*, source code used to perform a sort operation).

The visualizer module is dynamically linked with the editor module, such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer, and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor. (*See, e.g.*, *id.* at claim 1 and col. 16, lines 31-46.)

**B. The '936 Patent Prosecution History**

The '936 patent was allowed after responses to two non-final Office Actions were filed. The first non-final Office Action rejected the original claims based upon the Coad reference (Ex. 1006, U.S. Patent No. 6,851,107). The Office found that all four elements of independent claim 1 were disclosed in the Coad reference and in the same arrangement as claimed in the '936 patent. (Ex. 1002, Office Action dated May 31, 2005, at 2-3, noting that "Coad clearly anticipates the claimed limitations of independent claims 1 and 18.") In an attempt to distinguish the pending claims from Coad, Complementsoft amended independent claim 1 as follows:

> 1. (Currently Amended) An integrated development environment, comprising:
>
> a document manager for retrieving source code programmed using one of a plurality of types of data manipulation languages;
>
> an editor for displaying the retrieved source code and providing a means for a user to edit the retrieved source code;
>
> a parser layer which detects the one of the plurality of types of data manipulation languages in which the ~~software language of the~~ retrieved

> source code is programmed and which activates rules and logic applicable to the detected one of the plurality of types of data manipulation languages ~~software language~~; and
>
> a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code using the rules and logic applicable to the detected one of the plurality of types of data manipulation languages and activated by the parser, wherein the editor, parser layer and visualizer cooperate such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor.

In its remarks, Complementsoft did not dispute that all four elements of the independent claim (*i.e.*, the document manager, editor, parser, and visualizer) were disclosed in Coad. Instead, Complementsoft argued:

> Considering now Coad, Coad discloses a software development tool for use with object-oriented programming languages, such as Java or C++. More particularly, Coad discloses a system in which the object-oriented language of the source code is identified by the extension of a retrieved file to thereby identify a template which is to be used to convert the source code into a transient meta model (TMM), *i.e.*, a language-neutral representation of the source code. The system then uses the TMM to display a graphical representation of a project.
>
> From the foregoing it will be appreciated that Coad fails to expressly or inherently disclose the claimed system and method. More

> particularly, rather than disclose the claimed system and method for examining software written using one of the plural **data manipulation languages, *i.e.*, a language which suffers the problem of being unstructured** (*see* Background section of subject application), the system and method described in Coad is limited to software written using an **object oriented language, *i.e.*, a structured programming language** such as Java or C++. . . . [T]he system and method of Coad leverages the **structured** nature of the programming language to merely "convert" the source code into a language-neutral representation and then use the language-neutral representation to display a graphical representation of the project. (Ex. 1003, Response to Office Action dated August 4, 2005, at 9-10, emphasis added.)

Complementsoft thus argued that the object-oriented languages of Coad were "structured" programming languages, while the data manipulation languages of the claims were "unstructured" programming languages, and that the claims were distinguishable from Coad on this basis.

Claim 1 ultimately issued in the form shown above. A supplemental notice of allowability provided the following with respect to claim 1 overcoming Coad:

> Coad further discloses the ability to detect the particular language of the source code and applying rules. However, Coad does not explicitly disclose the specific arrangement of elements including a document manager, editor, parser layer, or visulizer [sic] as noted above, or that the detected language is a <u>data manipulation language</u>. (Ex. 1004, Supplemental Notice of Allowability at 5, emphasis in original.)

## IV. Identification of Challenge Pursuant to 37 C.F.R. § 42.104(b)

### A. 37 C.F.R. § 42.104(b)(1): Claims For Which *Inter Partes* Review Is Requested

*Inter partes* review is requested for claims 1-16 of the '936 patent.

### B. 37 C.F.R. § 42.104(b)(2): The Prior Art and Specific Grounds On Which The Challenge to the Claims Is Based

*Inter partes* review is requested in view of the following prior art references:

- U.S. Patent No. 5,572,650 to Antis *et al*. ("Antis") (Exhibit 1005). Antis was filed on June 30, 1994, and issued on November 5, 1996. Antis is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).
- U.S. Patent No. 6,851,107 to Coad *et al.* ("Coad") (Exhibit 1006). Coad was filed on October 4, 2000, and issued on February 1, 2005. Coad is therefore prior art to the '936 patent under 35 U.S.C. § 102(e).
- U.S. Patent No. 6,356,285 to Burkwald *et al.* ("Burkwald") (Exhibit 1007). Burkwald was filed on December 17, 1997, and issued on March 12, 2002. Burkwald is therefore prior art to the '936 patent under 35 U.S.C. § 102(e).
- U.S. Patent No. 5,937,064 to Eick *et al.* ("Eick") (Exhibit 1008). Eick was filed on March 3, 1997, and issued on August 10, 1999. Eick is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).
- "Microsoft Access 97 Visual Basic Step by Step" (hereinafter "Access 97 Visual Basic") (Exhibit 1009). Access 97 Visual Basic was published in

1997. Access 97 Visual Basic is therefore prior art to the ‘936 patent under 35 U.S.C. § 102(b).

- U.S. Patent No. 5,758,122 to Corda *et al.* (“Corda”) (Exhibit 1010). Corda was filed on March 15, 1995, and issued on May 26, 1998. Corda is therefore prior art to the ‘936 patent under 35 U.S.C. § 102(b).
- “Building Applications with Microsoft Access 97” (hereinafter “Building Applications”) (Exhibit 1011). “Building Applications” was published in 1996. “Building Applications” is therefore prior art to the ‘936 patent under 35 U.S.C. § 102(b).
- “Oracle Programming – A Primer” (hereinafter “Oracle Primer”) (Exhibit 1012). Oracle Primer was published in 1999. Oracle Primer is therefore prior art to the ‘936 patent under 35 U.S.C. § 102(b).
- “Oracle8 Programming: A Primer” (hereinafter “Oracle8 Primer”) (Exhibit 1013). Oracle8 Primer was published in 1999. (*See* Ex. 1017.) Oracle8 Primer is therefore prior art to the ‘936 patent under 35 U.S.C. § 102(b).

The specific statutory grounds on which the challenge to the claims is based and the patents relied upon for each ground are as follows:

a) Claim 1 is anticipated by Coad under 35 U.S.C. § 102(e).

b) Claim 1 is unpatentable under 35 U.S.C. § 103(a) over Coad in view of Oracle Primer and Oracle8 Primer.

c) Claims 1-3 and 5 are anticipated by Antis under 35 U.S.C. § 102(b).

d) Claims 1-3, 5, 6, 8, 10-12, 15, and 16 are unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad.

e) Claim 4 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and Burkwald.

f) Claim 7 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and Eick.

g) Claim 9 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and "Building Applications."

h) Claim 13 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and Corda.

i) Claim 14 is unpatentable under 35 U.S.C. § 103(a) over Antis in view of Coad and Access 97 Visual Basic.

j) To the extent not explicitly enumerated above, claims 2-16 are unpatentable over each reference and combination of references asserted for claim 1 in view of the prior art.

### C. 37 C.F.R. § 42.104(b)(3): Claim Construction

Pursuant to 37 C.F.R. § 42.100(b), and solely for the purposes of this review, Petitioner construes the claim language such that the claims are given their broadest reasonable interpretation in light of the specification of the '936 patent. For terms not specifically listed and construed below, Petitioner interprets them for purposes of this review in accordance with their plain and ordinary meaning under the required broadest reasonable interpretation.

"***automatically***" — Claim 1. For purposes of this review, Petitioner adopts the construction "*without user intervention*" for the term "automatically." This is consistent with the '936 patent's specification as well as applicable case law. In the specification, for example, the '936 patent states: "Preferably, the visualizer and editor are integrated so that changes made to the code in the editor are *immediately* reflected in the visualizer and vice-versa." ('936 patent at 2:42-45; emphasis added). The term "automatic" has been construed previously in other litigations involving software technology. *See*, for example: *Collegenet, Inc. v. Applyyourself, Inc.*, 418 F.3d 1225 (Fed. Cir. 2005) (affirming the District Court's interpretation that the claim term "automatically" means "once initiated, the function is performed by a machine, without the need for manually performing the function."); *Eolas Technologies, Inc. v. Adobe Systems, Inc.*, 810 F. Supp.2d 795 (E.D. Texas 2011) (finding that the claim term "automatically invoking the executable applications" meant that the executable application is launched without user activation.); and *Rockwell Electronic Commerce Corporation v. Apropos Technology, Inc.*, 2002 U.S. Dist. LEXIS 272 (N.D. Illinois 2002) (finding that the claim term "automatically" in several phrases means "initiated and performed without human intervention.")

“***data manipulation language***” — Claim 1. For purposes of this review, Petitioner adopts the construction “*a programming language used to access data in a database, such as to retrieve, insert, delete, or modify data in the database*” for the term “data manipulation language.” (*See* Ex. 1015, ¶¶ 48-54.) This is consistent with the ‘936 patent’s specification. In the specification, for example, the ‘936 patent cites SQL and Oracle as data manipulation languages for accessing data in a database. (*See, e.g.*, ‘936 patent at col. 1, lines 20-25.) Based on such disclosures, the adopted claim construction for “data manipulation language” is consistent with the disclosure in the ‘936 patent with respect to data manipulation languages.

Because the standard for claim construction at the Patent Office is different than that used during a U.S. District Court litigation, *see In re Am. Acad. Of Sci. Tech Ctr.*, 367 F.3d 1359, 1364, 1369 (Fed. Cir. 2004); MPEP § 2111, Petitioner expressly reserves the right to argue a different claim construction in litigation for any term of the ‘936 patent as appropriate in that proceeding.

### D. 37 C.F.R. § 42.104(b)(4): How the Construed Claims are Unpatentable

A detailed explanation of how claims 1-16 are unpatentable is set forth below at Section V.

### E. 37 C.F.R. § 42.104(b)(5): Supporting Evidence

An Appendix of Exhibits supporting this Petition is attached. Included at Exhibit 1015 is a Declaration of Dr. Nick Roussopoulos under 37 C.F.R. § 1.68. References discussed in the Declaration of Dr. Nick Roussopoulos are attached as Exhibits 1021 – 1040.

## V. There Is a Reasonable Likelihood That at Least One Claim of the '936 Patent Is Unpatentable

Section V.A contains a description of each piece of relevant prior art and an explanation of the motivation to combine the relevant prior art (where applicable), and Section V.B contains claim charts identifying where each element of the challenged claims are found in the prior art.

### A. Overview of the Prior Art

#### 1. Coad

"Coad" (Ex. 1006), U.S. Patent No. 6,851,107, was filed on October 4, 2000, and issued on February 1, 2005. Coad is therefore prior art to the '936 patent under 35 U.S.C. § 102(e). Coad was applied against the claims of the '936 patent during prosecution. (*See* Ex. 1002, Office Action dated May 31, 2005.) During prosecution, the Office found that Coad disclosed numerous limitations of the claims, including "<u>a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code.</u>" (*See id.* at 3.)

Coad is directed to a "software development tool which allows a developer to simultaneously view a graphical and a textual display of source code." (Coad at Abstract.) The graphical and textual views are synchronized, such that modifications made in one view are automatically reflected in the other view. (*Id.* at col. 4, line 64 to col. 5, line 2.) To display the graphical view of the source code, the software development tool of Coad "determines the language of the source code of the chosen file, converts the source code from the language into a language-neutral representation, uses the language-neutral representation to textually display the source code of the chosen file in the language, and uses the language-neutral representation to display a graphical representation of at least a portion of the project." (*Id.* at col. 15, lines 21-28.) The graphical view of the source code includes graphical representations of flows within the source code. (*Id.* at col. 16, line 57 to col. 17, line 47.)

The software development tool of Coad further discloses a Quality Assurance (QA) module used to check source code for various types of errors (*i.e.*, coding errors, critical errors, and declaration style errors, among others). (*Id.* at col. 9, line 66 to col. 10, line 5.) If errors are detected in the source code, an error message and debugging tip for correcting the code are provided to the developer. (*Id.* at col. 14, line 64 to col. 15, line 15.)

### 2. Antis

"Antis" (Ex. 1005), U.S. Patent No. 5,572,650, was filed on June 30, 1994, and issued on November 5, 1996. Antis is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Antis is directed to a relational database analysis system that provides graphical displays for visualizing structures and relationships within relational databases. (Antis at col. 1, lines 15-18.)  To provide the visualizations, the relational database analysis system includes a number of linked "views."  For example, a highest level view of a relational database is presented in an "over view," as illustrated in Fig. 2 of Antis.  The over view displays high-level statistics and characteristics of the relational database including a listing of all relations of the database.  (*Id.* at col. 4, lines 1-19.)

Further information on the relational database is provided in additional views that are interactively linked to the over view and to each other.  (*Id.* at col. 2, lines 36-39; *see also id.* at Figs. 2-12, depicting linked views including specification view, associations view, code view, and domain view, among others.)  For example, in Fig. 8, Antis discloses a "code view" including a plurality of code boxes, where each code box represents a unit of source code of a relational database management system (RDBMS). The code view is linked to the over view, such that associations between relations of the database and the source code that accesses the relations can be visually represented. For example, if a relation is selected in the over view, the application source code that "use[s] the currently selected relation [is] displayed in the code view."  (*Id.* at col. 7, lines 36-38.)  Similarly, "[c]licking the mouse button on a code box [in the code view] highlights in the over view all relations that use the corresponding unit of code."  (*Id.* at col. 7, lines 52-54.)  The associations between the relations of the database and the source code that accesses the relations are determined via a parser layer that is based on the YACC computer program.  (*Id.* at col. 9, lines 29-41.)  The parser layer detects one of a plurality of different types of programming languages used in source code and applies appropriate rules and logic based on the detected language. (*Id.*)

The linking of the various views "makes them interactive with each other, such that changes to one view causes corresponding changes in the other actively linked views." (*Id.* at col. 9, lines 23-25.) For example, Antis discloses that a piece of source code can be retrieved via Cscope, "a well known code browser." (*Id.* at col. 9, line 46.) Any edits to the retrieved source code "will have coordinated interactive results in the other open views." (*See id.* at col. 8, line 63 to col. 9, line 1, and Fig. 12.)

### 3. Motivation to Combine Antis with Coad

A person of ordinary skill in the art (POSITA) would have been motivated to combine Antis's teachings with Coad because both Antis and Coad are directed to software development tools that provide visual representations of source code. The similar purposes and overlapping teachings that would have motivated the POSITA to combine the teachings of Antis and Coad are further reflected in the interlinked, synchronized "views" disclosed in both references. The POSITA in software development systems would have been motivated to supplement Antis's database analysis system with Coad's teaching of a Quality Assurance (QA), because this would allow for easier source code debugging and a more accurate code view display in Antis by enabling automatic detection of source code errors. (*See* Ex. 1015, ¶¶ 164-167.)

### 4. Burkwald

"Burkwald" (Ex. 1007), U.S. Patent No. 6,356,285, was filed on December 17, 1997, and issued on March 12, 2002. Burkwald is therefore prior art to the '936 patent under 35 U.S.C. § 102(e). Burkwald was applied against the claims of the '936 patent during prosecution. (*See* Ex. 1002, Office Action dated May 31, 2005, at 5-6.) The Office found that Burkwald disclosed limitations of the claims including "<u>graphical representations of data flows [that] are expandable and collapsible.</u>" (*See id.*)

Burkwald is directed to a "software visualization technique [that] allows a software application or portfolio to be visually analyzed." (Burkwald at Abstract.) The software visualization technique of Burkwald allows a user to expand or collapse a displayed graphical representation of source code flow. (*Id.* at col. 14, line 43 to col. 15, line 15, and Figs. 6-8.)

### 5. Motivation to Combine Burkwald with Antis and Coad

A POSITA would have been motivated to combine Burkwald's teachings with Antis and Coad because all three references are directed to software development tools that provide visual representations of source code, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. The POSITA would have been motivated to supplement the development tools of Antis and Coad with Burkwald's teaching of expanding and collapsing graphical representations of source code flows, because this would provide a developer with flexibility to modify the views of Antis and Coad to display information at a higher or lower level of detail. (*See* Ex. 1015, ¶ 218.)

#### 6. Access 97 Visual Basic

Access 97 Visual Basic (Ex. 1009) was published in 1997. Access 97 Visual Basic is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Access 97 Visual Basic is an instructional book that includes tutorials for using Microsoft Access 97 Visual Basic software, which is a database management system. A portion of the book entitled "Finding and Fixing Bugs in Your Code" discloses that the Microsoft Access 97 Visual Basic software highlights and displays source code in red text when the system determines there is an error. (Access 97 Visual Basic at 140.)

#### 7. Motivation to Combine Access 97 Visual Basic with Antis and Coad

A POSITA would have been motivated to combine the teachings of Access 97 Visual Basic with Antis and Coad because all three references are directed to software development tools that provide visual representations of source code, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. As another example, the POSITA in software development systems would have been motivated to supplement the development tools of Antis and Coad with the teachings of Access 97 Visual Basic regarding changing the appearance of source code determined to have errors, because this would simplify source code debugging in Antis and Coad. (*See* Ex. 1015, ¶ 242.)

### 8. Eick

"Eick" (Ex. 1008), U.S. Patent No. 5,937,064, was filed on March 3, 1997, and issued on August 10, 1999. Eick is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Eick is directed to a system for "visual analysis of a database." (Eick at col. 1, lines 6-7.) The disclosed system allows a variety of remote clients to access information from a database for the purpose of allowing the clients to perform interactive visualization of the database. (*Id.* at col. 2, lines 9-15.) Eick discloses that a security layer can be used to manage the connections between the clients and the database, where the security layer utilizes, for example, password protection or data encryption techniques. (*Id.* at col. 4, lines 19-27.)

### 9. Motivation to Combine Eick with Antis and Coad

A POSITA would have been motivated to combine Eick's teachings with Antis and Coad because all three references are directed to visual representations of source code and data structures, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. As another example, the POSITA in software development systems would have been motivated to supplement the development tools of Antis and Coad with Eick's teaching of a security layer for managing secure connections with remote computers, because this would allow the systems of Antis and Coad to be distributed to one or more remote locations without a substantially increased risk that remotely stored data could be compromised. (*See* Ex. 1015, ¶ 224.)

**10. Corda**

"Corda" (Ex. 1010), U.S. Patent No. 5,758,122, was filed on March 16, 1995, and issued on May 26, 1998. Corda is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

Corda is directed to an "immersive visual programming environment" that "permits the flow of data objects and the interaction among the data objects to be visually displayed to the user." (Corda at Abstract.) The disclosed programming environment is configured to change an appearance of a displayed data flow if an error is detected in source code: "Semantic error detection is enhanced through observation of the flow of data and the motion of the structure itself. . . . Run-time error messages in the form of sight and sound alerts are provided, with the offending component highlighted in some way." (*Id.* at col. 4, lines 41-48.)

**11. Motivation to Combine Corda with Antis and Coad**

A POSITA would have been motivated to combine Corda's teachings with Antis and Coad because all three references are directed to providing visual representations of source code, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. As another example, the POSITA in software development systems would have been motivated to supplement the development tools of Antis and Coad with the teachings of Corda regarding changing the appearance of displayed flows to indicate source code determined to have errors, because this would simplify source code debugging in Antis and Coad. (*See* Ex. 1015, ¶ 236.)

### 12. "Building Applications"

"Building Applications" (Ex. 1011) was published in 1996. "Building Applications" is therefore prior art to the '936 patent under 35 U.S.C. § 102(b).

"Building Applications" is an instructional book that includes tutorials for using Microsoft Access 97 software, which is a database management system. A portion of the book entitled "Writing and Editing Code" discloses that the Microsoft Access 97 software includes an "automatic syntax checking" feature configured to automatically correct segments of source code determined to have errors: "Visual Basic checks the syntax of that line and displays a message if it finds an error. To enable automatic syntax checking, select the Auto Syntax Check check box on the Module tab of the Options dialog box (Tools menu)." ("Building Applications" at 54.)

### 13. Motivation to Combine "Building Applications" with Antis and Coad

A POSITA would have been motivated to combine the teachings of "Building Applications" with Antis and Coad because all three references are directed to software development tools that provide visual representations of source code, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. The POSITA in software development systems would have been motivated to supplement the development tools of Antis and Coad with the teachings of “Building Applications” regarding automatically correcting segments of source code determined to have errors, because this would simplify the debugging of source code in Antis and Coad. (*See* Ex. 1015, ¶ 230.)

### 14. Oracle Primer

Oracle Primer (Ex. 1012) was published in 1999. Oracle Primer is therefore prior art to the ‘936 patent under 35 U.S.C. § 102(b).

Oracle Primer is an instructional book that includes information on the Oracle database system. Oracle Primer also includes information on the JDBC (Java Database Connectivity) application programming interface (API), which enables database access in the Java programming language. (Ex. 1012 at 187-189 and 194.)

### 15. Oracle8 Primer

Oracle8 Primer (Ex. 1013) was published in 1999. (*See* Ex. 1017, Declaration of Morris M. Jackson.) Oracle8 Primer is therefore prior art to the ‘936 patent under 35 U.S.C. § 102(b).

Oracle8 Primer is an instructional book that includes information on the Oracle database system. Oracle8 Primer discloses that SQL can be embedded in the Java, C, and C++ programming languages, thus allowing each of these programming languages to access data in a database. (Ex. 1013, Oracle8 Primer at 93, 95, 103, 108, 118, 277, 280, 281, 294, and 301.)

#### 16. Motivation to Combine Oracle Primer and Oracle8 Primer with Coad

A POSITA would have been motivated to combine Coad's teachings with Oracle Primer and Oracle8 Primer because all three references are directed to computer programming, generally, and to the Java and C++ programming languages, specifically, such that the similar purposes and overlapping teachings would have motivated the POSITA to combine the teachings of the references. Further, the POSITA would have been motivated to supplement Coad with the disclosure in the Oracle Primer and the Oracle8 Primer regarding using the Java and C++ programming languages to access data in a database, because this would enhance the utility of these programming languages by allowing them to retrieve, insert, delete, or modify data of a database. (*See* Ex. 1015, ¶ 115.)

### B. Claim 1 Is Anticipated by Coad (U.S. Patent No. 6,851,107)

Claim 1 is anticipated by Coad as indicated below.

| 1. An integrated development environment, comprising: |
|---|
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 82-84.) To the extent the preamble is considered a limitation to the claim, Coad discloses an integrated software development environment at Abstract:**<br><br>"Methods and systems consistent with the present invention provide an improved software development tool which allows a developer to simultaneously view a |

graphical and a textual display of source code. The graphical and textual views are synchronized so that a modification in one view is automatically reflected in the other view."

**Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Coad. (Ex. 1016 at 18.)**

[1.1] a document manager for retrieving source code programmed using one of a plurality of types of data manipulation languages;

**Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 85-87.) At col. 15, lines 60-64, Coad discloses a document manager for retrieving source code from a selected file:**

"As previously stated, the project comprises a plurality of files. The developer either uses the software development tool to open a file which contains existing source code, or to create a file in which the source code will be developed."

**At Abstract, Coad further discloses that the retrieved source code may be programmed using one of a plurality of types of programming languages:**

"In addition, the software development tool is designed for use with more than one programming language."

**As stated above in Section IV.C, a data manipulation language is "*a programming language used to access data in a database, such as to retrieve, insert, delete, or modify data in the database*." Coad discloses use of the software development tool with the C++ and Java programming languages. (*See, e.g.*, Coad at col. 16, lines 1-4.) Because both of these programming languages had functions that allowed them to access data in a database (*see* Ex. 1015, ¶¶ 48-54), the C++ and Java**

**programming languages of Coad disclose the recited “data manipulation languages” of claim 1.**

[1.2] an editor for displaying the retrieved source code and providing a means for a user to edit the retrieved source code;

**Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 88-89.) At col. 4, lines 54-60, Coad discloses an incremental code editor (ICE) for displaying and editing retrieved source code:**

“Although modifications made on the displays 204 and 206 may appear to modify the displays 204 and 206, in actuality all modifications are made directly to the source code 202 via an incremental code editor (ICE) 208, and the TMM 200 is used to generate the modifications in both the graphical 204 and the textual 206 views from the modifications to the source code 202.”

**Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Coad. (Ex. 1016 at 18.)**

[1.3.1] a parser layer which detects the one of the plurality of types of data manipulation languages in which the retrieved source code is programmed and

**Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 90-91.) At col. 2, lines 57-58 and col. 5, lines 50-55, Coad discloses a parser layer configured to detect the language of the retrieved source code:**

“The method comprises the steps of determining the language of the source code of the chosen file . . . .” (Coad at col. 2, lines 57-58.)

“The core 700 includes a parser 706 and an ICE 208. The parser 706 converts the source code into the language-neutral representation in the TMM, and the ICE 208 converts the text from the displays into source code.” (*Id.* at col. 5, lines 50-55.)

[1.3.2] [the parser layer] which activates rules and logic applicable to the detected one of the plurality of types of data manipulation languages; and

**Coad discloses this claim element. (*See* Ex. 1015, ¶ 92.) Coad discloses that the parser layer is configured to apply rules and logic applicable to the detected language. *See* Coad at col. 5, lines 50-55, reproduced above for element 1.3.1. *See also* col. 16, lines 4-16:**

"The software development tool then obtains a template for the current programming language, *i.e.*, a collection of generalized definitions for the particular language that can be used to build the data structure (step 904). For example, the definition of a new Java class contains a default name, *e.g.*, 'Class1,' and the default code, 'public class Class1 { }.' Such templates are well known in the art. For example, the 'Microsoft Foundation Class Library' and the 'Microsoft Word Template For Business Use Case Modeling' are examples of standard template libraries from which programmers can choose individual template classes. The software development tool uses the template to parse the source code (step 906), and create the data structure (step 908)."

***See also* Tables 1-17 at col. 6, line 30 to col. 14, line 63, disclosing rules applied by the parser layer.**

[1.4] a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code using the rules and logic applicable to the detected one of the plurality of types of data manipulation languages and activated by the parser,

**Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 93-107.) Coad discloses a viewer (*i.e.*, visualizer) for displaying graphical representations of flows in source code at col. 16, line 57 to col. 17, line 47:**

"The software development tool is collectively broken into three views of the application: the static view, the dynamic view, and the functional view. The static view is modeled using the use-case and class diagrams. A use case diagram 1200, depicted in FIG. 12, shows the relationship among actors 1202 and use cases 1204 within the system 1206. A class diagram 1300, depicted in FIG. 13 with its associated source code 1302, on the other hand, includes classes 1304, interfaces, packages and their relationships connected as a graph to each other and to their contents.

The dynamic view is modeled using the sequence, collaboration and statechart diagrams. As depicted in FIG. 14, a sequence diagram 1400 represents an interaction, which is a set of messages 1402 exchanged among objects 1404 within a collaboration to effect a desired operation or result. In a sequence diagram 1400, the vertical dimension represents time and the horizontal dimension represents different objects. A collaboration diagram 1500, depicted in FIG. 15, is also an interaction with messages 1502 exchanged among objects 1504, but it is also a collaboration, which is a set of objects 1504 related in a particular context. Contrary to sequence diagrams 1400 (FIG. 14), which emphasize the time ordering of messages along the vertical axis, collaboration diagrams 1500 (FIG. 15) emphasize the structural organization of objects.

A statechart diagram 1600 is depicted in FIG. 16. The statechart diagram 1600 includes the sequences of states 1602 that an object or interaction goes through during its life in response to stimuli, together with its responses and actions. It uses a graphic notation that shows states of an object, the events that cause a transition from one state to another, and the actions that result from the transition.

The functional view can be represented by activity diagrams 1700 and more traditional descriptive narratives such as pseudocode and minispecifications. An activity diagram 1700 is depicted in FIG. 17, and is a special case of a state diagram where most, if not all, of the states are action states 1702 and where most, if not all, of the transitions are triggered by completion of the actions in the source states. Activity diagrams 1700 are used in situations where all or most of the events represent the completion of internally generated actions.

There is also a fourth view mingled with the static view called the architectural view. This view is modeled using package, component and deployment diagrams. Package diagrams show packages of classes and the dependencies among them. Component diagrams 1800, depicted in FIG. 18, are graphical representations of a system or its component parts. Component diagrams 1800 show the dependencies among software components, including source code components, binary code components and executable components. As depicted in FIG. 19, Deployment diagrams 1900 are used to show the distribution strategy for a distributed object system. Deployment diagrams 1900 show the configuration of run-time processing elements and the software components, processes and objects that live on them."

**Figs. 11 – 17 of Coad depict aspects of the viewer for displaying graphical representations of flows in source code. For example, Fig. 16 is an example of a**

**graphical representation of flow disclosed in Coad, where a statechart diagram 1600 depicts the sequence of states 1602 that an object or interaction goes through during its life in response to stimuli.**

[1.5] wherein the editor, parser layer and visualizer cooperate such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor.

**Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 108-110.) Coad discloses that graphical and textual views are synchronized such that modifications made to one view are automatically reflected in the other view, as disclosed at col. 4, line 61 to col. 5, line 3:**

"The improved software development tool provides simultaneous round-trip engineering, *i.e.*, the graphical representation 204 is synchronized with the textual representation 206. Thus, if a change is made to the source code 202 via the graphical representation 204, the textual representation 206 is updated automatically. Similarly, if a change is made to the source code 202 via the textual representation 206, the graphical representation 204 is updated to remain synchronized. There is no repository, no batch code generation, and no risk of losing code."

Complementsoft's arguments made with respect to Coad during prosecution are inconsistent with the claim language and lack technical merit. In particular, the argument that equated "data manipulation languages" with "unstructured programming languages" is technically untrue because a person of ordinary skill in the art would understand that most data manipulation languages are in fact structured programming languages. (*See* Ex. 1015, ¶¶ 24-60.)

Further, the scope of claim 1, covering both program flows *and* data flows, is inconsistent with the argument made by Complementsoft when distinguishing claim 1 from Coad. This is because the distinction argued by Complementsoft (*i.e.*, that displaying flow in unstructured programming languages is distinguishable from displaying flow in structured programming languages) applies, if at all, only to displaying *program flows*. This distinction does not apply to data flows. (*Id.* at ¶¶ 61-68.)

For these reasons, and in light of the fact that Coad discloses a "data manipulation language" (*see* Ex. 1015, ¶¶ 48-54), Petitioner believes that the Office's rejection of claim 1 over Coad was proper and therefore proposes a similar ground of unpatentability here.

### C. Claim 1 Is Obvious Over Coad in View of Oracle Primer and Oracle8 Primer

As discussed above, Coad discloses all limitations of claim 1. To the extent it could be argued that any further disclosure may be required with respect to the "data manipulation language" feature of claim 1, Oracle Primer and Oracle8 Primer provide such further disclosure. As explained above in Section IV.C, a data manipulation language is *a programming language used to access data in a database, such as to retrieve, insert, delete, or modify data in the database*. The Oracle Primer and Oracle8 Primer references disclose that the programming languages used in the Coad reference (*i.e.*, Java and C++) included functions that allowed the programming languages to access data in a database. Therefore, Oracle Primer and Oracle8 Primer disclose that the Java and C++ programming languages of Coad are "data manipulation languages," as recited in claim 1. (*See* Ex. 1015, ¶¶ 111-114.)

For example, Oracle Primer discloses the JDBC (Java Database Connectivity) application programming interface (API), which enables database access in the Java programming language. (Ex. 1012, Oracle Primer at 187-189 and 194.) The Oracle8 Primer discloses that SQL can be embedded in the Java, C, and C++ programming languages, thus allowing each of these languages to access data in a database. (Ex. 1013, Oracle8 Primer at 93, 95, 103, 108, 118, 277, 280, 281, 294, and 301.)

Oracle Primer and Oracle8 Primer are combinable with Coad for the reasons stated in Section V.A.16. (*See also* Ex. 1015, ¶ 115.)

## D. Claims 1-3 and 5 Are Anticipated by Antis (U.S. Patent No. 5,572,650)

### 1. Claim 1

Claim 1 is anticipated by Antis as indicated below.

| 1. An integrated development environment, comprising: |
|---|
| **Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 125-127.) To the extent the preamble is considered a limitation to the claim,[3] Antis discloses a relational** |

[3] *See* MPEP § 2111.02 Effect of Preamble ("The determination of whether a preamble limits a claim is made on a case-by-case basis in light of the facts in each case").

**database analysis system including a plurality of software development features,**

**at col. 2, lines 25-29; col. 3, lines 23-30; and Abstract**:

"Therefore, there is a long felt need in the art for a method and apparatus for displaying characteristics of a database without semantic information such that explicit and implicit data structures can be readily observed to facilitate *use, development and maintenance of large databases*." (Antis at col. 2, lines 25-29, emphasis added.)

"Referring now to FIG. 1, a block diagram of an example of a relational database analysis system 101 is shown. The system 101 includes terminal 103, which provides output to and receives input from a user of the system, processor 113, which performs the actual analysis operations, memory system 115, which contains programs 117 executed by processor 113 and relations 119cI19i, each of which contains a respective set of attributes and tuples." (*Id.* at col. 3, lines 23-30.)

"A method and apparatus for visually displaying structural characteristics of a large database for visualizing areas where new developments may be added, or finding places where old developments may need to be re-structured or replaced." (*Id.* at Abstract.)

[1.1] a document manager for retrieving source code programmed using one of a plurality of types of data manipulation languages;

**Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 128-131.) At col. 9, lines 45-46, Antis discloses the use of Cscope, a "well known code browser," to retrieve source code programmed using one of a plurality of types of data manipulation languages.**

"[T]he source code references to the relations are extracted from a Cscope database (Cscope is a well known code browser)."

**Antis also discloses an "over view" display 201 of the relational database analysis system, which is described at col. 4, lines 2-3, and depicted at Fig. 2:**

"Display 201 is an over view, which is the highest view level of the large relational database."

**The over view can be used to retrieve source code via a plurality of different methods. For example, as illustrated in Fig. 2, the display 201 of the over view includes activity buttons near the top of the GUI window. The activity buttons allow different views of the relational database analysis system to be invoked, including a "code view." From the code view, the aforementioned Cscope code browser is used to retrieve source code:**

"Across the top of display 201 are a number of activity buttons which may be activated by the cursor to provide additional information about the selected relation, in this case relation RLls_lnk. The buttons are labeled: Assocs (abbreviation for Associations), Paths, Code, PRL (abbreviation for Population Rule Language, commonly referred to as the Specification View), Layout, Relations List, Domains, Entity, and Distance. *Each of these buttons, except Distance, may be used to open another window on the screen 105 (see FIG. 1) which displays another view of the relational database under analysis*." (Antis at col. 4, line 60 to col. 5, line 3, emphasis added.)

**Source code can also be retrieved from the over view by selecting one of the relations displayed in the over view display 201. As disclosed at col. 7, lines 32-35, based on the relation selected in the over view, source code that accesses the selected relation is retrieved:**

"The code view displays the application source code of the RDBMS that uses the currently selected relation, in this case relation RLls_lnk."

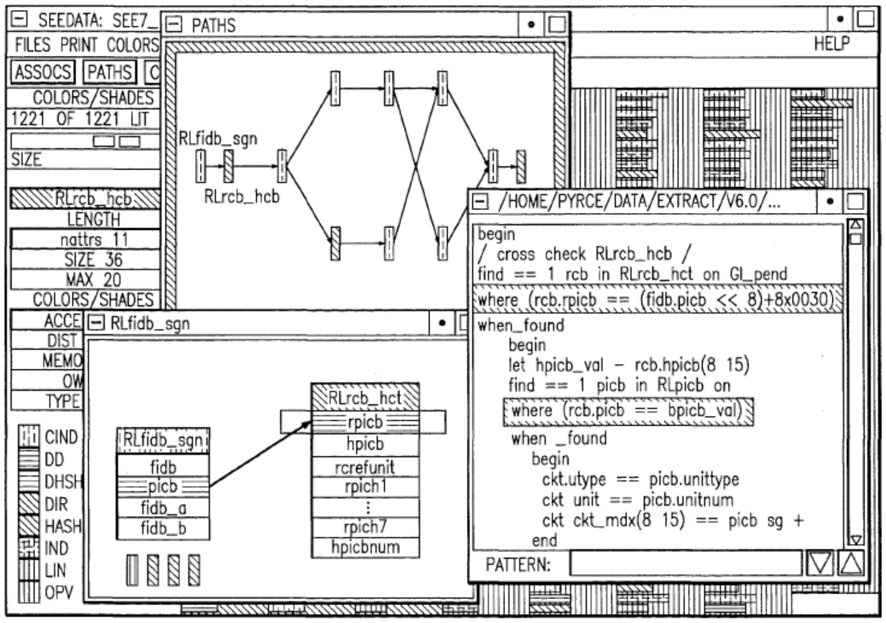
**Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Antis. (Ex. 1016 at 3.)**

[1.2] an editor for displaying the retrieved source code and providing a means for a user to edit the retrieved source code;

**Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 132-134.) Antis discloses an**

**editor for displaying and editing retrieved source code in an "expanded code view," as illustrated at the bottom right hand corner of Fig. 12 and as described at col. 8, lines 63-66:**

FIG. 12



```
begin
/ cross check RLrcb_hcb /
find == 1 rcb in RLrcb_hct on GI_pend
where (rcb.rpicb == (fidb.picb << 8)+8x0030)
when_found
   begin
   let hpicb_val - rcb.hpicb(8 15)
   find == 1 picb in RLpicb on
   where (rcb.picb == bpicb_val)
   when _found
     begin
      ckt.utype == picb.unittype
      ckt unit == picb.unitnum
      ckt ckt_mdx(8 15) == picb sg +
     end
```

"FIG. 12 shows an over view, a relations view, a path view and an expanded code view all displayed in partially overlapping windows on a display screen."

**The expanded code view is used to edit the retrieved source code. Antis discloses that changes made in any of the disclosed views (*e.g.*, edits to the source code made in the expanded code view or a definition of a new object) cause corresponding changes in the other views, at col. 9, lines 17-28:**

"The overall view is purely structural without semantic details to clutter it. If semantic details are desired, they are available in views called the specification view and the code view. If further structural details of a particular relation are desired, further details are available in the associations view, the path view, the code view and the domain view. *The linking of the views makes them interactive with each other such that changes to one view causes corresponding changes in the other actively linked views*. This not only *speeds up  analysis, but also gives the user a chance to analyze the effects of any changes, e.g. definition of a new object, on the relational database system*."  (Emphasis added.)

[1.3.1]  a parser layer which detects the one of the plurality of types of data manipulation languages in which the retrieved source code is programmed and

**Antis discloses this claim element.  (*See* Ex. 1015, ¶¶ 135-137.)  At col. 9, lines 35-41, Antis discloses a parser layer based on the YACC computer program.  The parser layer disclosed in Antis can be used for "any type of database" and is thus used to detect any of a plurality of data manipulation languages having "a YACC grammar":**

"*For any type of database,* the associations could be extracted from the database query language with an A * version (A* is a combination of pattern matching that is based on AWK, and parsing that is based on YACC which allows AWK style pattern matching to be applied to grammar productions of *a language specified with a YACC grammar) built for that language grammar*."  (Emphasis added.)

***See also id.* at col. 5, lines 4-8, disclosing applicability of the relational database analysis system to multiple different data manipulation languages:**

"Referring now to FIG. 3, a display 301 is shown on display screen 105 that is called the specification view. In the specification view the actual specification(s) of the database *in the database description language or languages that are understood by the RDBMS may be directly viewed*."  (Emphasis added.)

**Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Antis.  (Ex. 1016 at 3.)**

[1.3.2]  [the parser layer] which activates rules and logic applicable to the detected one of the plurality of types of data manipulation languages; and

**Antis discloses this claim element.  (*See* Ex. 1015, ¶ 138.)  Antis discloses that the**

**parser layer based on the YACC computer program activates rules and logic for a detected one of the plurality of types of languages. *See* Antis at col. 9, lines 35-41, reproduced above for element 1.3.1. Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Antis. (Ex. 1016 at 3.)**

[1.4] a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code using the rules and logic applicable to the detected one of the plurality of types of data manipulation languages and activated by the parser,

**Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 139-146.) Antis discloses a visualizer for displaying graphical representations of flows within the retrieved source code in a "code view," as illustrated at Fig. 8 and as described at col. 7, lines 31-45. Specifically, the code view provides a graphical display of flow in the retrieved source code by providing a visual representation of the data within the relational database (*i.e.*, the relations) that are accessed by the retrieved source code. The code view also depicts a graphical display of flow by providing a visual representation of which pieces of source code are executed for a selected relation, where the relation is selected via the over view display described above:**

"Referring now to FIG. 8, a display 801 is shown on display screen 105 that is called the code view. The code view displays the application source code of the RDBMS that uses the currently selected relation, in this case relation RLls_lnk. *The application source code may be represented as a horizontal tree representing the code hierarchy. The subset or subsets of that horizontal code tree that use the currently selected relation are displayed in the code view. The code view may be used by the user to find the application source code which references a relation and also to find where in the*

| |
|---|
| *application source code the relation is accessed. The code view is linked to the over view such that all relations accessed from a unit of application code may be easily identified*. Each unit of application code is represented 45 by a box in the code view." (Emphasis added.)<br><br>**The code view of Antis further discloses a graphical representation of flow within the retrieved code at col. 7, lines 51-53. This portion discloses that clicking on a code box (*i.e.*, an icon associated with a particular piece of source code) produces a visual representation of the data in the relational database (*i.e.*, the relations) that are accessed by the particular piece of source code:**<br><br>"Clicking the mouse button on a code box highlights in the over view all relations that use the corresponding unit of code."<br><br>**Antis further discloses a graphical representation of flows within the retrieved source code at col. 7, lines 57-61. This portion discloses that a parallel flow of execution is displayed in the code view if the retrieved source code is executed on multiple processors (*i.e.*, the code view display will be changed depending on the number of processors executing the retrieved source code, thus providing a graphical display of the flow of execution for the code):**<br><br>"For the purposes of the code view, application code executing on multiple separate processors in a distributed system is considered to be multiple separate applications, each of which may be examined as a separate code sub-tree." |
| [1.5] wherein the editor, parser layer and visualizer cooperate such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor. |

| |
|---|
| **Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 147-149.) Antis discloses that the various views (*i.e.*, the over view, code view, etc.) of the of the relational database analysis system are linked, such that changes made in one view are automatically made in the other linked views, at col. 9, lines 17-28:**<br><br>“The overall view is purely structural without semantic details to clutter it. If semantic details are desired, they are available in views called the specification view and the code view. If further structural details of a particular relation are desired, further details are available in the associations view, the path view, *the code view* and the domain view. *The linking of the views makes them interactive with each other such that changes to one view causes corresponding changes in the other actively linked views.* This not only speeds up analysis, but also gives the user a chance to analyze the effects of any changes, *e.g.* definition of a new object, on the relational database system.” (Emphasis added.) |

### 2. Claim 2

Claim 2 is anticipated by Antis as indicated below.

| 2. The integrated development environment as recited in claim 1, wherein the graphical representations of flows depict data flows. |
|---|
| **Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 150-153.) The code view of Antis discloses graphical representations of data flows within the retrieved source code, as described at col. 7, lines 31-45 (reproduced above for element 1.4 of claim 1). Specifically, the code view provides a visualization of data flows in the retrieved source by providing a visual representation of which pieces of source code access which data in the relational database.** |

### 3. Claim 3

Claim 3 is anticipated by Antis as indicated below.

| 3. The integrated development environment as recited in claim 1, wherein the graphical representations of flows depict program flows. |
|---|
| **Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 154-157.) The code view of Antis discloses graphical representations of program flows within the retrieved source code, as described at col. 7, lines 31-45 (reproduced above for element 1.4 of claim 1). Specifically, the code view provides a visualization of program flows in the retrieved source by providing a visual representation of which pieces of source code are executed for a selected relation of the database. Antis further discloses a graphical representation of program flows at col. 7, lines 57-61. This portion discloses that a parallel flow of execution (*i.e.*, program flows) is displayed in the code view if the retrieved source code is executed on multiple processors:**<br>"For the purposes of the code view, application code executing on multiple separate processors in a distributed system is considered to be multiple separate applications, each of which may be examined as a separate code sub-tree." (Antis at col. 7, lines 57-61.) |

### 4. Claim 5

Claim 5 is anticipated by Antis as indicated below.

| 5. The integrated development environment as recited in claim 1, wherein the document manager retrieves all files related to the source code to be edited. |
|---|
| **Antis discloses this claim element. (*See* Ex. 1015, ¶¶ 158-161.) At col. 7, lines 31-35, Antis discloses that the code view is used to retrieve source code related to a relation that is selected in the over view. Further, as explained above with reference to claim 1, the expanded code view of Antis depicted in Fig. 12 allows the** |

**retrieved code to be edited:**

"Referring now to FIG. 8, a display 801 is shown on display screen 105 that is called the code view. The code view displays the application source code of the RDBMS that uses the currently selected relation, in this case relation RLls_lnk."

**Further, at col. 7, lines 39-44, Antis discloses that all pieces of source code related to the selected relation are retrieved:**

"The code view may be used by the user to find the 40 application source code which references a relation and also to find where in the application source code the relation is accessed. The code view is linked to the over view such that all relations accessed from a unit of application code may be easily identified."

**Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Antis. (Ex. 1016 at 5.)**

### E. Claims 1-3, 5, 6, 8, 10-12, 15, and 16 Are Obvious Over Antis in View of Coad

#### 1. Claim 1

As discussed above, Antis discloses all limitations of claim 1. To the extent it could be argued that any further disclosure may be required with respect to limitations 1.2, 1.4, and 1.5, Coad provides such further disclosure as follows.

| [1.2] an editor for displaying the retrieved source code and providing a means for a user to edit the retrieved source code; |
|---|
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 88-89.) Coad discloses an incremental code editor (ICE) for displaying and editing retrieved source code. *See* Coad at col. 4, lines 54-60, reproduced above for element 1.2 of claim 1.** |

CLI-2084525

| |
|---|
| **Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Coad. (Ex. 1016 at 18.)** |
| [1.4] a visualizer dynamically linked to the editor for displaying graphical representations of flows within the retrieved source code using the rules and logic applicable to the detected one of the plurality of types of data manipulation languages and activated by the parser, |
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 93-107.) Coad discloses a viewer (*i.e.*, visualizer) for displaying graphical representations of flows in source code at col. 16, line 57 to col. 17, line 47, reproduced above for element 1.4 of claim 1. Further, Figs. 11 – 17 of Coad depict aspects of the viewer for displaying graphical representations of flows in source code. For example, Fig. 16 is an example of a graphical representation of flow disclosed in Coad, where a statechart diagram 1600 depicts the sequence of states 1602 that an object or interaction goes through during its life in response to stimuli.** |
| [1.5] wherein the editor, parser layer and visualizer cooperate such that edits made to the source code using the editor are automatically reflected in the graphical representations of flows displayed by the visualizer and edits made to the graphical representations of flows in the visualizer are automatically reflected in the source code displayed by the editor. |
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 108-110.) Coad discloses a software development tool that allows a developer to simultaneously view a graphical representation and a text representation of source code. (*See, e.g.* Coad at Abstract and Fig. 2.) The graphical and textual views are synchronized such that modifications made to one view are automatically reflected in the other view,** |

| |
|---|
| **as disclosed at col. 4, line 61 to col. 5, line 3 (reproduced above for element 1.5 of claim 1).** |

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) The proposed ground of unpatentability based on Antis and Coad is included because Coad provides additional details on the visualization of flow that are not provided in Antis. The proposed ground of unpatentability is further included because the Office previously found that Coad disclosed these claim limitations, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 3, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

### 2. Claim 2

As discussed above, Antis discloses all limitations of claim 2. To the extent it could be argued that any further disclosure may be required with respect to the limitations of claim 2, Coad provides such further disclosure as follows.

| 2. The integrated development environment as recited in claim 1, wherein the graphical representations of flows depict data flows. |
|---|
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 168-171.) Coad discloses a visualizer for displaying graphical representations of data flows at col. 16, line 57 to col. 17, line 47, reproduced above for element 1.4 of claim 1.** |

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) The proposed ground of unpatentability based on Antis and Coad is included because Coad provides additional details on the visualization of data flow that are not provided in Antis. The proposed ground of unpatentability is further included because the Office previously found that Coad disclosed this claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

### 3. Claim 3

As discussed above, Antis discloses all limitations of claim 3. To the extent it could be argued that any further disclosure may be required with respect to the limitations of claim 3, Coad provides such further disclosure as follows.

| 3. The integrated development environment as recited in claim 1, wherein the graphical representations of flows depict program flows. |
|---|
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 172-175.) Coad discloses a visualizer for displaying graphical representations of program flows at col. 16, line 57 to col. 17, line 47, reproduced above for element 1.4 of claim 1.** |

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) The proposed ground of unpatentability based on Antis and Coad is included because Coad provides additional details on the visualization of program flow that are not provided in Antis. The proposed ground of unpatentability is further included because the Office previously found that Coad disclosed this claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

### 4. Claim 5

Claim 5 depends from claim 1, which, as discussed above, is obvious over Antis in view of Coad. As explained in Section V.D.4, Antis discloses the additional limitations in claim 5, which accordingly is obvious over Antis in view of Coad. (*See* Ex. 1015, ¶ 176.)

### 5. Claim 6

Claim 6 depends from claim 1, which, as discussed above, is (1) anticipated by Antis and (2) obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 6, which accordingly is obvious over Antis in view of Coad.

| 6. The integrated development environment as recited in claim 1, wherein the document manager comprises a site manager and a connectivity layer for retrieving source code from one or more remote computers. |
|---|
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 177-180.) At col. 5, lines 31-49, Coad discloses that an incremental code editor (ICE) (*see, e.g.*, Fig. 2 and col. 4, line 58) includes site manager and connectivity layer functionality for retrieving source code from remote computers over the Internet:** “FIG. 6 depicts a data processing system 600 suitable for practicing methods and systems consistent with the present invention. Data processing system 600 comprises a memory 602, a secondary storage device 604, an I/O device 606, and a processor 608. Memory 602 includes the improved software development tool 610. The software development tool 610 is used to develop a software project 612, and create the TMM |

200 in the memory 602. The project 612 is stored in the secondary storage device 604 of the data processing system 600. One skilled in the art will recognize that data processing system 600 may contain additional or different components.

Although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks or CD-ROM; a carrier wave from a network, such as Internet; or other forms of RAM or ROM either currently known or later developed."

**Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Coad. (Ex. 1016 at 20-21.)**

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation, and Complementsoft did not dispute this finding. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

### 6. Claim 8

Claim 8 depends from claim 1, which, as discussed above, is (1) anticipated by Antis and (2) obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 8, which accordingly is obvious over Antis in view of Coad.

| |
|---|
| 8. The integrated development environment as recited in claim 1, wherein the editor comprises a template manager for allowing preprogrammed segment of source code to be placed within the source code being edited. |
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 181-185.) At col. 16, lines 4-9, Coad discloses "a collection of generalized definitions for the particular language** |

| |
|---|
| **that can be used to build [a] data structure." The generalized definitions include "default code" that can be placed within source code being edited.**<br><br>"The software development tool then obtains a template for the current programming language, *i.e.*, a collection of generalized definitions for the particular language that can be used to build the data structure (step 904). For example, the definition of a new Java class contains a default name, *e.g.*, 'Class1,' and the default code, 'public class Class1 { }.' Such templates are well known in the art." |

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.)

### 7. Claim 10

Claim 10 depends from claim 8, which, as discussed above, is obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 10, which accordingly is obvious over Antis in view of Coad.

| 10. The integrated development environment as recited in claim 8, wherein the template manager is adapted to automatically generate segments of the source code. |
|---|
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 186-190.) Coad discloses that the template manager is configured to automatically generate segments of source code. *See* Coad at col. 16, lines 4-9, reproduced above for claim 8.** |

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.)

### 8. Claim 11

Claim 11 depends from claim 1, which, as discussed above, is (1) anticipated by Antis and (2) obvious over Antis in view of Coad. Coad and Antis disclose the additional limitations in claim 11, which accordingly is obvious over Antis in view of Coad.

| 11. The integrated development environment as recited in claim 1, further comprising a means for allowing the source code to be executed both locally and remotely. |
|---|
| **Coad and Antis disclose this claim element. (*See* Ex. 1015, ¶¶ 191-195.) Coad discloses that the ICE editor includes functionality for allowing retrieved source to be executed locally (*i.e.*, from a local memory) or remotely (*i.e.*, via the Internet). *See* Coad at col. 5, lines 31-49, reproduced above for claim 6. Further, at col. 7, lines 55-61, Antis discloses a distributed system with remote execution of source code:**<br><br>"If there are multiple applications that use the same relation, the result will be multiple code sub-trees in the code view, as sub-trees 805 and 809. For the purposes of the code view, application code executing on multiple separate processors in a distributed system is considered to be multiple separate applications, each of which may be examined as a separate code sub-tree." |

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation. (Ex. 1002, Office Action dated May 31, 2005, at 4.)

### 9. Claim 12

Claim 12 depends from claim 11, which, as discussed above, is obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 12, which accordingly is obvious over Antis in view of Coad.

| 12. The integrated development environment as recited in claim 11, wherein the parser layer further examines error log files generated by the means for allowing the source |
|---|

| |
|---|
| code to be executed to determine segments of the source code determined to include errors. |
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 196-201.) At col. 9, line 66 to col. 10, line 5, Coad discloses a quality assurance (QA) module for detecting segments of source code containing errors. If the QA module determines that the source code "does not conform, an error message is provided to the developer" (*see* Coad at col. 14, lines 64-65):** "The QA module also provides audits, *i.e.*, the module checks for conformance to predefined or user-defined styles. The types of audits provided by the module include coding style, critical errors, declaration style, documentation, naming style, performance, possible errors and superfluous content. Examples of these audits with their respective definitions are identified in Tables 10-17 below." ***See also*** **Coad at col. 10, line 8 to col. 14, line 63, disclosing Tables 10-17, including examples of coding style audits, critical error audits, and declaration style audits, among others.** |

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation. (Ex. 1002, Office Action dated May 31, 2005, at 4.)

#### 10. Claim 15

Claim 15 depends from claim 12, which, as discussed above, is obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 15, which accordingly is obvious over Antis in view of Coad.

| |
|---|
| 15. The integrated development environment as recited in claim 12, further comprising |

| a message manager cooperating with the parser layer for displaying debugging hints as a function of the source code segments determined to have errors. |
|---|
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 202-208.) Coad discloses a quality assurance (QA) module for determining segments of source code including errors. *See* Coad at col. 9, line 66 to col. 10, line 5, reproduced above for claim 12. At col. 14, line 64 to col. 15, line 16, Coad further discloses that if errors in the source code are detected by the QA module, an error message is provided to the developer:** <br><br> "If the QA module determines that the source code does not conform, an error message is provided to the developer. For example, as depicted in FIG. 8A, the software development tool checks for a variety of coding styles 800. If the software development tool were to check for 'Access Of Static Members Through Objects' 802, it would verify whether static members are referenced through class names rather than through objects 804. Further, as depicted in FIG. 8B, if the software development tool were to check for 'Complex Assignment' 806, the software development tool would check for the occurrence of multiple assignments and assignments to variables within the same expression to avoid complex assignments since these decrease program readability 808. An example of source code having a complex assignment 810 and source code having a non-complex assignment 812 are depicted in FIGS. 8B and 8C, respectively. The QA module of the software development tool scans the source code for other syntax errors well known in the art, as described above, and provides an error message if any such errors are detected." <br><br> **At Fig. 8B, Coad discloses a message manager for displaying debugging hints for a variety of errors. As illustrated in Fig. 8B, the errors include, for example, "Complex Assignment" and "Don't Use the Negation Operator Frequently" errors. An example debugging hint is illustrated in Fig. 8B ("Tip: Break statement into several ones.").** |

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

### 11. Claim 16

Claim 16 depends from claim 15, which, as discussed above, is obvious over Antis in view of Coad. Coad discloses the additional limitations in claim 16, which accordingly is obvious over Antis in view of Coad.

| 16. The integrated development environment as recited in claim 15, wherein the message manager allows a user to edit and maintain debugging hints for a variety of different errors. |
|---|
| **Coad discloses this claim element. (*See* Ex. 1015, ¶¶ 209-213.) At Fig. 8B, Coad discloses a message manager that maintains debugging tips for a variety of errors. As illustrated in Fig. 8B, the errors include, for example, "Complex Assignment" and "Don't Use the Negation Operator Frequently" errors. Corresponding debugging tips are maintained for each of the errors.** |

Coad is combinable with Antis for the reasons stated in Section V.A.3. (*See also* Ex. 1015, ¶¶ 164-167.) During prosecution, the Office found that Coad disclosed this claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

## F. Claim 4 Is Obvious Over Antis in View of Coad and Burkwald (U.S. Patent No. 6,356,285)

Claim 4 depends from claim 1, which, as discussed above, is obvious over Antis in view of Coad. Burkwald discloses the additional limitations in claim 4, which accordingly is obvious over Antis in view of Coad and Burkwald.

| 4. The integrated development environment as recited in claim 1, wherein the graphical representations of data flows are expandable and collapsible. |
|---|
| **Burkwald discloses this claim element. (*See* Ex. 1015, ¶¶ 214-217.) At col. 14, line 43 to col. 15, line 4, Burkwald discloses a software visualization technique that allows a user to expand or collapse a displayed graphical representation of source code:**<br><br>"In FIGS. 6-8, each bar 612 of the bar chart 610 represents one of the subsystems of the large systems 110 or one of the small systems 120 of the exemplary legacy software application. As shown in the bar chart 610 of FIGS. 6-8, the height of each bar 612 is proportional to the numbers of the programs 114 in each of the subsystems 116, sorted in decreasing order. For each program 114 in the legacy software application, the collapsed values list 620 shows both the McCabe complexity metric 622 and the McClure complexity metric 624, the total number of lines 626, and the total number of affected lines 628. It should be appreciated that, while FIGS. 6-8 show the collapsed values list 620 using the above-outlined metrics and statistical values for the software subsystem, the collapsed values list 620 can be used to show collapsed values for any metric and/or any statistic for which values have been obtained. That is, the analyst can interactively change the displayed metrics and statistics to any metric and/or statistic for which the analyst has generated data for the software application being displayed.<br><br>By manipulating the zoom bar 629 of the values list 620, the user can expand or collapse the values list 620 shown in FIG. 6 to control the height consumed by each line of data. In the views 600-800 shown in FIGS. 6-8, the values list 620 has been |

|  |
|---|
| collapsed so that each line of data is shown as a one-pixel high row. Furthermore, the rows have been sorted in decreasing order according to the number of affected lines 628. The length of each row encodes the value of the corresponding metric." |

Burkwald is combinable with Antis and Coad for the reasons stated in Section V.A.5. (*See also* Ex. 1015, ¶ 218.) During prosecution, the Office found that Burkwald disclosed this claim limitation, and this finding was not disputed by Complementsoft. (Ex. 1002, Office Action dated May 31, 2005, at 4-6, and Ex. 1003, Amendment dated August 4, 2005, at 9-12.)

### G. Claim 7 Is Obvious Over Antis in view of Coad and Eick (U.S. Patent No. 5,937,064)

Claim 7 depends from claim 6, which, as discussed above, is obvious over Antis in view of Coad. Eick discloses the additional limitations in claim 7, which accordingly is obvious over Antis in view of Coad and Eick.

| 7. The integrated development environment as recited in claim 6, wherein the document manager comprises a security layer for managing secure connections with the one or more remote computers. |
|---|
| **Eick discloses this claim element. (*See* Ex. 1015, ¶¶ 219-223.) Eick discloses a "system and method for visual analysis of a database." (Eick at Abstract.) The disclosed system and method allow a variety of remote clients to access information from a database for the purpose of allowing the clients to perform interactive visualization of the database. At col. 4, lines 19-27, Eick discloses a security layer for managing secure connections with the remote clients:**<br><br>"Two approaches may be used to permit only authorized access to the database or authorization to change the database. One approach from the web server side would be |

| |
|---|
| to pass data or access to the data base to the client upon showing the appropriate password. The other would be to have the CGI 22 scripts paths translated to pass encrypted data and make the data available and then pass the data to the applet which would then decide what data to show the client in view of establishing the appropriate security."<br><br>***See also id.* at claims 7 and 8:**<br><br>"7. The method of claim 6 in which said live document program includes a security subprogram which prevents display of at least one selected part of the data unless preselected security requirements are satisfied at the at least one computer terminal."<br><br>"8. The method of claim 7 including the step of selectively decoding encrypted data only if the preselected security requirements are satisfied."<br><br>**Additionally, in documents filed in the related litigation, Complementsoft has conceded that this claim element is disclosed in Eick. (Ex. 1016 at 6.)** |

Eick is combinable with Antis and Coad for the reasons stated in Section V.A.9. (*See also* Ex. 1015, ¶ 224.)

## H. Claim 9 Is Obvious Over Antis in view of Coad and "Building Applications"

Claim 9 depends from claim 8, which, as discussed above, is obvious over Antis in view of Coad. "Building Applications" discloses the additional limitations in claim 9, which accordingly is obvious over Antis in view of Coad and "Building Applications."

| 9. The integrated development environment as recited in claim 8, wherein the template manager is adapted to automatically correct segments of the source code. |
|---|
| **"Building Applications" discloses this claim element. (*See* Ex. 1015, ¶¶ 225-229.)** |

**At page 54, "Building Applications" discloses that "automatic syntax checking" may be enabled, which allows for syntax errors to be detected and corresponding error messages to be displayed:**

"Automatic syntax checking – As you move the insertion point off a line, Visual Basic checks the syntax of that line and displays a message if it finds an error. To enable automatic syntax checking, select the Auto Syntax Check check box on the Module tab of the Options dialog box (Tools menu)."

**Further, at pages 52-53, "automatic statement building" is used to assist the user in writing source code by providing pre-written segments of source code that can be selected by the user. Thus, the automatic syntax checking informs the user of errors in the source code, and the automatic statement building provides segments of source code to the user, such that the errors can be automatically corrected:**

"**Automatic statement building**. When you type certain Visual Basic elements, Microsoft Access automatically tries to assist you in writing code by displaying a drop-down list of appropriate choices for the code element you've typed. For example, if you type an object variable, and follow it with a period to indicate that you intend to enter a method or property, Microsoft Access automatically displays a list of the methods and properties that apply to the object. . . . To complete the statement you're typing, you can either click an item in the list or continue typing your code. If you continue typing code, the list displays the closest match to what you've typed. To enter the selected item in the list at any time, press TAB. To make the list disappear, press ESC."

"Building Applications" is combinable with Antis and Coad for the reasons stated in Section V.A.13. (*See also* Ex. 1015, ¶ 230.)

## I. Claim 13 Is Obvious Over Antis in view of Coad and Corda (U.S. Patent No. 5,758,122)

Claim 13 depends from claim 12, which, as discussed above, is obvious over Antis in view of Coad. Corda discloses the additional limitations in claim 13, which accordingly is obvious over Antis in view of Coad and Corda.

| 13. The integrated development environment as recited in claim 12, wherein the visualizer cooperates with the parser layer to change the appearance of displayed flows as a function of the source code segments determined to have errors. |
|---|
| **Corda discloses this claim element. (*See* Ex. 1015, ¶¶ 231-235.) Corda discloses an "immersive visual programming environment" that "permits the flow of data objects and the interaction among the data objects to be visually displayed to the user." (Corda at Abstract.) Corda further discloses that if an error is detected in source code, the programming environment changes the appearance of the displayed flow:** <br><br> "Once the programmer is satisfied that the program as constructed is complete, compilation provides visual and/or audio clues as to the nature and location of compiler errors." (Corda at col. 4, lines 30-32.) <br><br> "Semantic error detection is enhanced through observation of the flow of data and the motion of the structure itself. In this way, the virtual programming environment of the present invention supports verification and validation. Run-time error messages in the form of sight and sound alerts are provided, with the offending component highlighted in some way." (*Id.* at col. 4, lines 41-48.) <br><br> "A method for programming . . . , said method comprising the steps of . . . compiling the computer program while providing visual and audio clues to a user in response to a detected compiler error, said clues indicating a location and nature of the detected compiler error . . . ." (*Id.* at col. 13, lines 13-22.) |

Corda is combinable with Antis and Coad for the reasons stated in Section V.A.11. (*See also* Ex. 1015, ¶ 236.)

### J. Claim 14 Is Obvious Over Antis in view of Coad and Access 97 Visual Basic

Claim 14 depends from claim 12, which, as discussed above, is obvious over Antis in view of Coad. Access 97 Visual Basic discloses the additional limitations in claim 14, which accordingly is obvious over Antis in view of Coad and Access 97 Visual Basic.

| 14. The integrated development environment as recited in claim 12, wherein the editor cooperates with the parser layer to change the appearance of portions of the displayed source code as a function of the software segments determined to have errors. |
|---|
| **Access 97 Visual Basic discloses this claim element. (*See* Ex. 1015, ¶¶ 237-241.) At page 140, Access 97 Visual Basic discloses that code determined to have an errors is highlighted or shown in red text:** “Now that you’ve cleared away that message, you’ll notice that the ‘<’ symbol is highlighted – this is the point in the line that Microsoft Access doesn’t like – and the line is still shown in red text.” |

Access 97 Visual Basic is combinable with Antis and Coad for the reasons stated in Section V.A.7. (*See* Ex. 1015, ¶ 242.)

### K. Dependent Claims 2-16 Are Unpatentable Over Each Reference and Combination of References Asserted for Claim 1 in View of the Prior Art

To the extent that they are not explicitly enumerated above, Petitioner hereby asserts grounds of unpatentability for dependent claims 2-16 that are premised upon the respective prior art references asserted for those dependent claims in view of the following ground of unpatentability for independent claim 1 asserted above: Claim 1 is obvious over Coad  in view of Oracle Primer and Oracle8 Primer.

(*See* Ex. 1015, ¶¶ 243-244.)

## VI. Mandatory Notices Pursuant to 37 C.F.R. § 42.8(a)(1)

Pursuant to 37 C.F.R. § 42.8(a)(1), the mandatory notices identified in 37 C.F.R. § 42.8(b) are provided below as part of this Petition.

### A. C.F.R. § 42.8(b)(a): Real Party-In-Interest

SAS is the real party-in-interest for Petitioner.

### B. C.F.R. § 42.8(b)(2): Related Matters

The ‘936 patent is currently the subject of a patent infringement lawsuit brought by the assignee of the ‘936 patent, Complementsoft LLC, against SAS, captioned *ComplementSoft, LLC v. SAS Institute Inc.*, Docket No. 1:12-cv-07372 (N.D. Ill. Sept 14, 2012). This judicial matter may affect, or be affected by, decisions made in this proceeding.

### C. C.F.R. § 42.8(b)(3) and (4): Lead and Back-up Counsel and Service Information

SAS provides the following designation of counsel:

| Lead Counsel | Back-up Counsel |
|---|---|
| David B. Cochran<br>Reg. No. 39,142<br>JONES DAY<br>901 Lakeside Avenue | John V. Biernacki<br>Reg. No. 40,511<br>JONES DAY<br>901 Lakeside Avenue |

|  |  |
|---|---|
| Cleveland, Ohio 44114<br>(216) 586-7029<br>dcochran@jonesday.com | Cleveland, Ohio 44114<br>(216) 586-7747<br>jvbiernacki@jonesday.com<br><br>John A. Marlott<br>Reg. No. 37,031<br>JONES DAY<br>77 West Wacker<br>Suite 3500<br>Chicago, Illinois 60601-1692<br>(312) 269-4236<br>jamarlott@jonesday.com |

Pursuant to 37 C.F.R. § 42.10(b), a Power of Attorney accompanies this Petition. Please address all correspondence to lead and back-up counsel at the address above. SAS also consents to electronic service by email at the email addresses listed above.

## VII. Conclusion

For at least the reasons set forth above, Petitioner requests *inter partes* review of the '936 patent because it is more likely than not that at least one of the claims challenged in this Petition is unpatentable. It is therefore respectfully submitted that this Petition be granted and claims 1-16 of the '936 patent be judged invalid. Counsel for the Petitioner may be contacted at the below listed telephone number. As identified in the attached Certificate of Service and in accordance with §§ 1.33(c), 42.205, and 42.300, a copy of the present Request, in its entirety, is being served on the patent owner at the correspondence address of record for the subject patent as reflected in the publicly-available records of the United States Patent and Trademark Office as designated in the Office's Patent Application Information Retrieval system.

The Director is hereby authorized to charge any deficiency in the fees filed, asserted to be filed or which should have been filed herewith (or with any paper hereafter filed in this proceeding by this firm) to Jones Day Deposit Account No. 50-1432, ref: 343355-610009.

Respectfully submitted,

Date: March 29, 2013

/John V. Biernacki/
John V. Biernacki
Registration No. 40,511
JONES DAY
North Point, 901 Lakeside Avenue
Cleveland, Ohio 44114-1190

Direct No. (216) 586-7747

## CERTIFICATE OF SERVICE

The undersigned hereby certifies that a copy of the foregoing Petition for *Inter Partes* Review of U.S. Patent No. 7,110,936, along with all exhibits supporting and filed with the Petition, were served on March 29, 2013, via Express Mail delivery directed to the attorney of record for the patent at the following address:

Barry Horwitz
Greenberg Traurig, LLP
77 West Wacker Drive
Suite 3100
Chicago, IL 60601-1732

Date: March 29, 2013

/John V. Biernacki/
John V. Biernacki
Registration No. 40,511
JONES DAY
North Point, 901 Lakeside Avenue
Cleveland, Ohio 44114-1190

## EXHIBIT LIST

| EXHIBIT NO. | TITLE |
|---|---|
| 1001 | U.S. Patent No. 7,110,936 (“the ‘936 Patent”) |
| 1002 | Excerpt from the file history of the ‘936 Patent: May 31, 2005 Office Action |
| 1003 | Excerpt from the file history of the ‘936 Patent: August 4, 2005 Response to Office Action |
| 1004 | Excerpt from the file history of the ‘936 Patent: August 1, 2006 Supplemental Notice of Allowability |
| 1005 | U.S. Patent No. 5,572,650 (“Antis”) |
| 1006 | U.S. Patent No. 6,851,107 (“Coad”) |
| 1007 | U.S. Patent No. 6,356,285 (“Burkwald”) |
| 1008 | U.S. Patent No. 5,937,064 (“Eick”) |
| 1009 | “Microsoft Access 97 Visual Basic Step by Step” (“Access 97 Visual Basic”) |
| 1010 | U.S. Patent No. 5,758,122 (“Corda”) |
| 1011 | “Building Applications with Microsoft Access 97” (“Building Applications”) |

| | |
|---|---|
| 1012 | Oracle Programming – A Primer (“Oracle Primer”) |
| 1013 | Oracle8 Programming: A Primer (“Oracle8 Primer”) |
| 1015 | Declaration of Dr. Nick Roussopoulos Under 37 C.F.R. § 1.68 in Support of Petition for *Inter Partes* Review of U.S. Patent No. 7,110,936 |
| 1016 | Complementsoft’s LPR 2.5 Initial Response to SAS Institute’s Invalidity Contentions, *ComplementSoft, LLC v. SAS Institute Inc.*, Docket No. 1:12-cv-07372 (N.D. Ill. Sept 14, 2012) |
| 1017 | Declaration of Morris M. Jackson Under 37 C.F.R. § 1.68 |
| 1018 | Pearson Customer Service Website |
| 1019 | Outgoing Email Message from Morris M. Jackson |
| 1020 | Email Message Received by Morris M. Jackson |
| 1021 | “Advanced System Editor, Enhanced P-System Editor,” Joel Pitt, Infoworld, September 26, 1983 |
| 1022 | “Encyclopedia of Computer Science, Fourth Edition,” Anthony Ralston et al. eds., Nature Publishing Group 2000 |
| 1023 | “IBM Dictionary of Computing,” George McDaniel ed., McGraw-Hill 1994 |
| 1024 | “The New IEEE Standard Dictionary of Electrical and Electronics Terms, Fifth Edition,” Christopher J. Booth ed., The Institute of Electrical and Electronics Engineers, Inc. 1993 |

| | |
|---|---|
| 1025 | "DB2 Update," July 1999 |
| 1026 | "dBase is Loaded," Jan L. Harrington, MacUser, February 1, 1988 |
| 1027 | "FoxPro 2.5 for Macintosh," Jeffrey Sullivan, MacUser, June 1, 1994 |
| 1028 | "McGraw-Hill Dictionary of Scientific and Technical Terms," Fifth Edition, McGraw-Hill, Inc., New York, 1994 |
| 1029 | "Use Cases in Object-Oriented Software Development," Mehmet Aksit, et al., AMIDST, February 5, 1999 |
| 1030 | "Conformance Testing of Object-Oriented Components Specified by State/Transition Classes," Leonard Gallagher, National Institute of Standards and Technology, Information Technology Laboratory, May 19, 1999 |
| 1031 | "Flowcharts, State Transition Tables and State Transition Diagrams," Herbert J. Bernstein, 1999 |
| 1032 | "An ASM Semantics for UML Activity Diagrams," Egon Borger et al., Springer-Verlag Berlin Heidelberg, June 2000 |
| 1033 | "Modeling Systems With UML," A Popkin Software White Paper, 1998 |
| 1034 | "UML Notation Guide, Version 1.1," September 1, 1997 |
| 1035 | "Mole – A Java Based Mobile Agent System," Markus Straßer et al., Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems, 1996 |

| | |
|---|---|
| 1036 | “Saving Portable Computer Battery Power through Remote Process Execution,” Alexey Rudenko et al., Mobile Computing and Communications Review, Volume 2, Number 1, January 1998 |
| 1037 | “Better Template Error Messages,” Andrei Alexandrescu, March 1999, available at http://erdani.com/publications/better_template_error_messages.html |
| 1038 | “CAP: An Automated Self-Assessment Tool to Check Pascal Programs for Syntax, Logic, and Style Errors,” SIGCSE '95 Proceedings of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education, 1995 |
| 1039 | “Automatic Correction of Syntax Errors in Programming Languages,” Jean-Pierre Levy, Cornell University, Thesis, December 1971 |
| 1040 | “IEEE 100, The Authoritative Dictionary of IEEE Standards Terms,” Seventh Edition, IEEE Press, 2000 |