(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2017-519300
(P2017-519300A)
(43)公表日　平成29年7月13日(2017.7.13)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 11/36	(2006.01)	G06F	11/36	104	5B042
G06F 21/57	(2013.01)	G06F	21/57	370	5B376
G06F 9/44	(2006.01)	G06F	9/06	620A	

審査請求 未請求　予備審査請求 未請求　（全 34 頁）

(21)出願番号　特願2016-572723 (P2016-572723)
(86)(22)出願日　平成27年6月10日 (2015.6.10)
(85)翻訳文提出日　平成29年2月10日 (2017.2.10)
(86)国際出願番号　PCT/US2015/035131
(87)国際公開番号　WO2015/191731
(87)国際公開日　平成27年12月17日 (2015.12.17)
(31)優先権主張番号　62/012,127
(32)優先日　平成26年6月13日 (2014.6.13)
(33)優先権主張国　米国 (US)

(71)出願人　591044474
ザ・チャールズ・スターク・ドレイパー・ラボラトリー・インコーポレイテッド
アメリカ合衆国　０２１３９　マサチューセッツ州、ケンブリッジ、テクノロジー・スクエア　５５５
(74)代理人　100087941
弁理士　杉本　修司
(74)代理人　100086793
弁理士　野田　雅士
(74)代理人　100112829
弁理士　堤　健郎
(74)代理人　100144082
弁理士　林田　久美子

最終頁に続く

(54)【発明の名称】ソフトウェアアナリティクスのためのシステム及び方法

(57)【要約】
【課題】ソフトウェア開発、保守、および修復ライフサイクルにおける重要な側面を自動化する、大量のソフトウェアファイルを活用することが可能なシステム及び方法を提供する。
【解決手段】デザインパターンを特定する方法は、複数のファイルのそれぞれについての複数のアーチファクトを有するデータベースにアクセスする過程と、複数のファイルのうちの第１のファイルについての、複数のアーチファクトのうちの少なくとも１つに基づいて、デザインパターンを自動的に特定する過程と、を備える。
【選択図】図７

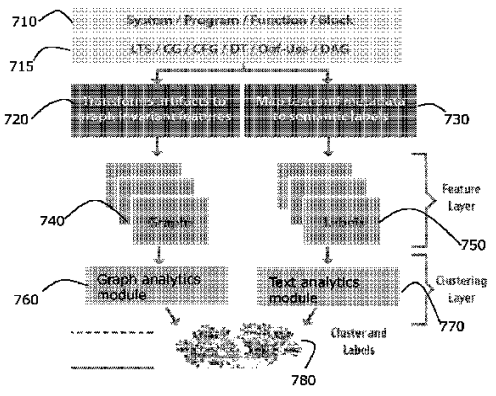

FIG. 7

【特許請求の範囲】
【請求項１】
デザインパターンを特定する方法であって、
複数のファイルのそれぞれについての複数のアーチファクトを有するデータベースにアクセスする過程と、
前記複数のファイルのうちの第１のファイルについての、前記複数のアーチファクトのうちの少なくとも１つに基づいて、デザインパターンを自動的に特定する過程と、
を備える、方法。
【請求項２】
請求項１に記載の方法において、前記デザインパターンが、前記第１のファイル内のものである、方法。
【請求項３】
請求項１に記載の方法において、前記デザインパターンを自動的に特定する過程が、前記デザインパターンの前記特定を、前記複数のファイルのうちの第２のファイルについての、前記複数のアーチファクトのうちの少なくとも１つにも基づかせることを含み、前記第１のファイルおよび前記第２のファイルが、いずれも同じプロジェクトに属する、方法。
【請求項４】
請求項３に記載の方法において、前記デザインパターンを自動的に特定する過程が、前記第１のファイルについての、前記複数のアーチファクトのうちの前記少なくとも１つと、前記第２のファイルについての、前記複数のアーチファクトのうちの前記少なくとも１つとを、前記デザインパターンを表す予め特定されたパターンに照合することを含む、方法。
【請求項５】
請求項４に記載の方法において、前記デザインパターンが、前記第１のファイルと前記第２のファイルとの間のインターフェースに関するものである、方法。
【請求項６】
請求項１に記載の方法において、前記デザインパターンが、欠陥または修復である、方法。
【請求項７】
請求項１に記載の方法において、前記デザインパターンが、機能または付加拡張機能である、方法。
【請求項８】
請求項１に記載の方法において、前記デザインパターンが、予め特定されたプログラム断片である、方法。
【請求項９】
請求項１に記載の方法において、前記複数のアーチファクトのうちの前記少なくとも１つに基づいて、前記デザインパターンを自動的に特定する過程が、前記複数のアーチファクトのうちの前記少なくとも１つにおいて、欠陥または修復を表す文字列を探し出すことを含む、方法。
【請求項１０】
請求項９に記載の方法において、前記複数のアーチファクトのうちの前記少なくとも１つが、開発中アーチファクトである、方法。
【請求項１１】
請求項１に記載の方法において、前記複数のアーチファクトのうちの前記少なくとも１つに基づいて、前記デザインパターンを自動的に特定する過程が、前記複数のアーチファクトのうちの前記少なくとも１つにおいて、機能または付加拡張機能を表す文字列を探し出すことを含む、方法。
【請求項１２】
請求項１１に記載の方法において、前記複数のアーチファクトのうちの前記少なくとも

１つが、開発中アーチファクトである、方法。
【請求項１３】
請求項１に記載の方法において、前記複数のアーチファクトのうちの前記少なくとも１つに基づいて、前記デザインパターンを自動的に特定する過程が、前記複数のアーチファクトのうちの前記少なくとも１つを、前記デザインパターンを表す予め特定されたパターンに照合することを含む、方法。
【請求項１４】
請求項１に記載の方法において、前記複数のアーチファクトのうちの前記少なくとも１つが、それぞれ静的アーチファクトである、方法。
【請求項１５】
請求項１に記載の方法において、前記複数のアーチファクトのうちの前記少なくとも１つが、それぞれ動的アーチファクトである、方法。
【請求項１６】
請求項１に記載の方法において、前記複数のアーチファクトのうちの前記少なくとも１つが、それぞれ導出アーチファクトである、方法。
【請求項１７】
請求項１に記載の方法において、前記複数のアーチファクトのうちの前記少なくとも１つが、それぞれメタデータアーチファクトである、方法。
【請求項１８】
請求項１に記載の方法において、さらに、
前記デザインパターンについての識別子を前記データベースに記憶する過程、
を備える、方法。
【請求項１９】
請求項１８に記載の方法において、前記デザインパターンについての識別子を記憶する過程が、前記第１のファイルについての、前記複数のアーチファクトのうちの少なくとも１つから得られた文字列を用いて、前記デザインパターンについてのラベルを記憶することを含む、方法。
【請求項２０】
請求項２に記載の方法において、さらに、
前記第１のファイルにおいて、前記デザインパターンに対応するプログラム断片を見つけ出す過程、
を備える、方法。
【請求項２１】
請求項２０に記載の方法において、前記第１のファイルが、バイナリコードフォーマットである、方法。
【請求項２２】
請求項２０に記載の方法において、前記第１のファイルが、ソースコードフォーマットである、方法。
【請求項２３】
請求項２０に記載の方法において、前記第１のファイルが、中間表現（ＩＲ）フォーマットである、方法。
【請求項２４】
デザインパターンを特定する方法であって、
複数のアーチファクトを有するデータベースにアクセスする過程と、
前記複数のアーチファクトをクラスタ化する過程と、
前記クラスタ化から、これまで特定されていなかったデザインパターンを、少なくとも１つの予め特定されたデザインパターンに基づいて特定する過程と、
を備える、方法。
【請求項２５】
請求項２４に記載の方法において、前記これまでに特定されていなかったデザインパタ

ーンと前記少なくとも１つの予め特定されたデザインパターンとが、同じデザインパターンである、方法。
【請求項２６】
請求項２４に記載の方法において、前記予め特定されたデザインパターンが、欠陥である、方法。
【請求項２７】
請求項２６に記載の方法において、さらに、
予め特定された前記欠陥に対応付けられた少なくとも１つの修復を特定する過程、
を備える、方法。
【請求項２８】
請求項２４に記載の方法において、前記複数のアーチファクトが、複数の開発中アーチファクトを含み、当該方法が、さらに、
クラスタ化された前記複数のアーチファクトに対応する前記複数の開発中アーチファクトから、当該開発中アーチファクトにおける文字、単語またはフレーズの出現に基づいて語義上意味を抽出する過程、
を備える、方法。
【請求項２９】
請求項２４に記載の方法において、前記複数のアーチファクトをクラスタ化する過程が、機械学習を用いることを含む、方法。
【請求項３０】
請求項２４に記載の方法において、前記複数のアーチファクトをクラスタ化する過程が、深層学習を用いることを含む、方法。
【請求項３１】
請求項２４に記載の方法において、前記複数のアーチファクトをクラスタ化する過程が、オートエンコーダを用いることを含む、方法。
【請求項３２】
請求項２４に記載の方法において、さらに、
前記複数のアーチファクトの前記クラスタ化のための訓練を提供する過程、
を備え、前記訓練が、ソフトウェアファイルの第１のバージョンと当該ソフトウェアファイルの第２のバージョンとの間の少なくとも１つの違いを用いることを含む、方法。
【請求項３３】
請求項３２に記載の方法において、前記少なくとも１つの違いが、欠陥または修復に対応する、方法。
【請求項３４】
請求項３３に記載の方法において、前記欠陥がセキュリティ脆弱性であるか、あるいは、前記修復がパッチである、方法。
【請求項３５】
請求項３２に記載の方法において、前記少なくとも１つの違いが、機能または付加拡張機能に対応する、方法。
【請求項３６】
デザインパターンを特定するシステムであって、
複数のファイルのそれぞれについての複数のアーチファクトを有する少なくとも１つの記憶装置と、
前記複数のファイルのうちの第１のファイルについての、前記複数のアーチファクトのうちの少なくとも１つに基づいて、デザインパターンを自動的に特定するように構成されたプロセッサと、
を備える、システム。
【請求項３７】
請求項３６に記載のシステムにおいて、さらに、前記プロセッサを備え、当該プロセッサが、さらに、前記第１のファイルにおいて、前記デザインパターンを組み込むプログラ

ム断片を見つけ出すように構成されている、システム。
【請求項３８】
請求項３６に記載のシステムにおいて、前記デザインパターンを自動的に特定することが、前記デザインパターンの前記特定を、前記複数のファイルのうちの第２のファイルについての、前記複数のアーチファクトのうちの少なくとも１つにも基づかせることを含み、前記第１のファイルおよび前記第２のファイルが、いずれも同じプロジェクトに属する、システム。
【請求項３９】
請求項３６に記載のシステムにおいて、前記デザインパターンが、欠陥または修復である、システム。
【請求項４０】
請求項３６に記載のシステムにおいて、前記デザインパターンが、機能または付加拡張機能である、システム。
【請求項４１】
請求項３６に記載のシステムにおいて、前記デザインパターンが、予め特定されたプログラム断片である、システム。
【請求項４２】
デザインパターンを特定するシステムであって、
複数のアーチファクトを有する少なくとも１つの記憶装置と、
前記複数のアーチファクトをクラスタ化するように、かつ、当該クラスタ化から、これまで特定されていなかったデザインパターンを、少なくとも１つの予め特定されたデザインパターンに基づいて特定するように構成されたプロセッサと、
を備える、システム。
【請求項４３】
請求項４２に記載のシステムにおいて、前記予め特定されたデザインパターンが、欠陥である、システム。
【請求項４４】
請求項４２に記載のシステムにおいて、さらに、
予め特定された前記欠陥に対応付けられた少なくとも１つの修復を特定すること、
を備える、システム。
【請求項４５】
請求項４２に記載のシステムにおいて、前記複数のアーチファクトをクラスタ化することが、機械学習を用いることを含む、システム。
【請求項４６】
請求項４２に記載のシステムにおいて、前記複数のアーチファクトをクラスタ化することが、深層学習を用いることを含む、システム。
【請求項４７】
実行可能なプログラムが記憶された、非過渡的なコンピュータ読取り可能な媒体であって、前記プログラムが、処理装置に：
複数のファイルのそれぞれについての複数のアーチファクトを有するデータベースにアクセスする手順；および
前記複数のファイルのうちの第１のファイルについての、前記複数のアーチファクトのうちの少なくとも１つに基づいて、デザインパターンを自動的に特定する手順；
を実行させる、非過渡的なコンピュータ読取り可能な媒体。
【請求項４８】
実行可能なプログラムが記憶された、非過渡的なコンピュータ読取り可能な媒体であって、前記プログラムが、処理装置に：
複数のアーチファクトを有するデータベースにアクセスする手順；
前記複数のアーチファクトをクラスタ化する手順；および
前記クラスタ化から、これまで特定されていなかったデザインパターンを、少なくとも

１つの予め特定されたデザインパターンに基づいて特定する手順；
を実行させる、非過渡的なコンピュータ読取り可能な媒体。
【発明の詳細な説明】
【関連出願】
【０００１】
本願は、2014年6月13日出願の米国仮特許出願第62/012,127号の利益を主張する。この米国仮特許出願の全教示内容は、参照をもって本願に取り入れたものとする。
［政府支援］
【０００２】
本発明は、アメリカ空軍からの助成金登録番号FA8750-14-C-0056およびアメリカ国防総省高等研究計画局からの助成金登録番号FA8750-15-C-0242の下の政府支援を受けてなされたものである。政府は、本発明に一定の権利を有する。
【背景技術】
【０００３】
現今のソフトウェア開発、保守および修復は、人間によって行われる。ソフトウェアベンダーは、時間をかけて、コンピュータプログラムの計画、実装、マニュアル化、テスト、導入（インストール）および保守を行う。当初の計画、実装、マニュアル、テストおよび導入は、しばしば不完全であり、所望の機能を有していなかったり欠陥を含んでいたりすることが必ず起こる。多くのベンダーは、ソフトウェアの運用が進むにつれて逐次バグ修正、セキュリティパッチおよび付加拡張機能を配信してこれらの欠点に対処する、ライフサイクル保守プランを有している。
【０００４】
世界には、何十億行もの大量のソフトウェアコードが配備されており、保守およびバグ修正に取り組むには、大量の時間および費用が必要となる。歴史的にみると、ソフトウェア保守は、場当たり的で且つ反作用的な（つまり、バグレポート、セキュリティ脆弱性レポート、および付加拡張機能についてのユーザ要求に対応する）人的プロセスであった。
【発明の概要】
【発明が解決しようとする課題】
【０００５】
本発明の実施形態は、例えば、バグ（コード内のエラー）、セキュリティ脆弱性、プロトコル不備などのプログラム欠陥を見つけ出して修復すること等を含む、ソフトウェア開発、保守、および修復ライフサイクルにおける重要な側面を自動化する。本発明の例示的な実施形態は、公衆が利用可能なソフトウェアや工業所有権によって保護されているソフトウェアを含む、大量のソフトウェアファイルを活用することが可能なシステム及び方法を提供する。
【課題を解決するための手段】
【０００６】
本発明の一実施形態において、デザインパターンを特定する例示的な方法は、複数のファイルのそれぞれについての複数のアーチファクトを有するデータベースにアクセスする過程と、前記複数のファイルのうちの第１のファイルについての、前記複数のアーチファクトのうちの少なくとも１つに基づいて、デザインパターンを自動的に特定する過程と、を備える。これらのファイルは、例えば、バイナリコードフォーマット、ソースコードフォーマット、中間表現（ＩＲ）フォーマットであり得る。
【０００７】
一部の実施形態において、前記デザインパターンは、前記第１のファイル内のものである。例示的な他の実施形態において、前記デザインパターンは、ファイル間（例えば、同じプロジェクト内のファイル間など）またはコードの断片間の相互作用に関するものであり得て、この場合、前記デザインパターンを自動的に特定する過程は、第２のファイル等についてのアーチファクトにも基づくものとされ得る。
【０００８】

一部の実施形態において、前記デザインパターンは、欠陥、修復、機能、付加拡張機能、または予め特定されたプログラム断片であり得る。さらなる他の実施形態は、前記複数のアーチファクトのうちの少なくとも１つ（例えば、開発中アーチファクト）において、欠陥、修復、機能もしくは付加拡張機能を表す文字列、または前記デザインパターンを表す予め特定されたパターンを探し出し得る。例示的な実施形態において、前記アーチファクトは、静的アーチファクト、動的アーチファクト、導出アーチファクトまたはメタデータアーチファクトであり得る。

【０００９】

例示的な他の実施形態は、前記デザインパターンについての識別子を前記データベースに記憶し得る。例えば、前記第１のファイルについての、前記複数のアーチファクトのうちの少なくとも１つから得られた文字列を用いる等して、前記デザインパターンについてのラベルが使用されてもよい。他の実施形態は、前記第１のファイルにおいて、前記デザインパターンに対応するプログラム断片を見つけ出し得る。

【００１０】

本発明の一実施形態において、デザインパターンを特定する例示的な方法は、複数のソフトウェアファイルに対応する複数のアーチファクトを有するデータベースにアクセスする過程と、前記ソフトウェアファイルのうちの少なくとも１つについてのデザインパターンを、前記ソフトウェアファイルに対応付けられた前記アーチファクトのうちの少なくとも１つを自動的に分析することによって特定する過程と、を備える。例示的な前記方法の他の実施形態は、さらに、前記ソフトウェアファイルについての前記デザインパターンについての、識別子を前記データベースに記憶する過程を備える。

【００１１】

例示的な一部の実施形態において、前記アーチファクトは、インラインコードコメント、コミット履歴、ドキュメンテーションファイル、および共通脆弱性識別子ソース登録のうちの少なくとも１つを含む。例示的な一部の実施形態において、前記アーチファクトのうちの少なくとも１つを分析することは、欠陥または修復を表す列について、開発中アーチファクトの検索を行うことを有する。また、例示的な前記方法の他の実施形態は、前記ソフトウェアファイルにおいて、前記デザインパターンを組み込むプログラム断片を見つけ出す過程を備える。例示的な一部の実施形態において、前記デザインパターンに対応する前記プログラム断片は、前記ソフトウェアファイルの中間表現において、前記デザインパターンを組み込むコードを探し出すことによって見つけ出される。

【００１２】

例示的な他の実施形態において、前記ソフトウェアファイルの前記デザインパターンについての識別子を前記データベースに記憶する過程は、前記ソフトウェアファイルについての、前記アーチファクトのうちの少なくとも１つから得られた列を用いて、前記デザインパターンについてのラベルを記憶することを含む。例示的な実施形態において、前記デザインパターンは、欠陥、修復、機能または付加拡張機能である。

【００１３】

本発明の例示的な他の実施形態は、デザインパターン（例えば、欠陥等）を特定する方法であって、ソフトウェアファイルに対応するアーチファクトを有するデータベースにアクセスする過程と、前記アーチファクトをクラスタ化する過程と、前記クラスタ化から、これまで特定されていなかったデザインパターンを、少なくとも１つの予め特定されたデザインパターンに基づいて特定する過程と、を備える、方法である。例示的な一部の実施形態において、そのデザインパターンは同じであるが、例えば、互いに別のファイルに存在し得る。例示的な一部の実施形態において、例示的な前記方法は、さらに、
予め特定された前記欠陥に対応付けられた少なくとも１つの修復を特定する過程、を備える。

【００１４】

例示的な一部の実施形態において、前記アーチファクトは、開発中アーチファクトを含み、例示的な前記方法が、さらに、前記開発中アーチファクトから、当該アーチファクト

における文字（英数字や特殊文字を含む）、単語またはフレーズの出現に基づいて語義上意味（意味論的意味、セマンティック意味）を抽出する過程、を備える。例示的な一部の実施形態において、前記複数のアーチファクトをクラスタ化する過程は、オートエンコーダを用いることを含む。また、他の実施形態は、前記複数のアーチファクトの前記クラスタ化のための訓練を提供する過程、を備え、前記訓練が、ソフトウェアファイルの第１のバージョンと当該ソフトウェアファイルの第２のバージョンとの間の少なくとも１つの違いを用いることを含む。一部の実施形態において、これらの違いは、欠陥（例えば、セキュリティ脆弱性）または修復（例えば、パッチ）に対応し得る。一部の実施形態において、これらの違いは、機能または付加拡張機能に対応し得る。さらなる他の実施形態では、アーチファクトの種類毎に、クラスタ化が行われる。例示的な実施形態において、そのような種類には、コールグラフ、制御フローグラフ、ｕｓｅ－ｄｅｆチェイン、ｄｅｆ－ｕｓｅチェイン、支配木、基本ブロック、変数、定数、ブランチセマンティック（分岐意味）、およびプロトコルが含まれる。例示的な一部の実施形態では、複数の種類のアーチファクトに基づいて、クラスタ化が行われ得る。

【００１５】

本発明の例示的な他の実施形態は、デザインパターンを特定するシステムであって、ソフトウェアファイルに対応するアーチファクトを有する少なくとも１つの記憶装置であって、前記アーチファクトが、当該記憶装置に記憶されたアーチファクトを含む、少なくとも１つの記憶装置と、前記ソフトウェアファイルのうちの少なくとも１つについてのデザインパターンを、前記アーチファクトのうちの、当該ソフトウェアファイルに対応付けられた少なくとも１つを自動的に分析することによって特定するように構成されたプロセッサと、を備える、システムである。また、例示的な前記システムは、前記プロセッサを備え得て、当該プロセッサが、前記ソフトウェアファイルにおいて、前記デザインパターンを組み込むプログラム断片を見つけ出すように構成され得る。

【００１６】

本発明の例示的な他の実施形態は、デザインパターンを特定するシステムであって、複数のアーチファクトを有する少なくとも１つの記憶装置と、前記複数のアーチファクトをクラスタ化するように、かつ、当該クラスタ化から、これまで特定されていなかったデザインパターンを、少なくとも１つの予め特定されたデザインパターンに基づいて特定するように構成されたプロセッサと、を備える、システムである。例示的な一部の実施形態において、前記デザインパターンは、欠陥、修復、機能、付加拡張機能または予め特定されたパターンである。一部の実施形態において、前記クラスタ化は、機械学習または深層学習を用いることを含む。

【００１７】

本発明の例示的な他の実施形態は、実行可能なプログラムが記憶された、非過渡的なコンピュータ読取り可能な媒体であって、前記プログラムが、処理装置に：

ソフトウェアファイルに対応するアーチファクトを有するデータベースにアクセスする手順；および

前記複数のファイルのうちの第１のファイルについての、前記複数のアーチファクトのうちの少なくとも１つに基づいて、デザインパターンを自動的に特定する手順；

を実行させる、非過渡的なコンピュータ読取り可能な媒体である。

【００１８】

本発明の例示的な他の実施形態は、実行可能なプログラムが記憶された、非過渡的なコンピュータ読取り可能な媒体であって、前記プログラムが、処理装置に：

複数のアーチファクトを有するデータベースにアクセスする手順；

前記複数のアーチファクトをクラスタ化する手順；および

前記クラスタ化から、これまで特定されていなかったデザインパターンを、少なくとも１つの予め特定されたデザインパターンに基づいて特定する手順；

を実行させる、非過渡的なコンピュータ読取り可能な媒体である。

【００１９】

前述の内容は、添付の図面に示された本発明の例示的な実施形態についての、以下の詳細な説明から明らかになる。異なる図面をとおして、同一の符号は同一の構成／構成要素を指すものとする。図面は必ずしも縮尺どおりではなく、むしろ、本発明の実施形態を示すことに重点が置かれている。

【図面の簡単な説明】

【００２０】

【図１】ソフトウェアファイルについてのコーパスを提供する方法の例示的な一実施形態を示すフロー図である。

【図２】本発明の一実施形態における、コーパスへの入力ソフトウェアファイルから中間表現（ＩＲ）を抽出するための処理の一例を示すフロー図である。

【図３】本発明の一実施形態における、ソフトウェアファイルについてのアーチファクト間の階層関係を示すブロック図である。

【図４】ソフトウェアファイルについてのアーチファクトのコーパスを提供するシステムの例示的な一実施形態を示すブロック図である。

【図５】デザインパターンを特定する方法の例示的な一実施形態を示すブロック図である。

【図６】欠陥を特定する方法の例示的な一実施形態を示すフロー図である。

【図７】本発明の一実施形態における、デザインパターンを特定するためのアーチファクトのクラスタ化を示すブロック図である。

【図８】コーパスを用いてソフトウェアファイルを特定する方法の例示的な一実施形態を示すフロー図である。

【図９】プログラム断片を特定する方法の例示的な一実施形態を示すフロー図である。

【図１０】本発明の一実施形態における、コーパスを用いるシステムを示すブロック図である。

【発明を実施するための形態】

【００２１】

以下では、本発明の例示的な実施形態について説明する。本明細書で引用する特許文献や刊行物の全教示内容は、参照をもって本明細書に取り入れたものとする。

【００２２】

本明細書での例示的な実施形態におけるソフトウェア解析は、公衆が利用可能なソースからのファイルや工業所有権によって保護されているソフトウェアからのファイルを含む、既存のソフトウェアファイルからの知識を活用することを可能にする。そして、この知識は、他のソフトウェアファイルに適用されることが可能である。この適用には、欠陥を修復すること、脆弱性を特定すること、プロトコル不備を特定すること、またはコード改善を提案することが含まれる。

【００２３】

本発明の例示的な実施形態は、ソフトウェア解析における様々な構成に向けられ得る。そのような様々な構成には、知識データベースのための、ソフトウェアファイルのコーパス（集成）および当該ソフトウェアファイルについての関連アーチファクトのコーパスを、作成、更新、保有または提供することが含まれる。このコーパスは、本発明の構成に従って様々な目的に用いられ得る。そのような様々な目的には、ソフトウェアファイルのより新しいバージョン、ソフトウェアファイルに利用可能なパッチ、欠陥を有することが知られているファイルにおける当該欠陥、および既知の欠陥（エラー）を含むことがこれまで知られていなかったファイルにおける当該既知の欠陥を自動的に特定することが含まれる。また、本発明の実施形態は、これらの問題に対処するために前記コーパスからの知識を活用し得る。

【００２４】

図１は、本発明の一実施形態における、コーパスへの入力ソフトウェアファイルの処理の一例を示すフロー図である。図示の最初のステップでは、複数のソフトウェアファイルを得る（符号１１０）。これらのソフトウェアファイルは、ソースコードフォーマット（

典型的には、プレーンテキストである）、バイナリコードフォーマット、または他の何らかのフォーマットであり得る。また、本発明の例示的な一部の実施形態において、前記ソースコードフォーマットは、コンパイル可能なコンピュータ言語であれば、どのようなコンピュータ言語であってもよい。そのようなコンピュータ言語には、Ａｄａ、Ｃ／Ｃ＋＋、Ｄ、Ｅｒｌａｎｇ、Ｈａｓｋｅｌｌ、Ｊａｖａ（登録商標）、Ｌｕａ、Ｏｂｊｅｃｔｉｖｅ　Ｃ／Ｃ＋＋、ＰＨＰ、Ｐｕｒｅ、Ｐｙｔｈｏｎ、およびＲｕｂｙが含まれる。例示的な他の一部の実施形態では、本発明の実施形態に使用するのに、インタプリタ型言語が得られてもよい。そのようなインタプリタ型言語には、ＰＥＲＬおよびｂａｓｈ　ｓｃｒｉｐｔが含まれる。

【００２５】

得られるソフトウェアファイルには、ソースコードファイルやバイナリファイルだけでなく、それらのファイルに関連付けられているか又は対応するソフトウェアプロジェクトに関連付けられている、任意のファイルが含まれてもよい。例えば、ソフトウェアファイルには、さらに、関連付けられている、ビルドファイル、ｍａｋｅファイル、ライブラリ、マニュアルファイル、コミットログ、変更履歴、バグジラ（Ｂｕｇｚｉｌｌａ）登録、共通脆弱性識別子（ＣＶＥ）登録、および他の非構造化テキストが含まれる。

【００２６】

これらのソフトウェアファイルは、様々なソースから得られ得る。例えば、ソフトウェアファイルは、ネットワークインターフェースを介して、インターネットにより、公衆が利用可能なソフトウェアレポジトリから得られ得る。そのようなソフトウェアレポジトリとして、例えば、ＧｉｔＨｕｂ、ＳｏｕｒｃｅＦｏｒｇｅ、ＢｉｔＢｕｃｋｔ、ＧｏｏｇｌｅＣｏｄｅ、共通脆弱性識別子（Ｃｏｍｍｏｎ　Ｖｕｌｎｅｒａｂｉｌｉｔｉｅｓ　ａｎｄ　Ｅｘｐｏｓｕｒｅｓ）システム（例えば、ＭＩＴＲＥ社により保有されるもの）が挙げられる。一般的に、これらのレポジトリは、ファイルおよび当該ファイルに施された変更の履歴を含む。この他に、例えば、ファイルが得られるサイトを指し示すユニフォームリソースロケータ（ＵＲＬ）が提供されてもよい。また、ソフトウェアファイルは、インターフェースを介してプライベートネットワークから得られるか、または、局所的なローカルハードドライブ又は他の記憶装置から得られてもよい。このようなインターフェースは、ソースとの通信可能な接続を提供する。

【００２７】

本発明の例示的な実施形態は、ソースから入手可能なファイルのうち、一部、ほとんど、または全てを取得してもよい。また、例示的な一部の実施形態は、ファイルを得ることを自動化し、例えば、ファイル、ソフトウェアプロジェクト全体（例えば、変更履歴、コミットログ、ソースコード等）、プロジェクトもしくはプログラムの全ての改変（リビジョン）、ディレクトリ内の全てのファイル、またはソースから入手可能な全てのファイルを自動的にダウンロードし得る。一部の実施形態は、レポジトリの全体について、入手可能なソフトウェアファイルの全てを得るために、各改変を丁寧に調べる。例示的な一部の実施形態は、各ソフトウェアプロジェクトについてのソース管理レポジトリの全体を前記コーパスで取得することにより、そのプロジェクトについての全ての関連付けられているファイルを自動的に得ること（各ソフトウェアファイル改変を得ることを含む）を容易にする。レポジトリ用のソース管理システムには、例えば、Ｇｉｔ、Ｍｅｒｃｕｒｉａｌ、Ｓｕｂｖｅｒｓｉｏｎ、Ｃｏｎｃｕｒｒｅｎｔ　Ｖｅｒｓｉｏｎｓ　Ｓｙｓｔｅｍ、ＢｉｔＫｅｅｐｅｒ、Ｐｅｒｆｏｒｃｅが含まれる。また、一部の実施形態は、ソースが変更又は更新されたか否かを判別するように当該ソースを継続的に又は周期的に再確認してもよく、かつ、ソースが変更又は更新された場合には、当該ソースから変更点もしくは更新点のみを得るか又は全てのソフトウェアファイルを再び得るものであってもよい。多くのソースは、当該ソースへの変更を判別するための方法（例えば、追加日付フィールド、変更日付フィールドであって、例示的な実施形態がソースから更新点を得るのに用い得る、追加日付フィールド、変更日付フィールド）を備えている。

【００２８】

また、本発明の例示的な一部の実施形態は、レポジトリから得られたソースコードファイルにより使用され得るライブラリソフトウェアファイルを、レポジトリがこのようなライブラリを含まなかった場合の当該ファイルの必要性に対処するために別個に得るものであってもよい。これらのうちの一部の実施形態は、前記コーパスに含めるために、任意のパブリックソースから合理的に入手可能であるか又はソフトウェアベンダーから得られる任意のライブラリソフトウェアファイルを得ることを試みる。一部の実施形態は、さらに、ソフトウェアファイルにより使用されるライブラリをユーザが提供することを可能にするか、あるいは、使用されるライブラリをユーザが特定して当該ライブラリを得られるようにすることを可能にする。一部の実施形態は、各プロジェクトについてのソフトウェアファイルをくまなく調べることで、そのプロジェクトにより使用されるライブラリを特定して当該ライブラリを得たり必要に応じてインストールしたりできるようにする。

【００２９】

本発明における例示的な方法での次のステップでは、前記複数のソフトウェアファイルのそれぞれについて複数のアーチファクトを決定する（符号１２０）。ソフトウェアアーチファクトは、ソフトウェアファイルの機能、アーキテクチャまたはデザインを記述し得る。アーチファクトの種類には、例えば、静的アーチファクト、動的アーチファクト、導出アーチファクト、メタデータアーチファクト等が含まれる。

【００３０】

例示的なこの方法での最後のステップでは、前記複数のソフトウェアファイルのそれぞれについての前記複数のアーチファクトをデータベースに記憶する（符号１３０）。これら複数のアーチファクトは、それらが決定された特定のソフトウェアファイルに対応するものとして特定可能なように記憶される。この特定は、データベーススキーマにより表現される前記データベース内のフィールド、ポインタ、記憶されている場所の位置、ファイル名などの他の任意の識別子といった周知の様々な方法のどれによって行われてもよい。同じプロジェクト又は同じビルドに属するファイル同士が、関係を維持可能なように同様に追跡されてもよい。

【００３１】

種々の実施形態に対して、前記データベースは、グラフデータベース、関係データベース、フラットファイルなどといった異なる形態を取り得る。好適な一実施形態は、Ｏｒｉｅｎｔ Ｔｅｃｈｎｏｌｏｇｉｅｓ社主体のＯｒｉｅｎｔＤＢ Ｏｐｅｎ Ｓｏｕｒｃｅ Ｐｒｏｊｅｃｔにより提供される分散グラフデータベースであるＯｒｉｅｎｔＤＢを用いる。好適な他の実施形態は、マルチマシンクラスタにわたって分散したグラフを記憶及びクエリするのに最適化されたスケーラブルなグラフデータベースであるＴｉｔａｎと、Ａｐａｃｈｅ Ｃａｓｓａｎｄｒａストレージバックエンドとを用いる。また、例示的な一部の実施形態は、グラフアーチファクトを記憶し当該グラフアーチファクトに作用する配列データベースである、Ｐａｒａｄｉｇｍ４からのＳｃｉＤＢを用いる。

【００３２】

一般的に、静的アーチファクト、動的アーチファクト、導出アーチファクトおよびメタデータアーチファクトは、ソースコードファイル、バイナリファイルまたは他のアーチファクトから決定され得る。これらの種類のアーチファクトの例については、以下で説明する。例示的な実施形態は、ソースコードソフトウェアファイル又はバイナリソフトウェアファイルについて、これらのうちの少なくとも１つのアーチファクトを決定し得る。一部の実施形態は、これら全種類の全アーチファクト又は特定の種類のアーチファクトにおける全アーチファクトを決定するのではなく、むしろ、一部の種類のアーチファクトおよび／または所与の種類における一部のアーチファクトを決定するものであってもよく、かつ／あるいは、特定の種類におけるアーチファクトを全く決定しないものであってもよい。

【００３３】

＜静的アーチファクト＞

ソフトウェアファイルについての静的アーチファクトは、コールグラフ、制御フローグラフ、ｕｓｅ－ｄｅｆチェイン、ｄｅｆ－ｕｓｅチェイン、支配木、基本ブロック、変数

、定数、ブランチセマンティック（分岐意味）、およびプロトコルを含む。

【００３４】

コールグラフ（ＣＧ）は、関数により呼び出される関数の有向グラフである。ＣＧは、高位レベルのプログラム構造を表現するものであり、そのグラフの各ノードは関数を表し、ノード間の各エッジは方向を有し且つある関数が別の関数を呼び出し得るか否かを示す。

【００３５】

制御フローグラフ（ＣＦＧ）は、関数内部の基本ブロック間の制御フローの有向グラフである。ＣＦＧは、関数レベルのプログラム構造を表現する。ＣＦＧの各ノードは基本ブロックを表し、ノード間のエッジは方向を有し且つフロー内の経路候補を示す。

【００３６】

Ｕｓｅ－Ｄｅｆ（ＵＤ）チェインおよびＤｅｆ－Ｕｓｅ（ＤＵ）チェインは、入力（使用）、出力（定義）、およびコードの基本ブロック内で行われる処理の、有向非巡回グラフである。例えば、ＵＤチェインは、変数の使用と、当該変数の定義であって、再定義を介さずにその使用に到達し得る全ての定義とである。ＤＵチェインは、変数の定義と、使用であって、再定義を介さずにその定義から到達し得る全ての使用とである。これらのチェインは、受け付けられた入力型、生成される出力型、およびコードの基本ブロック内で行われる処理についての、コードの基本ブロックの意味解析を可能にする。

【００３７】

支配木（ＤＴ）は、ＣＦＧのどのノードが他のノードを支配するのか（どのノードが他のノードの経路にあるのか）を表現する行列である。例えば、入口ノードから第２のノードへの全ての経路が第１のノードを通らなければならない場合、第１のノードが第２のノードを支配するという。ＤＴは、前支配木（入口順方向）と後支配木（出口逆方向）とで表現され得る。ＤＴは、ＣＦＧにおいてある経路が特定のノードに切り換わるときを強調する。

【００３８】

基本ブロックは、ＣＦＧの各ノード内の命令およびオペランドである。基本ブロック同士は比較可能であり、かつ、２つの基本ブロック間の類似性尺度が生成可能である。

【００３９】

変数は、情報及びその情報の型についての記憶単位であり、任意の関数パラメータ、任意のローカル変数又は任意のグローバル変数についての記憶可能な情報の型を表現し、デフォルト値が存在する場合にはデフォルト値を有し得る。変数は、プログラムに対する初期状態および基本制約を提供し得て、かつ、プログラム挙動に影響を与え得る前記型の変化又は初期値の変化を示し得る。

【００４０】

定数は、任意の定数の型及び数値であり、プログラムに対する初期状態および基本制約を提供し得る。定数は、プログラム挙動に影響を与え得る前記型の変化又は初期値の変化を示し得る。

【００４１】

ブランチセマンティック（分岐意味）は、ｉｆ文やループ内のブーリアン評価である。分岐は、基本ブロックが実行される条件を制御する。

【００４２】

プロトコルは、プログラムにより使用されるプロトコル、ライブラリ、システムコール、および他の既知の関数の、名前とリファレンス（参照先）とである。

【００４３】

本発明の例示的な実施形態は、ソフトウェアソースコードファイルの中間表現（ＩＲ）（例えば、公衆が入手可能なＬＬＶＭ（かつては低水準仮想機械）コンパイラインフラストラクチャプロジェクト等により提供される中間表現）から静的アーチファクトを自動的に決定し得る。ＬＬＶＭ ＩＲは、高水準言語を効果的に表現可能であり且つＡＲＭ、Ｘ８６、Ｘ６４、ＭＩＰＳ、ＰＰＣなどの命令セットアーキテクチャ（ＩＳＡ）から独立し

ている、低水準共通言語である。コンピュータ言語が異なっても、異なるコンピュータ言語用の異なるＬＬＶＭコンパイラ（ここでの「コンパイラ」はフロントエンドとも称される）を用いて、ソースコードを共通のＬＬＶＭ ＩＲに変換させることが可能であり得る。少なくともＡｄａ、Ｃ／Ｃ＋＋、Ｄ、Ｅｒｌａｎｇ、Ｈａｓｋｅｌｌ、Ｊａｖａ、Ｌｕａ、Ｏｂｊｅｃｔｉｖｅ Ｃ／Ｃ＋＋、ＰＨＰ、Ｐｕｒｅ、Ｐｙｔｈｏｎ、およびＲｕｂｙ用のフロントエンドは、公衆が入手可能である。また、そのほかの言語用のフロントエンドも、簡単にプログラム可能である。また、ＬＬＶＭには利用可能な最適化器が存在し、かつ、ＬＬＶＭ ＩＲを様々な異なるＩＳＡ用の機械語に変換可能なバックエンドも存在する。例示的な他の実施形態は、ソースコードファイルから静的アーチファクトを決定し得る。

【００４４】

図２は、本発明の一実施形態において利用可能な、コーパスへの入力ソフトウェアファイルの処理の他の例を示すフロー図である。例示的な実施形態は、特に、ソースコード２０５ソフトウェアファイルとバイナリコード２１０ソフトウェアファイルとの両方を得ることができる。ソースコードファイル２０５の言語用のＬＬＶＭコンパイラ２２０が利用可能な場合には、この言語用のＬＬＶＭコンパイラ２２０を用いてそのソースコードがＬＬＶＭ ＩＲ２５０に翻訳（変換）され得る。利用可能なＬＬＶＭコンパイラがないコンパイル後言語（コンパイル言語）の場合には、ソースコード２０５がまず、この言語用の任意のサポートされているコンパイラ２１５を用いてバイナリファイル２３０にコンパイルされ得る。次に、このバイナリファイル２３０が、デコンパイラ２３５（例えば、Ｄｒａｐｅｒ Ｌａｂｏｒａｔｒｙにより提供される公衆が入手可能なオープンソースデコンパイラであるＦｒａｃｔｕｒｅ等）を用いてデコンパイルされる。具体的に述べると、デコンパイラ２３５が、機械コード２３０をＬＬＶＭ ＩＲ２５０に翻訳する。バイナリ形式２１０で得られたファイルについては、これが機械コード２３０であることから、デコンパイラ２３５を用いてＬＬＶＭ ＩＲ２５０を得るようにデコンパイルされる。例示的な実施形態は、ＬＬＶＭ ＩＲから、言語非依存で且つＩＳＡから独立したアーチファクトを抽出し得る。

【００４５】

本発明の例示的な実施形態は、ソースコードソフトウェアファイルのそれぞれについてのＩＲを自動的に得ることができる。例えば、例示的な実施形態は、ａｕｔｏｃｏｍｆ、ｃｍａｋｅ、ａｕｔｏｍａｋｅ、ｍａｋｅファイル、ベンダーの命令などの標準ビルドファイルに対して、プロジェクト用のレポジトリの検索を自動的に行い得る。例示的な実施形態は、ビルドプロセスを監視してコンパイラ呼出しをソースコードで使用されている言語用のＬＬＶＭフロントエンド呼出しに変換することにより、プロジェクトをビルドするように上記のようなファイルを用いることを自動的かつ選択的に試み得る。ビルドファイルについてのこの選択プロセスは、ファイルのそれぞれを一つずつ調べて何が存在し且つ何が完成ビルド又は部分完成ビルド（部分的に完成したビルド）を提供するのかを判断し得る。

【００４６】

例示的な他の実施形態は、レポジトリからファイルを自動的に得るのに、および／または、ファイルをＬＬＶＭ ＩＲに変換するのに、および／または、ファイルについてのアーチファクトを決定するのに、分散型コンピュータシステムを使用し得る。分散型システムは、例えば、マスタコンピュータを用いて、プロジェクトやビルドを、スレーブマシンが処理するように当該スレーブマシンに渡し得る。それぞれのスレーブは、振り当てられたプロジェクト、バージョン、改変又はビルドを処理し得る。かつ、それぞれのスレーブは、ソースファイル又はバイナリファイルをＬＬＶＭ ＩＲへと翻訳し得て、および／または、アーチファクトを決定し得る。かつ、それぞれのスレーブは、結果を、前記コーパスに記憶されるように提供し得る。例示的な一部の実施形態は、超大規模のデータセットの分散記憶・分散処理のためのオープンソースソフトウェアフレームワークであるＨａｄｏｏｐを使用し得る。また、ソースレポジトリからファイルを得ることが、マシンの集団

内で分散されるものであってもよい。
【００４７】
例示的な実施形態において、ソフトウェアファイルおよびＬＬＶＭ ＩＲは、前記コーパス（分散ストレージを含む）に記憶され得る。また、例示的な実施形態は、ソフトウェアファイル又はＬＬＶＭ ＩＲコードがデータベースに既に記憶されていることを判定し得て、かつ、そのファイルを再び記憶しないことを選択し得る。ポインタ、グラフデータベースのエッジ、または他の参照先識別子を用いて、ファイルを、特定のプロジェクト、ディレクトリ、またはファイルの他の集まりに関連付けてもよい。
【００４８】
＜動的アーチファクト＞
動的アーチファクトは、プログラム挙動を表すものであり、ソフトウェアをその備えられた環境（例えば、仮想機械、エミュレータ（例えば、クイックエミュレータ（「ＱＥＭＵ」））、ハイパーバイザ等）で実行することにより生成される。動的アーチファクトは、システムコールトレース／ライブラリトレースおよび実行トレースを含む。
【００４９】
システムコールトレース又はライブラリトレースは、システムコール又はライブラリコールが実行される順序と頻度とである。システムコールは、プログラムが、入出力リクエストを管理するオペレーティングシステムのカーネルからのサービスを要求する方法である。ライブラリコールは、ソフトウェアプログラム及びアプリケーションを開発するのに再使用可能なプログラミングコードの集まりであるソフトウェアライブラリへの呼出しである。
【００５０】
実行トレースは、命令バイト、スタックフレーム、メモリ使用量（例えば、レジデント／ワーキングセットサイズ等）、ユーザ／カーネル時間、および他の実行時情報を含む、命令毎のトレースである。
【００５１】
本発明の例示的な実施形態は、仮想機械（様々なオペレーティングシステム用の仮想機械を含む）を生成し得てソースコードファイル及びバイナリファイルを実行およびコンパイルし得る。これらの環境は、動的アーチファクトが決定されることを可能にする。例えば、Ｖａｌｇｒｉｎｄ、Ｄａｉｋｏｎなどの公衆が入手可能なプログラムを用いることにより、当該プログラムについての実行時情報であって、アーチファクトとして機能する実行時情報が提供され得る。Ｖａｌｇｒｉｎｄは、特に、メモリのデバッグ、メモリリークの検出およびプロファイリングのためのツールである。Ｄａｉｋｏｎは、コードにおける不変式であって、コード内の決まった箇所で真となる条件である不変式を検出することが可能なプログラムである。
【００５２】
さらなる他の実施形態は、公衆が入手可能な、追加の診断・デバッグプログラム又はユーティリティ（例えば、ｓｔｒａｃｅ、ｄｔｒａｃｅ等）を使用し得る。ｓｔｒａｃｅは、プロセスとカーネルとの間の相互作用（システムコールを含む）を監視するのに用いられる。ｄｔｒａｃｅは、メモリ使用量、ＣＰＵ時間、特定の関数呼出し、および特定のファイルにアクセスするプロセスを含む、システムについての実行時情報を提供するのに用いられ得る。また、例示的な実施形態は、プログラムの複数の実行にわたって実行トレース（例えば、Ｖａｌｇｒｉｎｄ等を用いて）を追跡し得る。
【００５３】
他の実施形態は、ＫＬＥＥエンジンを介してＬＬＶＭ ＩＲを実行し得る。ＫＬＥＥは、公衆が入手可能なオープンソースコードであるシンボリックな仮想機械である。ＫＬＥＥは、ＬＬＶＭ ＩＲをシンボリックに実行し、かつ、全てのコードプログラム経路でのテストを自動的に生成する。シンボリック実行は、特に、どの入力がコードの各部分の実行を引き起こすのかを決定するようにそのコードを解析することに関するものである。ＫＬＥＥを使用することは、機能正確性エラーおよび挙動不一致を見つけ出すのに極めて効

果的なので、本発明の例示的な実施形態が類似コード同士の違い（例えば、改変にわたっての違い）を素早く特定することを可能にする。

【００５４】

＜導出アーチファクト＞

導出アーチファクトは、高位レベルの複雑なプログラム挙動を表すものであり、これらの挙動の特徴である特性及び事実を抽出する。導出アーチファクトは、プログラム特性、ループ不変条件、拡張型情報、Ｚ言語（Ｚ記法）およびラベル遷移体系表現を含む。

【００５５】

プログラム特性は、実行トレースから導出されるプログラムについての事実（情報）である。これらの事実は、最小メモリサイズ、最大メモリサイズ、平均メモリサイズ、実行時間およびスタック深さを含む。

【００５６】

ループ不変条件は、ループにおける全ての反復（又は選択された反復グループ）にわたって維持される特性である。ループ不変条件は、類似する挙動を明らかにするように分岐意味にマッピングされ得る。

【００５７】

拡張型情報は、型についての事実を含む。これらの事実には、変数が保持可能な数値の範囲、他の変数との関係、および抽象化可能な他の特徴が含まれる。型制約は、コードに関する挙動及び特徴を明らかにし得る。

【００５８】

Ｚ言語は、Ｚｅｒｍｅｌｏ－Ｆｒａｅｎｋｅｌ集合論に基づくものである。Ｚ言語は、型付き代数言語を提供し、基本ブロックと関数全体との間の、構造、順序及び型を無視した比較尺度を可能にする。

【００５９】

ラベル遷移体系（ＬＴＳ）表現は、プログラムから抽象化された高位レベルの状態を表現するグラフ体系である。このグラフのノードは状態であり、エッジは遷移内の関連する動作によりラベル付けされる。

【００６０】

例示的な一部の実施形態において、導出アーチファクトは、他のアーチファクトから決定され得たり、ソースコードファイルから決定され得たり（動的アーチファクトについて既述したプログラムを用いてソースコードファイルから決定されることを含む）、ＬＬＶＭ ＩＲから決定され得たりする。

【００６１】

＜メタデータアーチファクト＞

メタデータアーチファクトは、プログラムコンテキストを表すものであり、コードに関連付けられたメタデータを含む。これらのアーチファクトは、コンピュータプログラムに対してコンテキスト的関係を有する。メタデータアーチファクトは、ファイル名、改変番号、ファイルのタイムスタンプ、ハッシュ値、およびファイルの場所（例えば、特定のディレクトリ又はプロジェクトに属する等）を含む。一部のメタデータアーチファクトは、ファイル、プログラム又はプロジェクトの開発中プロセスに関するアーチファクトである開発中アーチファクトとも称され得る。開発中アーチファクトは、インラインコードコメント、コミット履歴、バグジラ登録、ＣＶＥ登録、ビルド情報、コンフィグスクリプト、およびマニュアルファイル（例えば、ＲＥＡＤＭＥ．＊、ＴＯＤＯ．＊等）を含み得る。

【００６２】

例示的な実施形態は、公衆が入手可能な文書（マニュアル）生成手段であるＤｏｘｙｇｅｎを使用し得る。Ｄｏｘｙｇｅｎは、特殊コメント付きソースコードファイル（つまり、インラインコードド文書）から、プログラマおよび／またはエンドユーザのためのソフトウェア文書を生成し得る。

【００６３】

他の実施形態は、Ａｎｏｔｈｅｒ Ｔｏｏｌ Ｆｏｒ Ｌａｎｇｕａｇｅ Ｒｅｃｏｇｎｉ

ｔｉｏｎ（ＡＮＴＬＲ）４－生成パーサ等のパーサ（構文解析ツール）を使用して抽象構文木（ＡＳＴ）を生成し得て、かつ、アーチファクトとしても機能し得る高位レベルの言語特徴を抽出し得る。ＡＮＴＬＲ４は、文法や言語についての列の生成則を捉えて、パース木を構築し得て当該パース木を辿り得るパーサを生成する。結果としてのパーサは、様々な型、関数定義／呼出し、およびプログラムの構造に関する他のデータを出力する。ＡＮＴＬＲ４－生成パーサにより抽出される低位レベルの属性は、複雑な型／構造、ループ不変条件／カウンタ（例えば、各パラダイムのａから）、および構造化されたコメント（例えば、形式的な事前／事後条件記述）を含む。例示的な実施形態は、この抽出されたデータをＬＬＶＭ ＩＲにおけるその被参照位置へとマッピングし得る。これは、ファイル名、行番号および列番号情報が、パーサにもＬＬＶＭ ＩＲにも存在するからである。

【００６４】

本発明の例示的な実施形態は、少なくとも１つのメタデータアーチファクトを、ソースソフトウェアファイルからインラインコメントなどの文字列を抽出することによって自動的に決定し得る。さらなる他の実施形態は、メタデータアーチファクトをファイルシステムまたはソース管理システムから自動的に決定する。

【００６５】

＜アーチファクト間階層関係＞

図３は、本発明の一実施形態における、ソフトウェアファイルについてのアーチファクト間の階層関係を示すブロック図である。例示的な実施形態は、これらのアーチファクト間階層関係を維持および利用し得る。また、異なる実施形態は、異なるスキーマおよび異なる階層関係を用い得る。図３の例示的な実施形態では、アーチファクト階層構造の最上位が、ＬＴＳアーチファクト３１０である。それぞれのＬＴＳノード３１０は、関数及び特定の変数状態の集合又は部分集合へとマッピング可能である。ＬＴＳアーチファクト３１０の下に、ＣＧアーチファクト３２０が存在する。それぞれのＣＧノード３２０は、ＣＦＧアーチファクト３３０を有する特定の関数にマッピング可能である（ＣＦＧアーチファクト３３０のエッジは、ループ不変条件及び分岐意味３３０を含み得る）。それぞれのＣＦＧノード３３０は、基本ブロック及びＤＴ３４０を含み得る。それらのアーチファクトの下に、変数、定数、ＵＤ／ＤＵチェインおよびＩＲ命令３５０が存在する。図３には、アーチファクトが、様々な動的情報を記述するＬＴＳノードから個々のＩＲ命令までの、階層構造における種々の階層へとマッピング可能であることが明らかに示されている。例示的な実施形態により、これらの階層関係は、マッチするアーチファクトをより効率的に検索することを含む、様々な用途に用いられ得る。これは、例えば、まず階層構造の最上位に近いアーチファクトと比較することにより（最下位に近いアーチファクトと比較するのではなくて）、上位のアーチファクトがマッチであるか（該当するか）否かに応じて当該上位のアーチファクトに関連付けられたそれよりも下位のアーチファクトの集合全体を含めるか又は除外することにより行われ得る。また、他の実施形態は、これらの階層関係を用いて、欠陥についての又は付加拡張機能についての修復コードを探し出し得るか又は提案し得る。これは、前記階層構造を上位に向かって上がっていき、より上位のアーチファクトとマッチする欠陥についての修復コードを探し出すことにより行うことを含む。

【００６６】

図４は、ソフトウェアファイルについてのアーチファクトのコーパスを提供するシステムの例示的な一実施形態を示すブロック図である。例示的な一実施形態は、複数のソフトウェアファイルを有するソース４３０と通信することが可能なインターフェース４２０を備え得る。このインターフェース４２０は、ローカルソース４３０と通信可能に接続され得る。一部の実施形態において、ローカルソース４３０は、ローカルハードドライブ又はディスクである。他の実施形態において、インターフェース４２０は、パブリックネットワーク又はプライベートネットワークを介してファイルを得るネットワークインターフェース４２０であり得る。これらのソフトウェアファイルのパブリックソース４３０には、例えば、ＧｉｔＨＵＢ、ＳｏｕｒｃｅＦｏｒｇｅ、ＢｉｔＢｕｃｋｅｔ、ＧｏｏｇｌｅＣｏｄｅ、共通脆弱性識別子システム等が含まれる。プライベートソースには、例えば、会

社の内部ネットワークと当該内部ネットワークに記憶されたファイル（共有ネットワークドライブおよび私設レポジトリを含む）とが含まれる。例示的なこのシステムは、さらに、ソース４３０から複数のソフトウェアファイルを得るようにインターフェース４２０に接続された少なくとも１つのプロセッサ４１０を備える。また、プロセッサ４１０は、複数のソフトウェアファイルのそれぞれについての複数のアーチファクトを決定するように用いられ得る。これらのアーチファクトは、静的アーチファクトおよび／または動的アーチファクトおよび／または導出アーチファクトおよび／またはメタデータアーチファクトであり得る。また、他の実施形態において、プロセッサ４１０は、ソフトウェアファイルのそれぞれを中間表現に変換するように、かつ、当該中間表現からアーチファクトを決定するように構成され得る。

【００６７】

例示的なこのシステムは、さらに、ソフトウェアファイルのそれぞれについてのアーチファクトを記憶する少なくとも１つの記憶装置４４０ａ～４４０ｎであって、プロセッサ４１０に接続された少なくとも１つの記憶装置４４０ａ～４４０ｎを備える。これらの記憶装置４４０ａ～４４０ｎは、ハードドライブ、ハードドライブのアレイ、他の種類の記憶装置、および分散ストレージ（例えば、Ｈａｄｏｏｐファイルシステム（ＨＤＦＳ）においてＴｉｔａｎおよびＣａｓｓａｎｄｒａを用いることにより提供されるもの）とされ得る。同様に、例示的なこのシステムは、単一のプロセッサ４１０を備えるものであってもよいし、分散処理を用いて複数のプロセッサ４１０を備えるものとされてもよい。また、さらなる他の実施形態は、インターフェース４２０と記憶装置４４０ａ～４４０ｎとの間を直接通信可能に接続することを提供する。

【００６８】

図５は、デザインパターンを探し出す方法の例示的な一実施形態を示すブロック図である。デザインパターンは、例えば、バグ、修復、脆弱性、セキュリティパッチ、プロトコル、プロトコル拡張、機能、付加拡張機能を含む。それぞれのデザインパターンは、ソフトウェアプロジェクト階層構造におけるさまざまな階位で抽出されるアーチファクト（例えば、仕様（specifications）、ＣＧ、ＣＦＧ、Ｄｅｆ－Ｕｓｅチェイン、命令のシーケンス、型、定数）と関連付けられ得る。

【００６９】

例示的なこの方法は、複数のソフトウェアファイルに対応する複数のアーチファクトを有するデータベースにアクセスする工程を備える（符号５１０）。このデータベースは、グラフデータベース、関係データベースまたはフラットファイルであり得る。このデータベースは、プライベートネットワークにおいて局所的に位置してもよいし、インターネット又はクラウドを介して利用可能なものであってもよい。この方法は、ひとたび前記データベースにアクセスすると、複数のファイルのうちの第１のファイルについての、前記複数のアーチファクトのうちの少なくとも１つに基づいて、デザインパターンを自動的に特定し得る（符号５２０）。例示的な一部の実施形態において、前記複数のアーチファクトのそれぞれは、静的アーチファクト、動的アーチファクト、導出アーチファクトまたはメタデータアーチファクトであり得る。他の実施形態は、異なる種類のアーチファクトの組合せを有し得る。また、前記ファイルのフォーマットに制限はなく、例えば、バイナリコードフォーマット、ソースコードフォーマット、中間表現（ＩＲ）フォーマット等であり得る。

【００７０】

一部の実施形態において、前記デザインパターンは、開発中アーチファクトのキーワード検索又は自然言語検索により特定され得る。例えば、ソースコードの所与の改変におけるインラインコードコメントは、見つけ出されて修正された欠陥を特定するものとなり得る。これらのコメントは、欠陥、バグ、エラー、問題、不具合、または誤作動などの単語を使用し得る。これらの単語を、メタデータのキーワード検索に利用することが可能であり得る。また、コミットログは、新しい改変やパッチが適用された理由（例えば、欠陥に対処するため、または機能を向上させる）を記述するテキストを含み得る。また、訓練や

フィードバックをこの検索に適用して検索結果を改良するようにしてもよい。

【００７１】

例示的な他の実施形態は、テキストにおける共通脆弱性及びエラーを特定して且つ欠陥及び利用可能な修復があれば当該修復を記述し得るＣＶＥソースから、開発中アーチファクトを検索し得る。このテキストが、アーチファクトとして得られてデータベースに記憶されてもよい。また、一部のソースは、欠陥をコード化して且つどのファイルが欠陥を含むのかを探し出すのにコードをキーワードとして用い得る。また、アーチファクトのソースが何であるのかが、ソフトウェアファイルの特定にあたって考慮されて重み付けされ得る。例えば、ＣＶＥソースは、出所又はインラインコメントのないレポジトリよりも、欠陥を特定するのにあたって信頼性が高くなり得る。さらなる他の実施形態は、ファイル名、改変回数などのメタデータアーチファクトを用いて少なくとも暫定的にソフトウェアファイルを特定してもよく、ＣＧやＣＦＧなどのマッチする追加のアーチファクトに基づいてその特定を確定してもよい。

【００７２】

本発明の一部の実施形態は、例示的なこの方法を実行して、一部、ほとんど、又は全てのソースコードファイル及びＬＬＶＭ ＩＲファイルについてのデザインパターンを特定することを試みる。また、一部の実施形態は、ファイルがコーパスに追加されるたびに、データベースにアクセスして任意のデザインパターンを特定することを試みる。また、一部の実施形態は、特定されたデザインパターンを、後の使用のためにラベル付けし得る。

【００７３】

また、一部の実施形態は、ソースコード又はファイルと関連付けられたＬＬＶＭ ＩＲにおける欠陥の位置であって、データベースに記憶された位置を見つけ出す。例えば、開発中アーチファクトは、ソースコードのどこに欠陥が存在するのか及びパッチのどこに修復が存在するのかを示し得る。また、ソースコード又はＬＬＶＭ ＩＲは、欠陥を有するファイル及び当該ファイルのより新しい修復後バージョンと、違いを取り出してどこに欠陥及び修復があるのかを判別するために分析および比較され得る。また、一部の実施形態では、開発中アーチファクトにおいて特定された欠陥の種類を用いて、その欠陥の位置について、コードの検索が絞り込まれ得る。また、他の実施形態は、ラベルなどを用いてデザインパターンを識別可能とし得て、かつ、この識別子をファイルについてのデータベースに記憶し得る。これにより、所与の欠陥又は欠陥の種類について、データベースを容易に検索することが可能となる。このようなラベルには、例えば、ソフトウェアファイルについての開発中アーチファクトまたはソースコードから得られた文字列等が含まれる。これと同じアプローチが、機能や付加拡張機能を特定してこれらにラベル付けすることにも適用可能である。

【００７４】

例示的な一部の実施形態において、デザインパターンは、ソフトウェアファイル内に存在する。例示的な一部の実施形態において、デザインパターンは、ファイル間の相互作用（例えば、インターフェース等）に関するものであり得る。例示的な実施形態は、デザインパターンを自動的に特定することを、その特定を、複数のソフトウェアファイル（例えば、いずれも同じソフトウェアプロジェクトに属する第１および第２のファイル）についてのアーチファクトに基づかせることによって行い得る。例えば、インターフェース不一致エラーなどのデザインパターンを表す予め特定されたパターンが、前記第１および第２のファイルからのアーチファクトを用いて当該ファイルについてのインターフェースエラーが存在することを特定可能なようにデータベース又はその他に記憶され得る。例示的な実施形態において、デザインパターンは、例えば、欠陥、修復、機能、付加拡張機能、または予め特定されたプログラム断片を含む。

【００７５】

例示的な一部の実施形態において、この方法は、アーチファクトにおいて、欠陥又は修復を表す文字列を探し出す。このような列（例えば、バグ、エラー、欠陥）、修復に関する列、さらには、コード内のどこにそのような列を見つけ出すことができるのかが、開発

中アーチファクトにしばしば存在する。また、これらの開発中アーチファクトは、機能又は付加拡張機能を表す列を有し得る。

【００７６】

例示的な一部の実施形態において、デザインパターンは、当該デザインパターンを表す予め特定されたパターンに基づくものである。これらの予め特定されたパターンは、ユーザにより作成されたものであってもよいし、本明細書の開示内容に関連する方法により予め特定されたものであってもよいし、他の何らかの方法で特定されたものであってもよい。これらの予め特定されたパターンは、欠陥、修復、機能、付加拡張機能、関心のあるもの、または他の重要なものに対応し得る。

【００７７】

図６は、欠陥を探し出す方法の例示的な一実施形態を示すフロー図である。この方法は、複数のソフトウェアファイルに対応する複数のソフトウェアアーチファクトを有するデータベース（例えば、コーパス）にアクセスする工程を備える（符号６１０）。次に、これらのアーチファクトが、その大量のデータからパターンを判別するように分析される。例えば、この分析は、前記複数のアーチファクトをクラスタ化することを含み得る（符号６２０）。データをクラスタ化することにより、既知の欠陥を含むことが知られていないファイルにおける当該既知の欠陥を見つけ出すことが可能となる。つまり、例示的なこの方法は、前記クラスタ化から、これまで特定されていなかった欠陥を、少なくとも１つの予め特定された欠陥に基づいて特定し得る（符号６３０）。

【００７８】

本発明の例示的な一部の実施形態は、コーパスに機械学習を用い得る。機械学習は、データ内の関連特徴を捉えるにあたって下位のアーチファクトから始めていってより複雑な表現を構築することにより、そのデータの階層構造を学習することに関するものである。例示的な一部の実施形態は、コーパスに深層学習を用い得る。深層学習は、データの表現を学習することに基づく機械学習手法の、広義のファミリーのなかの一部である。一部の実施形態では、クラスタ化のためにオートエンコーダが用いられ得る。

【００７９】

例示的な一部の実施形態において、前記アーチファクトは、ラベル付けされていないグラフアーチファクト及びドキュメントアーチファクトのコンパクトな表現を自動的に見つけ出すように、オートエンコーダのセットにより処理され得る。グラフアーチファクトは、ＣＧ、ＣＦＧ、ＵＤチェイン、ＤＵチェイン、ＤＴなどの、グラフ形式で表現可能なアーチファクトを含む。そして、これらグラフアーチファクトのコンパクトな表現が、ソフトウェアデザインパターンを見つけ出すようにクラスタ化され得る。対応するメタデータアーチファクトから抽出された知識を用いて、デザインパターンをラベル付け（例えば、バグ、修正、脆弱性、セキュリティパッチ、プロトコル、プロトコル拡張、機能、付加拡張機能）するようにしてもよい。

【００８０】

例示的な一部の実施形態において、前記オートエンコーダは、ベクトルを入力として共通の特徴を抽出し得る構造化スパースオートエンコーダ（ＳＳＡＥ）である。一部の実施形態では、プログラムの特徴を自動的に見つけ出すために、まず、抽出されたグラフアーチファクトが行列形式で表現される。抽出されるアーチファクトの多く（例えば、ＣＦＧ、ＵＤチェイン、ＤＵチェインを含む）は、隣接行列として表現可能である。構造的特徴は、ソフトウェアファイル・プロジェクト階層構造における各々の階位で学習可能であり得る。

【００８１】

グラフアーチファクトにおけるノードの数は、幅広く異なり得る。したがって、中間アーチファクトが、深層学習の入力として提供され得る。このような中間アーチファクトの一つは、グラフラプラシアンの最初のｋ個の固有値であり、深層学習がスペクトルクラスタ化と同様の処理を実行することを可能にする。他の中間アーチファクトは、グラフにおけるノード同士が互いにクラスタを形成する傾向についての度合いの尺度を提供するクラ

スタリング係数（例えば、グローバルクラスタリング係数、ネットワーク平均クラスタリング係数、推移性の比率）を含む。さらなる他の中間アーチファクトは、グラフの密度の尺度である、当該グラフの樹相度である。エッジが多いグラフは樹相度が大きく、樹相度が大きいグラフは高密度のサブグラフを有する。さらなる他の中間アーチファクトは、グラフがボトルネックを有するか否かの数値尺度である等周数（isoperimetric number）である。これらの中間アーチファクトは、グラフの構造の様々な側面を、機械学習手法への使用のために捉える。

【００８２】

機械学習（例示的な実施形態での深層学習も含む）は、単純なオートエンコーダ構造から始まるマルチステッププロセスを用いて且つこのアプローチを反復的に改良させて前記ＳＳＡＥを発展させるように訓練される、アルゴリズムを用い得る。また、前記ＳＳＡＥは、中間アーチファクトから特徴を学習するように訓練され得る。オートエンコーダは、ラベル付けされていないデータのコンパクトな表現を学習する。これは、少なくとも１つの隠された層からなり且つ恒等関数の近似を学習する同数の入力と出力とを有するニューラルネットワークによってモデル化可能である。オートエンコーダは、入力信号を記述パラメータの本質的な集合へと脱水（エンコード）し、かつ、これらの信号を元来の信号を再生成するように再水和（デコード）する。それらの記述パラメータは、全ての訓練信号にわたって再水和を最適化するように訓練時に自動的に選択され得る。脱水後の信号の本質的性質は、信号をクラスタへとグループ分けする際の基礎となる。

【００８３】

オートエンコーダは、入力信号を低次元の特徴空間へとマッピングすることにより、当該入力信号の次元を減少させ得る。例示的な実施形態は、次に、オートエンコーダにより見つけ出された特徴空間において、コードのクラスタ化および分類を実行し得る。ｋ平均法アルゴリズムは、学習された特徴をクラスタ化する。ｋ平均法アルゴリズムは、特徴を、もたらされるクラスタ平均を最小化するｋ個のクラスタへと分ける反復的洗練化（反復的改良）手法である。最初のクラスタの数ｋは、抽出されたトピックの数に基づいて選択され得る。この数のクラスタ候補にわたって検索を行い、多数の異なるｋのそれぞれについて新しい結果を算出することは、ｋ平均法の演算計量がユークリッド距離に基づいていることから極めて効率的である。例示的な実施形態は、クラスタ化された特徴が導き出されたソフトウェアファイル内において最も頻繁に出現するトピックのラベルを用いて、得られたクラスタを分類し得る。

【００８４】

特徴ベクトルはスパース（疎）で且つコンパクトであっても、特徴ベクトルを調べるだけでは入力ベクトルを理解することが困難であり得る。よって、例示的な実施形態は、予め学習された重みパラメータに関連付けられた事前情報（prior）を利用し得る。十分なコーパスであれば、「修復済み」コード等についての、パラメータ空間におけるパターンが出現するはずである。例示的な実施形態は、その時点までに集められたデータセットにより与えられる事前情報を用いて、特定のパターンをオートエンコーダに組み込み得る。具体的に述べると、例示的な実施形態は、ラベルがシステムにより学習されるたびに、その情報をオートエンコーダ処理に組み込み得る。

【００８５】

例示的な実施形態は、データベース管理（例えば、結合、フィルタ）と解析演算（例えば、特異値分解（ＳＶＤ）、バイクラスタリング）との組合せを用い得る。例示的な実施形態のグラフ理論（例えば、スペクトルクラスタリング）と機械学習アルゴリズム又は深層学習アルゴリズムとは、いずれも、特徴抽出のために同様のアルゴリズムプリミティブを用い得る。また、ＳＶＤを用いて、学習アルゴリズムの入力データのノイズを除去したり、より少ない次元を用いてデータを近似することによってデータ削減を実行したりしてもよい。

【００８６】

例示的な実施形態は、時間をかけて且つ複数のプログラムにわたって、コード状態につ

いてのヒトの理解を、ドキュメントアーチファクトの教師なしの意味ラベル生成（テキストアナリティクス（テキスト解析）によるものも含む）を介してカプセル化し得る。テキストアナリティクスの一例として、潜在的ディリクレ配分法（ＬＤＡ）が挙げられる。意味論的情報は、ＬＤＡおよびトピックモデリングを用いて、ドキュメントアーチファクトから抽出され得る。これらのアプローチは、単語又はフレーズの出現にその並び方を無視して注目する「ｂａｇ－ｏｆ－ｗｏｒｄｓ」手法である。例えば、「科学技術計算」を表すｂａｇは、「ＦＥＴ」、「ウェーブレット」、「ｓｉｎ」，「ａｔａｎ」などのシード用語を有するかもしれない。例示的な実施形態は、ソースコメント、ＣＧ／ＣＦＧノードラベル、コミットメッセージなどの、ソースからの抽出されたドキュメントアーチファクトを、用語の出現を計数することによって「ｂａｇ」を満たすように用い得る。得られる決まったビンヒストグラムが、テキスト用途に適した深層学習アルゴリズムの一応用である制限付きボルツマンマシン（ＲＢＭ）に与えられ得る。抽出されたトピックは、抽出されたドキュメントアーチファクトに関連付けられた意味論的情報を捉えて、かつ、オートエンコーダによるグラフアーチファクトの教師なし学習により形成されるクラスタについてのラベル（例えば、バグ／修正、脆弱性／パッチ等）として機能し得る。例示的な他の実施形態によって用いることが可能な他の形態のテキストアナリティクスには、自然言語処理、字句解析および予測解析が含まれる。

【００８７】

ドキュメントアーチファクトから抽出されたトピックラベルは、オートエンコーダの構造を教えるためにラベル付け情報を提供し得る。例示的な実施形態は、コーパスデータベースを、学習したトピックに基づいて、訓練データのかたまりについて、すなわち、順序的ソフトウェアパターン（つまり、ソフトウェア改変の前後）を表す意味的な共通性についてクエリし（検索し）得る。これらのパターンは、長期間のソフトウェア開発ライフサイクルにまつわるコミットログ、変更ログ、コメントなどのソフトウェア開発中ファイルに埋まっている変更点を捉え得る。それら変更点の連携が、バグ／修正、脆弱性／セキュリティパッチ、機能／付加拡張機能などの検出及び修復に関連した、ソフトウェアの進化についての知見を提供する。また、この情報は、アーチファクトコーパスから自動的に抽出された知識を理解しラベル付けするのに用いられ得る。

【００８８】

図７は、本発明の一実施形態における、デザインパターンを特定するためのアーチファクトのクラスタ化を示すブロック図である。構造的特徴は、ソフトウェアファイル階層におけるそれぞれの階位（システム、プログラム、関数、およびブロック７１０を含む）で学習可能であり得る。ＣＧ、ＣＦＧ、ＤＴなどのグラフアーチファクトが、クラスタ化７１５のために解析可能であり得る。これらのグラフアーチファクトは、グラフ不変量特徴７２０に変換され得る。そして、これらのグラフ特徴７４０は、オートエンコーダなどのグラフ解析モジュール７６０への入力として提供され得て、得られたクラスタ化は、類似のデザインパターンについて調べられ、これら類似のデザインパターンが、一緒にクラスタ化される（符号７８０）。ソースコードファイルからの又は開発中アーチファクトからの少なくとも１つの文字列などのテキストが、ラベル７３０にマッピングされ得る。これらのラベル７５０が、テキストアナリティクス（テキスト解析）モジュール７７０によりＬＤＡ又は他の自然言語処理を用いるなどして分析され得て、それらのラベルは、当該ラベルが導き出された対応する見つけ出されたクラスタ７８０に関連付けられ得る。これらのモジュール７６０，７７０は、ソフトウェア、ハードウェアまたはこれらの組合せにより実現可能である。

【００８９】

図８は、コーパスを用いてソフトウェアを特定する方法の例示的な一実施形態を示すフロー図である。例示的なこの実施形態は、ソフトウェアファイルを得る（符号８１０）。このファイルは、パブリックソース又はプライベートソースから（例えば、インターネット介した公開レポジトリ、クラウドまたは民間企業のサーバから）ネットワークインターフェースを介して得られるものとされ得る。また、例示的な一部の実施形態は、ローカル

ハードディスク、持ち運び可能なハードドライブ、持ち運び可能なハードディスクなどのローカルソースから前記ソフトウェアファイルを得ることができる。例示的な実施形態は、前記ソースから単一のファイル又は複数のファイルを得ることができ、かつ、スクリプト言語を用いるなどして自動的にこれを行い得るか、あるいは、ユーザが干渉することで人的にこれを行い得る。例示的なこの方法は、次に、前記ソフトウェアファイルについての複数のアーチファクト（例えば、本明細書で説明する他のアーチファクトのうちの任意のアーチファクト）を決定し得る（符号８２０）。例示的なこの方法は、次に、複数の参照ソフトウェアファイルのそれぞれについての複数の参照アーチファクトを記憶するデータベースにアクセスし得る（符号８３０）。前記参照アーチファクトは、コーパスデータベースに記憶されたものであってもよい。例示的な一部の実施形態において、これらの参照ファイルは、予め得られたものであるソフトウェアファイルであって、自身のアーチファクトが前記データベースにおいて記憶済みである（一部の実施形態では、当該ソフトウェアファイルと共に前記データベースにおいて記憶済みである）ソフトウェアファイルを含み得る。得られた前記ソフトウェアファイルについての決定された前記アーチファクト又は当該アーチファクトの複数の部分集合が、前記データベースに記憶された前記参照アーチファクト又は当該参照アーチファクトの複数の部分集合と比較される（符号８４０）。例示的な実施形態は、前記複数のアーチファクトとマッチする前記複数の参照アーチファクトを有する前記参照ソフトウェアファイルを特定することにより、前記ソフトウェアファイルを特定し得る（符号８５０）。前記ソフトウェアファイルと前記参照ソフトウェアファイルとが同じファイルであると特定される理由は、比較された前記アーチファクトと前記参照アーチファクトとがマッチ（一致）するからである。

【００９０】

また、その後、正確な特定がなされたという信頼度を増加させるように、追加のアーチファクト又はコードにおける追加の部位が比較され得る。信頼度は、固定されるか又は調節可能とされ得る。信頼度は、マッチするアーチファクトの数、どのアーチファクトがマッチするのか、マッチする数とどのアーチファクトがマッチするのかとの組合せなどの多種多様な条件に基づくものであり得る。このような調節は、例えば、特定のデータセットおよび当該特定のデータセットの観察結果について行われ得る。また、一部の実施形態では、マッチングがファジーマッチングを含み得る。このファジーマッチングは、例えば、１００％未満の、マッチと称するためのマッチング率の、調節可能な設定を有する。

【００９１】

例示的な一部の実施形態では、特定のアーチファクトに、マッチ・特定プロセスにおいてより大きいか又はより小さい重みが与えられ得る。例えば、命令が３２ビットプロセッサと対応付けられたものであるか、それとも、６４ビットプロセッサと対応付けられたものであるか等の共通するアーチファクトには、ゼロの重みか又は他の何らかの小さい重みが与えられ得る。一部のアーチファクトは変換後も多かれ少なかれ不変であり、例示的な一部の実施形態では、それに応じてこれらのアーチファクトの重みが調節され得る。例えば、ファイル名またはＣＧアーチファクトは、ファイルの正体（アイデンティティ）を明らかにするのに極めて有益と見なされ得る一方で、ＬＴＳ、ＤＴなどの一部のアーチファクトは、有用な手がかりではないと見なされ得て、例示的な一部の実施形態およびソースではより小さい重みを与えられ得る。他の実施形態は、比較を行ったときにマッチを特定するうえで、所与の組合せのアーチファクトにより大きい重みを与え得る。例えば、特定を行うときには、基本ブロックアーチファクトやＤＴアーチファクトのマッチよりもＣＦＧアーチファクトやＣＧアーチファクトのマッチを有するほうにより大きい重みが与えられ得る。同様に、ファイルの特定を行うときには、所与のアーチファクトがマッチしていないことにより大きいか又はより小さい重みが与えられ得る。特定プロセスにおける重み付けを評価する他の例は、特定閾値を、マッチするアーチファクトを百分率で表したもの又は他の何らかの尺度などで表現することを含み得る。他の実施形態は、前記特定閾値を変化させ得る。これは、前記特定閾値を、ファイルのソース、ファイルの種類、タイムスタンプ（ファイルの日付を含む）、ファイルのサイズ、ファイルについて所与のアーチフ

ァクトを決定できないか又はそのようなアーチファクトが存在しないことなどに基づいて変化させることを含む。

【００９２】

他の実施形態は、前記ソフトウェアファイルについての前記複数のアーチファクトのうちの一部を、当該ソフトウェアファイルをＬＬＶＭ ＩＲなどの中間表現に変換すること、および当該中間表現から前記複数のアーチファクトのうちの少なくとも１つを決定することによって決定し得る。さらなる他の実施形態は、前記複数のアーチファクトのうちの一部を、前記ソフトウェアファイル（例えば、ソースコードファイル、マニュアルファイル）から文字列を抽出することによって決定し得る。

【００９３】

また、例示的な実施形態は、前記ソフトウェアファイルのより新しいバージョンが存在するか否かを、前記参照アーチファクトのうちの、特定された前記参照ソフトウェアファイルと対応付けられた少なくとも１つを分析することによって判定することを含み得る。例えば、ソフトウェアファイルが特定されると、データベースが、そのソフトウェアファイルのより新しい改変が利用可能であるか否かを調べるために確認され得る。これは、例えば、対応する参照ファイルの改変番号又はタイムスタンプを確認することによって、あるいは、アーチファクトやファイルと関連付けられた、その参照ファイルが他のファイルのより古いバージョンであることを特定可能な、データベース内のラベルを確認すること等によって行われ得る。また、例示的な他の実施形態は、前記ソフトウェアファイルの前記より新しいバージョンを自動的に提供し得る。これは、ユーザ、パブリックソースまたはプライベートソースに提供することを含む。

【００９４】

他の一部の実施形態は、前記ソフトウェアファイルについてのパッチが存在するか否かを、前記参照アーチファクトのうちの、特定された前記参照ソフトウェアファイルと対応付けられた少なくとも１つを分析することによって判定し得る。例えば、例示的な実施形態は、前記参照ソフトウェアファイルと対応付けられたアーチファクトを調べ得て、かつ、当該ファイルについてのパッチが存在することを判定し得る。このパッチには、当該ソフトウェアファイルに未だ適用されていないパッチが含まれる。他の実施形態は、前記パッチを前記ソフトウェアファイルに自動的に適用し得るか又は前記パッチを適用したいか否かをユーザに尋ね得る。

【００９５】

他の一部の実施形態は、前記パッチを（一部の実施形態では、前記ソフトウェアファイル（あるいは、前記ソフトウェアファイルと前記参照ソフトウェアファイルとはマッチしているので、その参照ソフトウェアファイル）も）、当該パッチのうちの、前記ソフトウェアファイルにおける欠陥の修復に対応する修復部を決定するように分析し得る。一部の実施形態において、この分析は、前記ソフトウェアファイルが得られる前に又は前記ソフトウェアファイルが得られた後に生じ得る。他の実施形態は、前記パッチのうちの前記修復部のみを前記ソフトウェアファイルに適用し得る。これは、自動的に行われても、あるいは、前記パッチのうちの前記修復部を適用したいか否かをユーザに尋ねてもよい。他の実施形態は、前記パッチのうちの前記修復部を、その修復部がソースにおいて適用されるように当該ソースへと提供し得る。また、前記パッチおよび前記ソフトウェアファイルの分析は、これらパッチおよびソフトウェアファイルを中間表現に変換すること、および当該中間表現から前記複数のアーチファクトのうちの少なくとも１つを決定することを含み得る。同様に、他の実施形態は、前記パッチおよび前記ソフトウェアファイル（あるいは、前記ソフトウェアファイルと前記参照ソフトウェアファイルとはマッチしているので、その参照ソフトウェアファイル）を、当該パッチのうちの、前記ソフトウェアファイルにおける機能の向上または変更に対応する付加拡張機能部を決定するように分析し得る。他の実施形態は、前記パッチのうちの前記付加拡張機能部のみを前記ソフトウェアファイルに適用し得る。これは、自動的に行われても、あるいは、前記パッチのうちの前記付加拡張機能部を適用したいか否かをユーザに尋ねてもよい。

【００９６】
例示的な他の実施形態は、前記ソフトウェアファイルにおいて欠陥が存在するか否かを、前記参照アーチファクトのうちの、特定された前記参照ソフトウェアファイルと対応付けられた少なくとも１つを分析することによって判定し得る。例えば、前記参照ソフトウェアファイルは、あるアーチファクトであって、修復が利用可能である欠陥を自身が有することを示すアーチファクトを有し得る。他の実施形態は、前記ソフトウェアファイルにおける前記欠陥を自動的に修復し得る。これは、ソースコードのブロックをソースコードの修復ブロックに自動的に置き換えること、あるいは、前記ソフトウェアファイルにおける中間表現のブロックを中間表現の修復ブロックに自動的に置き換えることを含む。他の実施形態は、バイナリファイルにおける前記欠陥を、当該バイナリの一部をバイナリパッチに置き換えることによって修復し得る。一部の実施形態では、修復済みのファイルが、前記ソフトウェアファイルのソースへと送られ得る。他の実施形態は、修復コードを、前記ソフトウェアファイルが当該ソフトウェアファイルのソースにおいて修復されるように当該ソースへと提供し得る。

【００９７】
図９は、コードを特定する方法の例示的な一実施形態を示すフロー図である。例示的なこの方法は、少なくとも１つのソフトウェアファイルを得ることができる（符号９１０）。これらソフトウェアファイルについての複数のアーチファクトが決定され得る（符号９２０）。一部の実施形態は、前記アーチファクトが既に決定されている場合、当該アーチファクトを決定するのではなく当該アーチファクトを得る。複数の参照アーチファクトを記憶するデータベースがアクセスされ得る（符号９３０）。これら参照アーチファクトは、本明細書で説明するアーチファクトであり、かつ、参照ソフトウェアファイル、参照デザインパターン、または対象のコードにおける他のブロックに対応し得る。前記データベースは、局所的ドライブ、ネットワークドライブ、インターネット又はクラウドを介してアクセス可能な場所などの様々な場所に記憶され得て、かつ、複数の記憶装置にわたって分散され得る。そして、前記少なくとも１つのソフトウェアファイル内に存在するプログラム断片または前記少なくとも１つのソフトウェアファイルに関連付けられたプログラム断片（例えば、インターフェースバグ）は、当該プログラム断片に対応する前記複数のアーチファクトと当該プログラム断片に対応する前記複数の参照アーチファクトとを照合することによって特定され得る（符号９４０）。プログラム断片とは、ファイルの一部位、プログラムの一部位、基本ブロックの一部位、関数の一部位、または関数間のインターフェースの一部位である。プログラム断片は、最小では単一の命令、最大ではファイル全体、プログラム全体、基本ブロック全体、関数全体またはインターフェース全体になり得る。選ばれる部位は、プログラム断片を所望の任意の信頼度をもって特定するのに十分なものとされ得る。一部の実施形態において、この信頼度は、決まっているか又は調節可能である。この信頼度は、例えばファイルを特定する場合に関して既述したような方法で変化するものとされ得る。

【００９８】
一部の実施形態において、前記ソフトウェアファイルについてのアーチファクトを決定することは、前記ソフトウェアファイルを中間表現に変換すること、および当該中間表現から前記アーチファクトのうちの少なくとも１つを決定することを含む。一部の実施形態において、前記ソフトウェアファイルおよび前記参照ソフトウェアファイルは、いずれもソースコードフォーマットであるか又はそれぞれバイナリコードフォーマットである。他の実施形態において、前記プログラム断片は、前記ソフトウェアファイルにおける欠陥に対応するものであり、当該欠陥に対応させるために、前記データベースにおいて特定済みである。他の実施形態は、前記ソフトウェアファイルにおける前記欠陥を自動的に修復し得るか又は前記欠陥を修復するための少なくとも１つの修復選択肢をユーザに提示し得る。一部の実施形態は、修復選択肢を順序付けし得る。これは、例えば、前記ユーザにより選択された過去の少なくとも１つの修復選択肢に基づいて行われること、または、前記修復選択肢についての成功の確率に基づいて行われることを含む。

【００９９】
図１０は、本発明の一実施形態における、ソフトウェアファイルのデータベースコーパスを用いるシステムを示すブロック図である。例示的なこのシステムは、少なくとも１つのソフトウェアファイルを有するソース１０１０と通信することが可能なインターフェース１０２０を備える。インターフェース１０２０は、プロセッサ１０３０にも通信可能に接続されている。他の実施形態において、インターフェース１０２０は、記憶装置１０４０にも直接接続され得る。この記憶装置１０４０は、幅広い種類の周知の記憶装置又はシステムのうちの、どのような記憶装置又はシステムであってもよい。そのような記憶装置又はシステムとして、例えば、ネットワークストレージデバイス、ローカルストレージデバイスが挙げられ、これらは、例えば、単一のハードドライブであったり、複数のハードドライブを備えた分散ストレージシステムであったりし得る。記憶装置１０４０は、参照アーチファクト（複数の参照ソフトウェアファイルのそれぞれについての参照アーチファクトを含む）を記憶し得て、かつ、プロセッサ１０３０に通信可能に接続され得る。プロセッサ１０３０は、ソフトウェアファイルがソース１０１０から取得されるように構成され得る。このソフトウェアファイルの正体、当該ファイルのより新しいバージョンが存在するか否か、パッチが存在するか否か、当該ファイルが欠陥又は未向上の機能を含んでいるか否かなどが、例示的なこのシステムが取り組むことのできる問題の例である。プロセッサ１０３０は、さらに、前記ソフトウェアファイルについての複数のアーチファクトを決定するように、かつ、記憶装置１０４０内の前記参照アーチファクトにアクセスするように、かつ、前記ソフトウェアファイルについての前記アーチファクトを記憶装置１０４０内に記憶された前記参照アーチファクトと比較するように、かつ、前記ソフトウェアファイルについての比較された前記アーチファクトに対応する前記参照アーチファクトを有する前記参照ソフトウェアファイルを特定することにより、前記ソフトウェアファイルを特定するように構成されている。

【０１００】
例示的なこのシステムの他の実施形態において、プロセッサ１０３０は、パッチを前記ソフトウェアファイルに、当該ファイルについてのそのようなパッチが記憶装置１０４０に存在する場合、自動的に適用するように構成され得る。また、さらなる他の実施形態において、前記プロセッサは、特定されたパッチおよび前記ソフトウェアファイルを、そのパッチのうちの修復部であって、当該ソフトウェアファイルにおける欠陥の修復に対応する修復部が存在するか否かを判定するように分析するように、かつ、それが存在する場合には、前記パッチのうちのその修復部のみを前記ソフトウェアファイルに自動的に適用するように又はユーザに尋ねるように構成され得る。

【０１０１】
図１０は、本発明の一実施形態における、データベースコーパスを用いる例示的な別のシステムを示しているとも言える。例示的なこの別のシステムは、少なくとも１つのソフトウェアファイルを有するソース１０１０と通信することが可能なインターフェース１０２０を備える。インターフェース１０２０は、プロセッサ１０３０にも通信可能に接続されている。他の実施形態において、インターフェース１０２０は、記憶装置１０４０にも直接接続され得る。この記憶装置１０４０は、幅広い種類の周知の記憶装置又はシステムのうちの、どのような記憶装置又はシステムであってもよい。そのような記憶装置又はシステムとして、例えば、ネットワークストレージデバイス、ローカルストレージデバイスが挙げられ、これらは、例えば、単一のハードドライブであったり、複数のハードドライブを備えた分散ストレージシステムであったりし得る。記憶装置１０４０は、参照アーチファクトを記憶し得て、かつ、プロセッサ１０３０に通信可能に接続され得る。プロセッサ１０３０は、少なくとも１つのソフトウェアファイルが取得されるように、前記少なくとも１つのソフトウェアファイルについての複数のアーチファクトを決定するように、複数の参照アーチファクトを記憶するデータベースにアクセスするように、前記少なくとも１つのソフトウェアファイルについてのプログラム断片を、当該プログラム断片に対応する前記複数のアーチファクトと当該プログラム断片に対応する前記複数の参照アーチファ

クトとを照合することによって特定するように構成され得る。例示的な一部の実施形態において、前記プログラム断片は、欠陥に対応させるために、前記データベースにおいて特定済みである。そのような欠陥は、例えば、バグ、セキュリティ脆弱性、プロトコル不備を含む。これらの欠陥は、前記少なくとも１つのソフトウェアファイル内のものであり得るか、あるいは、前記ソフトウェアファイル間の少なくとも１つのインターフェースに関するものであり得る¥。また、他の実施形態は、前記プロセッサを備え、当該プロセッサは、前記少なくとも１つのソフトウェアファイルにおける前記欠陥を自動的に修復するように構成され得る。例示的な一部の実施形態において、前記プログラム断片は、機能に対応させるために、前記データベースにおいて特定済みであり、また、一部の実施形態は、付加拡張機能（ソースコード又はバイナリファイルについてのパッチの形態のものを含む）を自動的に提供し得る。

【０１０２】

＜修復＞

例示的な実施形態は、自動修復のためのプログラム合成を支援する。これは、ＣＧノード（関数）を置き換えること、ＣＦＧノード（基本ブロック）を置き換えること、特定の命令を置き換えること、あるいは、選ばれた修復をインスタンス化するように特定の変数及び定数を置き換えることを含む。これらの要素（例えば、関数、基本ブロック、命令等）は、互換性があるインターフェース（つまり、同数のパラメータ、同数の型および同数の出力）を有する要素とスワップ可能であり、ＬＬＶＭ ＩＲの欠陥ブロックをＬＬＶＭ ＩＲの修復ブロックに置き換えることによってＬＬＶＭ ＩＲを変換することが可能である。

【０１０３】

また、一部の実施形態は、基本ブロックを関数呼出しとスワップすること、および関数呼出しを少なくとも１つの基本ブロックとスワップすることを選択し得る。一部の実施形態は、ソースコードおよびバイナリをパッチし得る。また、他の実施形態は、スワップのための適切な要素を、そのような要素が存在しない場合に生成し得る。上位のアーチファクト（例えば、ＬＴＳ、Ｚ言語の述語（Z predicate））を用いて、ソフトウェアパッチに適合する実装が導出されてもよい。例示的な実施形態は、抽出されたグラフ表現の階層構造を利用し、まず修復パターンの適切な表現へとその階層構造を上った後、（コンパイルを介して）具体的な実装へとその階層構造を下り得る。アーチファクトの階層的性質は、修復コードを作成するのに役立つものとなり得る。

【０１０４】

例示的な実施形態は、ユーザがターゲットプログラム（ソース又はバイナリ）を投入（登録）することを可能にし得て、例示的な実施形態は、あらゆる欠陥デザインパターンの存在を見つけ出す。それぞれの欠陥について、修復戦略（つまり、修復デザインパターン）の候補がユーザに提示され得る。ユーザは、修復の合成及びターゲットのパッチについての戦略を選択することが可能とされる。また、例示的な一部の実施形態は、今後の修復ソリューションを最良にランク付けするようにユーザの選択から学習し得て、また、修復戦略が、ランク付けの順番でユーザに提示され得る。また、一部の実施形態は、ソフトウェアコーパス全体にわたって欠陥又は脆弱性を修復することを自律的に実行し得る。これは、継続的におよび／または周期的におよび／または設計の環境で実行することを含む。

【０１０５】

これまでに述べた実施形態のほかにも、本発明は、多種多様な用途に利用することが可能である。例えば、例示的な実施形態は、ソフトウェアコードのプログラミング時にプログラマを支援するように用いられ得る。これは、欠陥を特定することまたはコード再利用を提案することを含む。例示的な他の実施形態は、欠陥及び脆弱性を見つけ出すこと、ならびに場合によってはそれらを自動的に修理することに用いられ得る。例示的なさらなる他の実施形態は、コードを最適化するのに用いられ得る。これは、使用されてないコードを特定すること、非効率なコードを特定すること、および効率の低いコードを置き換えるためのコードを提案することを含む。

【０１０６】
また、例示的な実施形態は、どの脆弱性が所与のコードに存在する可能性があるのかを含む、リスク管理及び評価に用いられ得る。また、他の実施形態は、デザイン認定プロセスに用いられ得る。これは、ソフトウェアファイルにバグ、セキュリティ脆弱性、プロトコル不備などの既知の欠陥がないことの認定を提供することを含む。
【０１０７】
本発明の例示的なさらなる他の実施形態は、コード再利用発見手段（既に同じことをするものであるコードをコードベースにおいて見つけ出す）、コード品質測定、テキスト記述からコードへの翻訳手段、ライブラリ生成手段、テストケース生成手段、コードーデータ分離手段、コードマッピング・探索ツール、既存のコードの自動アーキテクチャ生成、アーキテクチャ改善提案手段、バグ／エラー推定手段、無駄なコードの発見、コードー機能マッピング、自動パッチ検証、コード改善決定ツール（機能リストを最小変更に対してマッピングする）、既存のデザインツールの拡張（例えば、ｅｎｔｅｒｐｒｉｓｅ ａｒｃｈｉｔｅｃｔ等）、代替実装提案手段、コード探索・学習ツール（例えば、教示のためのもの）、システムレベルコードライセンスフットプリント、および企業ソフトウェア使用マッピングを含む。
【０１０８】
これまでに述べた例示的な実施形態は、数多くの異なる方法で実現可能である。場合によっては、本明細書で説明する様々な方法や機械は、それぞれ、中央演算処理装置、メモリ、ディスク又は他の大容量記憶装置、少なくとも１つの通信インターフェース、少なくとも１つの入出力（Ｉ／Ｏ）装置、および他のペリフェラルを含む、物理的、仮想的又はハイブリッドの汎用コンピュータにより実現可能である。このような汎用コンピュータは、例えば、ソフトウェア命令をデータプロセッサにロードし当該命令の実行を引き起こして本明細書で説明する機能を行わせること等により、これまでに説明した方法を実行する機械へと変換される。また、それらのソフトウェア命令は、コーパスを形成するようにファイルを取り入れるインジェストモジュール、デザインパターンについての特定対象又は分析対象となる、コーパスのためのファイルについてのアーチファクトおよび／またはファイルを決定するアナリティクス（解析）モジュール、機械学習を実行するグラファナリティクス（グラフ解析）モジュール及びテキストアナリティクスモジュール、ファイル又はデザインパターンを特定する特定モジュール、コードを修復するか又は更新済みもしくは修復済みファイルを提供する修復モジュールなどにモジュール化され得る。例示的な一部の実施形態において、これらのモジュールは、さらなるモジュールへと結合又は分割され得る。
【０１０９】
当該技術分野において知られているように、そのようなコンピュータは、システムバスを備え得る。このバスは、コンピュータ（又は処理システム）の構成要素間のデータ転送に用いられるハードウェアラインのセットである。このような１つ以上のバスは、コンピュータシステムにおける相異なる構成要素（例えば、プロセッサ、ディスクストレージ、メモリ、入力／出力ポート、ネットワークポート等）同士を接続する共有の配管のようなものであり、それら構成要素間の情報のやり取りを可能にする。少なくとも１つの中央演算処理装置が、前記システムバスに取り付けられており、コンピュータ命令の実行を行う。典型的に、前記システムバスには、さらに、様々な入出力装置（例えば、キーボード、マウス、ディスプレイ、プリンタ、スピーカ等）を前記コンピュータに接続するためのＩ／Ｏ装置インターフェースが取り付けられる。少なくとも１つのネットワークインターフェースは、コンピュータがネットワークに取り付けられた他の様々なデバイスに接続することを可能にする。メモリは、一実施形態を実現するのに用いられるコンピュータソフトウェア命令及びデータを記憶する揮発性のメモリである。ディスクストレージ又は他の大容量記憶装置は、本発明で説明する様々な手順などを実施するのに用いられるコンピュータソフトウェア命令及びデータを記憶する、不揮発性のストレージ又は大容量記憶装置である。

【０１１０】
よって、典型的に、実施形態は、ハードウェア、ファームウェア、ソフトウェアまたはこれらの任意の組合せで実現可能である。また、例示的な実施形態は、全体的に又は部分的にクラウド上に存在し得て、かつ、インターネット又は他のネットワークアーキテクチャを介してアクセス可能であり得る。
【０１１１】
一部の実施形態において、本明細書で説明する手順、装置およびプロセスは、本発明にかかるシステムに対するソフトウェア命令の少なくとも一部を提供するコンピュータプログラムプロダクトを構成する。このようなコンピュータプログラムプロダクトは、非過渡的なコンピュータ読取り可能な媒体（例えば、少なくとも１つのＤＶＤ－ＲＯＭ、少なくとも１つのＣＤ－ＲＯＭ、少なくとも１つのディスク、少なくとも１つのテープなどといった取外し可能な記憶媒体等）を含む。このようなコンピュータプログラムプロダクトは、当該技術分野において周知である任意の適切なソフトウェアインストール方法によってインストール可能なものであり得る。他の実施形態において、前記ソフトウェア命令の少なくとも一部は、ケーブルおよび／または通信および／または無線接続を介してダウンロード可能なものであり得る。
【０１１２】
また、本明細書では、ファームウェア、ソフトウェア、ルーチンまたは命令が、データプロセッサの所与の動作および／または機能を実行しているかの如く説明されているかもしれない。しかし、本明細書に含まれるこのような説明はあくまでも便宜上のものに過ぎず、実際には、そのような動作は、それらファームウェア、ソフトウェア、ルーチン、命令などを実行するコンピューティング装置、プロセッサ、コントローラまたは他の装置から生じるものである。
【０１１３】
なお、フロー図、ブロック図およびネットワーク図は、構成要素の数が多くなっても又は少なくなってもよいし、配置構成が異なるものになってもよいし、表現が異なるものになってもよい。しかし、応用形態によってはブロック図やネットワーク図が変化し得て、かつ、実施形態の実行を示すブロック図やネットワーク図の数はその時々によって変化し得る。
【０１１４】
つまり、さらなる実施形態が、多種多様なコンピュータアーキテクチャおよび／または物理的なコンピュータおよび／または仮想的なコンピュータおよび／またはクラウドコンピュータおよび／またはこれらの所与の組合せによって実現可能である。よって、本発明で説明するデータプロセッサはあくまでも例示に過ぎず、実施形態を限定するものではない。
【０１１５】
本発明を、例示的な実施形態を参照しながら具体的に図示・説明したが、当業者であれば、添付の特許請求の範囲により包含される本発明の範囲を逸脱することなく形態や細部に様々な変更を施せることを理解するであろう。

【図１】

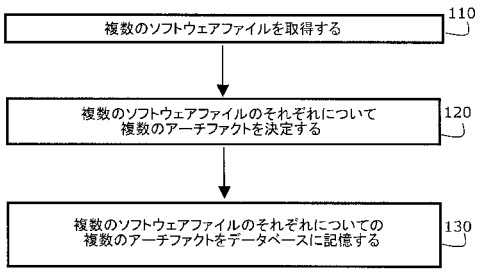

FIG. 1

【図２】

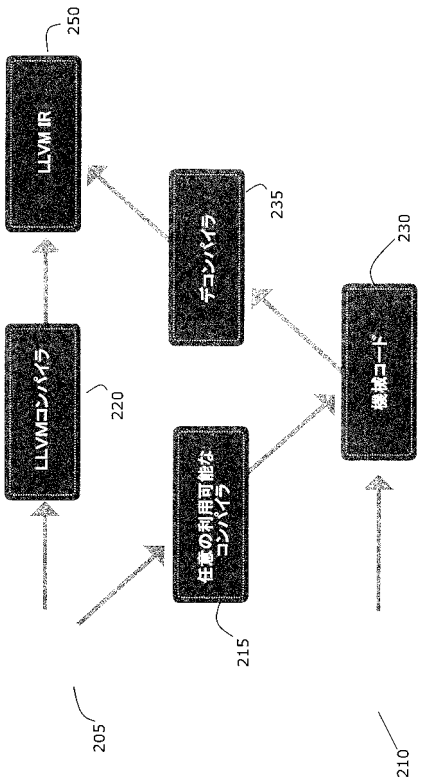

FIG. 2

【図３】

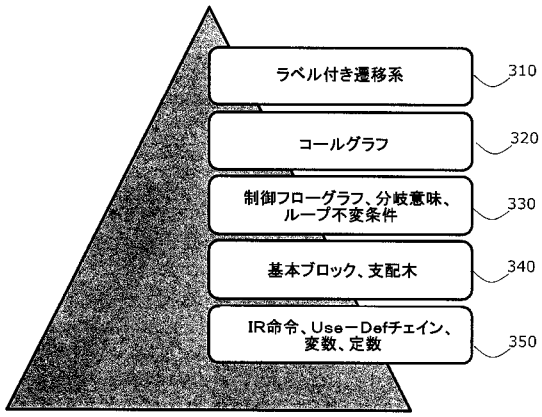

FIG. 3

【図４】

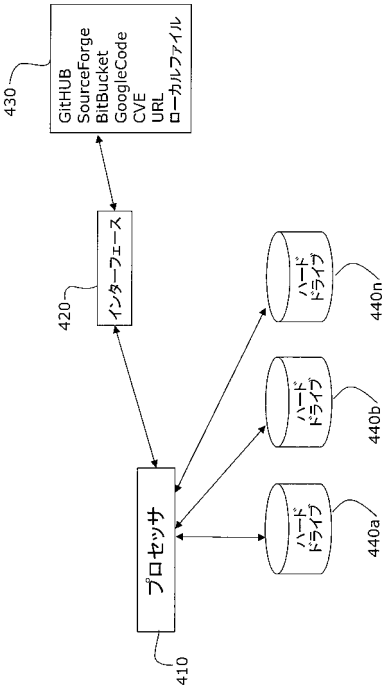

FIG. 4

【図５】

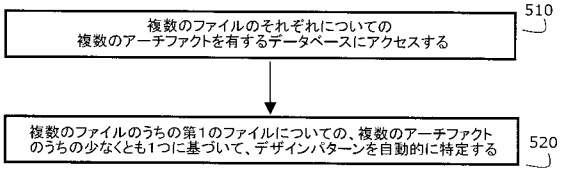

FIG. 5

【図６】

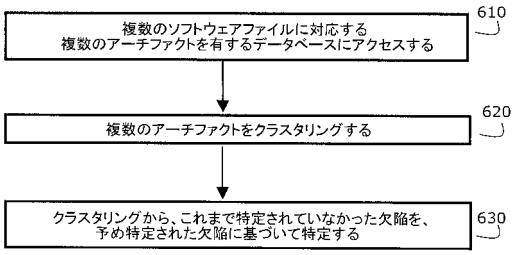

FIG. 6

【図７】

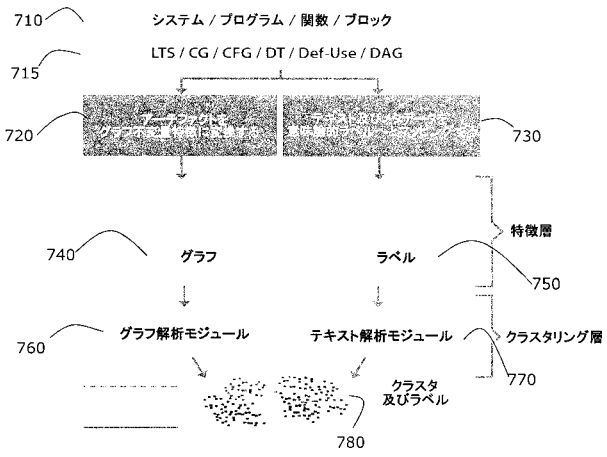

FIG. 7

【図８】

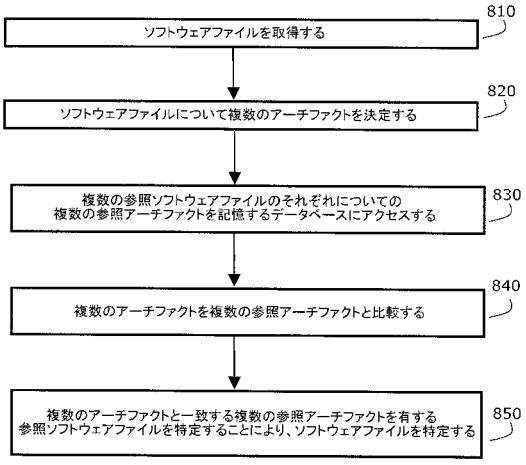

FIG. 8

【図９】

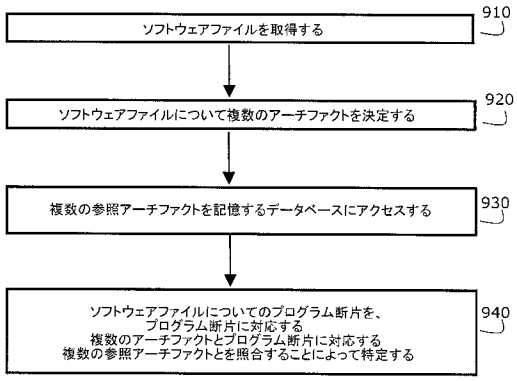

FIG. 9

【図１０】

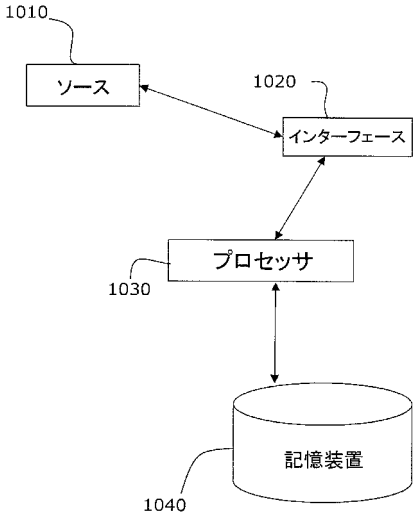

FIG. 10

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2015/035131

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/44
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2014/013304 A1 (VANGALA VIPINDEEP [IN] ET AL) 9 January 2014 (2014-01-09) paragraph [0001] paragraph [0024] - paragraph [0037] paragraph [0042] - paragraph [0044] -----	1-48
X	US 2012/311534 A1 (FOX BRIAN EDWARD [US] ET AL) 6 December 2012 (2012-12-06) paragraph [0004] - paragraph [0007] paragraph [0022] - paragraph [0032] paragraph [0039] paragraph [0054] paragraph [0088] - paragraph [0092] -----	1-48

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 August 2015	21/08/2015

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Milasinovic, Goran

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2015/035131

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014013304 A1	09-01-2014	NONE	
US 2012311534 A1	06-12-2012	US 2012311534 A1	06-12-2012
		US 2014075414 A1	13-03-2014

フロントページの続き

(81)指定国 AP(BW,GH,GM,KE,LR,LS,MW,MZ,NA,RW,SD,SL,ST,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,RU,TJ,TM),EP(AL,AT,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR,GB,GR,HR,HU,IE,IS,IT,LT,LU,LV,MC,MK,MT,NL,NO,PL,PT,RO,RS,SE,SI,SK,SM,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,KM,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AO,AT,AU,AZ,BA,BB,BG,BH,BN,BR,BW,BY,BZ,CA,CH,CL,CN,CO,CR,CU,CZ,DE,DK,DM,DO,DZ,EC,EE,EG,ES,FI,GB,GD,GE,GH,GM,GT,HN,HR,HU,ID,IL,IN,IR,IS,JP,KE,KG,KN,KP,KR,KZ,LA,LC,LK,LR,LS,LU,LY,MA,MD,ME,MG,MK,MN,MW,MX,MY,MZ,NA,NG,NI,NO,NZ,OM,PA,PE,PG,PH,PL,PT,QA,RO,RS,RU,RW,SA,SC,SD,SE,SG,SK,SL,SM,ST,SV,SY,TH,TJ,TM,TN,TR,TT,TZ,UA,UG,US

(74)代理人 100142608
弁理士 小林 由佳
(74)代理人 100154771
弁理士 中田 健一
(74)代理人 100155963
弁理士 金子 大輔
(74)代理人 100150566
弁理士 谷口 洋樹
(72)発明者 カーバック・ザ・サード・リチャード・ティー
アメリカ合衆国, マサチューセッツ州 ０２１４９, エベレット, ウッドローン ストリート ４３
(72)発明者 ゲイナー・ブラッド・ディー
アメリカ合衆国, マサチューセッツ州 ０２４５９, ニュートン, オークモント ロード １５
(72)発明者 シュニドマン・ネイサン・アール
アメリカ合衆国, マサチューセッツ州 ０２４２１, レキシントン, ピットケルン プレイス ６
(72)発明者 チン・サング・エイチ
アメリカ合衆国, マサチューセッツ州 ０２１３８, ケンブリッジ, コンコード アベニュー ２４８

Ｆターム(参考) 5B042 HH08 HH39 HH49
5B376 BC38 BC57 BC69 BC79