(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2012-98902
(P2012-98902A)
(43)公開日　平成24年5月24日(2012.5.24)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 9/44	(2006.01)	G06F	9/06	620K	5B376

審査請求 未請求　請求項の数 14　ＯＬ　（全 25 頁）

(21)出願番号　特願2010-245778 (P2010-245778)
(22)出願日　平成22年11月2日 (2010.11.2)

(71)出願人　000005108
株式会社日立製作所
東京都千代田区丸の内一丁目６番６号
(74)代理人　100100310
弁理士　井上　学
(74)代理人　100098660
弁理士　戸田　裕二
(74)代理人　100094271
弁理士　渡邊　孝弘
(74)代理人　100091720
弁理士　岩崎　重美
(72)発明者　手島　久典
神奈川県横浜市戸塚区吉田町２９２番地
株式会社日立製作所システム開発研究所内

最終頁に続く

(54)【発明の名称】ソフトウェア資産整理方法及び装置

(57)【要約】
【課題】
大規模システムにおいて、少ない計算資源で実行可能なソフトウェア資源の整理を提供する。
【解決手段】
ソースコードを所定のソースコード分割条件に基いて複数のソースブロックに分割し、複数のソースブロック各々について当該ソースブロック内で使用される変数を抽出し、複数のソースブロック間の関係を判別し、指定された一以上の変数各々について、当該変数と所定の対応関係を有する一以上のソースブロックを判定して、当該判定の結果に基いて、複数のソースブロックを一以上のグループにまとめるためのマーカを抽出し、マーカを用いた所定のグループ化条件に基いて、複数のソースブロックの内二以上のソースブロックを一つのソフトウェア部品として整理可能なソースブロック群としてグループ化する。
【選択図】　図４

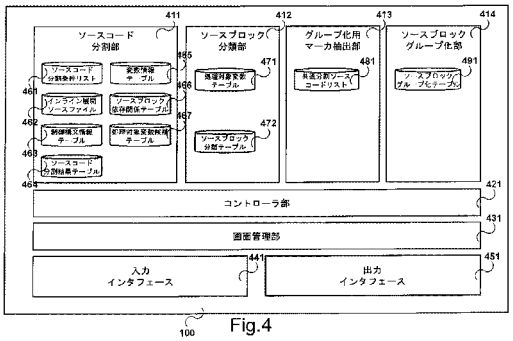


Fig.4

【特許請求の範囲】
【請求項１】
ソフトウェアのソースコードから、当該ソフトウェアの部品として管理可能なソースブロックを抽出する装置であって、
前記ソースコードを所定のソースコード分割条件に基いて複数のソースブロックに分割し、当該複数のソースブロック各々について当該ソースブロック内で使用される変数を抽出し、当該複数のソースブロック間の関係を判別するソースコード分割部と、
指定された一以上の変数各々について、当該変数と所定の対応関係を有する一以上のソースブロックを判定するソースブロック分類部と、
前記ソースブロック分類部の判定結果に基いて、前記複数のソースブロックを一以上のグループにまとめるためのマーカを抽出するグループ化用マーカ抽出部と、
前記グループ化用マーカ抽出部が抽出したマーカを用いた所定のグループ化条件に基いて、前記複数のソースブロックの内二以上のソースブロックを一つのソフトウェア部品として整理可能なソースブロック群としてグループ化するソースブロックグループ化部とを備えたソフトウェア資源整理装置。
【請求項２】
請求項１記載のソフトウェア資源整理装置であって、前記ソースコード分割条件は、ソースコードの言語に応じて設定されることを特徴とする、ソフトウェア資産整理装置。
【請求項３】
請求項１記載のソフトウェア資産整理装置であって、前記ソースコード分割部は、前記複数のソースブロック各々について、変数の読み出しおよび書き込みが生じるか否かを判別し、判別結果に基いて前記複数のソースブロック間の関係を判別することを特徴とするソフトウェア資産整理装置。
【請求項４】
請求項１記載のソフトウェア資産整理装置であって、前記ソースブロック分類部は、ソースブロックの除去により前記変数の値が変化し得うる場合に、当該ソースブロックと当該変数とが前記所定の対応関係を有すると判断することを特徴とするソフトウェア資産整理装置。
【請求項５】
請求項１記載のソフトウェア資産整理装置であって、前記グループ化用マーカ抽出部は、前記一以上の変数全てと前記所定の対応関係を持つソースブロックを前記マーカとして抽出することを特徴とする、ソフトウェア資産整理装置。
【請求項６】
請求項１記載のソフトウェア資産整理装置であって、前記ソースブロックグループ化部はさらに、作成されたソースブロック群の総数と、ソースブロック群に含まれる二以上のソースブロックのソースコードの一致行数とを算出し、当該ソースブロック群の総数および当該ソースコードの一致行数とに基いて、グループ化による効果を示す指標をユーザに提示することを特徴とする、ソフトウェア資産整理装置。
【請求項７】
請求項５記載のソフトウェア資産整理装置であって、前記ソースブロックグループ化部は、二つのマーカを結ぶプログラム実行経路において、他のマーカを含まない前記プログラム実行経路上に存在する複数のソースブロックを、ソースブロック群としてグループ化することを特徴とする、ソフトウェア資産整理装置。
【請求項８】
ソフトウェアのソースコードから、当該ソフトウェアの部品として管理可能なソースブロックを抽出するソフトウェア資産整理方法であって、
前記ソースコードを所定のソースコード分割条件に基いて複数のソースブロックに分割し、
当該複数のソースブロック各々について当該ソースブロック内で使用される変数を抽出し、

当該複数のソースブロック間の関係を判別し、
指定された一以上の変数各々について、当該変数と所定の対応関係を有する一以上のソースブロックを判定して、当該判定の結果に基いて、前記複数のソースブロックを一以上のグループにまとめるためのマーカを抽出し、
前記マーカを用いた所定のグループ化条件に基いて、前記複数のソースブロックの内二以上のソースブロックを一つのソフトウェア部品として整理可能なソースブロック群としてグループ化することを特徴とするソフトウェア資産整理方法。
【請求項９】
請求項８記載のソフトウェア資源整理方法であって、前記ソースコード分割条件は、ソースコードの言語に応じて設定されることを特徴とする、ソフトウェア資産整理方法。
【請求項１０】
請求項８記載のソフトウェア資産整理方法であって、前記ソースブロック間の関係を判別する場合に、前記複数のソースブロック各々について、変数の読み出しおよび書き込みが生じるか否かを判別し、判別結果に基いて前記複数のソースブロック間の関係を判別することを特徴とするソフトウェア資産整理方法。
【請求項１１】
請求項８記載のソフトウェア資産整理方法であって、ソースブロックの除去により前記変数の値が変化し得うる場合に、当該ソースブロックと当該変数とが前記所定の対応関係を有すると判断することを特徴とするソフトウェア資産整理方法。
【請求項１２】
請求項８記載のソフトウェア資産整理方法であって、前記一以上の変数全てと前記所定の対応関係を持つソースブロックを前記マーカとして抽出することを特徴とする、ソフトウェア資産整理方法。
【請求項１３】
請求項８記載のソフトウェア資産整理方法であって、
さらに、作成されたソースブロック群の総数と、ソースブロック群に含まれる二以上のソースブロックのソースコードの一致行数とを算出し、当該ソースブロック群の総数および当該ソースコードの一致行数とに基いて、グループ化による効果を示す指標をユーザに提示することを特徴とする、ソフトウェア資産整理方法。
【請求項１４】
請求項１２記載のソフトウェア資産整理方法であって、二つのマーカを結ぶプログラム実行経路において、他のマーカを含まない前記プログラム実行経路上に存在する複数のソースブロックを、ソースブロック群としてグループ化することを特徴とする、ソフトウェア資産整理方法。
【発明の詳細な説明】
【技術分野】
【０００１】
大規模システムにおける、既存ソフトウェアの部品化に関する。
【背景技術】
【０００２】
鉄道運行管理システムのような大規模システムは、一度構築されると十年以上の長期に渡り使用される。このようなシステムは、システムが構築された後も状況に応じて改造・改修を行い、順次機能を拡張していく必要がある。
【０００３】
例えばコンピュータ制御の発展期に構築された鉄道運行管理システムは、高密度輸送への対応や旅客案内の電子化など新たなニーズへの対応を常に迫られてきた。また、他線区からの直通運転など従来は想定されていなかった運行形態も誕生し、運行管理システムは新しい運行形態への対応にも迫られてきた。
【０００４】
システムの改修において、一度作成したソースコードに手を加える場合は、改修の影響

が及ぶ範囲について再度テストを実施しなければならない。しかし、不用意に改修を行うと影響範囲が拡大しテスト工数の増大を招く。このような工数の増大は大規模システムでは特に顕著である。このため、既存のソースコードには極力手をつけず、改修場所の直近か末尾に新たなソースコードを付加することにより影響範囲を限定するという方法が一般に用いられている。

【０００５】

例えばシステムの改修は特定のデータに関連する処理のみを対象として行う場合がほとんどであり、ソースコードの構造もこうした改修を見据えて設計されている。すなわち、例え類似の処理であっても処理対象のデータごとに別個の関数やソースコードが用意されている。このような設計であれば、改修による影響が他のデータに波及し、予期せぬ不具合が発生する可能性を減じることができる。もし改修部分が当初は変更が予期されていなかった場所であり、全データに対する共通処理とされていた場合でも、関連するソースコードを直近か末尾にコピー＆ペーストして書き換えることにより、他のデータへの影響を最小限に抑えることができる。

【０００６】

しかし、このような改修が繰り返されると、ソースコード規模の増大や不要ソースコードの蓄積、ソースコード構造の変容などが引き起こされ、可読性が大きく低下する。また、一見すると同一の処理に見えるが部分的に異なっているソースコードが多数発生してしまうため、それらの差異に気付かず誤った改修を行ってしまう危険性もある。結果として保守性が悪化し、長期に渡り改修に携わってきた作業者以外には手が出せないといった状況になってしまった。さらに、団塊世代の退職により有識者が大幅に減少しており、保守工数の加速的な増加が進行している。

【０００７】

このような問題の顕在化に加えて、例えば鉄道運行管理システムのように成熟したシステムでは新規機能の追加は回数が減少してきており、改修に適したプログラム構造より保守性の高いプログラム構造が強く求められている。

【０００８】

保守性の高いプログラムを製造するために有効な技術として、ソフトウェア製品系列開発が提案されている。ソフトウェア製品系列開発では、複数のソフトウェア製品群に対して製品間の共通性と相違性を定義し、ソフトウェアを共通部品と非共通部品に分けて製造する。製品間の共通性に従って部品を作ることにより、改修時の影響範囲を容易に知ることができ、保守性や再利用性を高めることができる。

【０００９】

鉄道運行管理システムのような大規模システムでも、処理対象のデータごとに作成されている従来のソースコードをソフトウェア製品群と見做すことができる。従って、ソースコードを処理対象のデータによらない共通処理と処理対象のデータに固有な処理に分類し、それぞれのソースコードを部品化すれば、ソフトウェア製品系列開発の考え方を鉄道運行管理システムのような大規模システムに適用できる。この際、既存のソースコードから共通処理と固有処理を抽出するための技術が必要である。

【００１０】

既存のソースコードを対象に共通処理を分析する技術として、ソースコードの一致部分を重複コードとして検出するものがある。重複コードは構文解析により単語や行といった単位で判別される。さらに、完全一致ではないが類似しているソースコードを検出するための技術として、ソースコード上の位置に基づき周辺の重複コードをグループ化する技術が開示されている（特許文献１）。

【先行技術文献】

【特許文献】

【００１１】

【特許文献１】特許第３８６８８５９号

【発明の概要】

【発明が解決しようとする課題】
【００１２】
特許文献１は対象文字列中の類似部分である類似部分文字列を検出する装置に関する。ソースコード中の類似部分文字列を検出することにより、既存のソースコードに含まれる共通処理を抽出することが出来る。しかし、抽出された共通処理は断片化されており処理の区切りとは必ずしも一致しないため、部品化のためには近傍のソースコードに含まれる処理の区切りを探索する必要がある。しかし、類似部分の中には変数宣言部や偶発的な一致部分などのノイズが多量に含まれるため、これら全てについて判定を行うためには多大な計算資源が必要である。この問題は大規模システムにおいて特に顕著である。また、類似部分文字列の検出はソースコード全体に渡る検索が必要なため、検出条件の変更といったチューニングを行う度に再計算を行う必要がある。さらに、偶発的な一致部分をフィルタリングする方法も実装しなければならない。
【００１３】
そこで、より少ない計算資源で実行可能なソフトウェア資産の整理方法および装置を提供する。
【課題を解決するための手段】
【００１４】
ソフトウェアのソースコードから、当該ソフトウェアの部品として管理可能なソースブロックを抽出する装置は、ソースコード分割部と、ソースブロック分類部と、グループ化用マーカ抽出部と、ソースブロックグループ化部とを有する。
【００１５】
ソースコード分割部は、ソースコードを所定のソースコード分割条件に基いて複数のソースブロックに分割し、複数のソースブロック各々についてソースブロック内で使用される変数を抽出し、複数のソースブロック間の関係を判別する。
【００１６】
ソースブロック分類部は、指定された一以上の変数各々について、変数と所定の対応関係を有する一以上のソースブロックを判定する。
【００１７】
グループ化用マーカ抽出部は、判定結果に基いて、複数のソースブロックを一以上のグループにまとめるためのマーカを抽出する。
【００１８】
ソースブロックグループ化部は、グループ化用マーカ抽出部が抽出したマーカを用いた所定のグループ化条件に基いて、複数のソースブロックの内二以上のソースブロックを一つのソフトウェア部品として整理可能なソースブロック群としてグループ化する。
【発明の効果】
【００１９】
より少ない計算資源でソフトウェア資産の整理が可能であり、チューニングも容易となる。
【図面の簡単な説明】
【００２０】
【図１】ソフトウェア資産整理装置が提供する入力画面例である。
【図２】ソフトウェア資産整理装置が提供する出力画面例である。
【図３】ソフトウェア資産整理装置のハードウェア構成例である。
【図４】ソフトウェア資産整理装置のソフトウェア構成例である。
【図５】ソフトウェア資産整理装置の全体処理の一例の概念を示す図である。
【図６】ソフトウェア資産整理装置の全体処理の一例を示すフローである。
【図７】ソースコードの分割処理の一例を示すフローである。
【図８】変数とソースブロックを対応付ける処理の一例を示すフローである。
【図９】ソースブロック間の依存関係を求める処理の一例を示すフローである。
【図１０】ソースブロック分類部の処理の一例を示すフローである。

【図１１】ソースブロック分類部の処理の一例を示すフローである。
【図１２】ソフトウェア資産整理装置の詳細処理の一例を示すフローである。
【図１３】ソフトウェア資産整理装置の詳細処理の一例を示すフローである。
【図１４】ソースコード分割条件リストの一例を示す図である。
【図１５】制御構文情報テーブルの一例を示す図である。
【図１６】ソースコード分割結果テーブルの一例を示す図である。
【図１７】変数情報テーブルの一例を示す図である。
【図１８】ソースブロック依存関数テーブルの一例を示す図である。
【図１９】処理対象データ候補テーブルの一例を示す図である。
【図２０】処理対象データテーブルの一例を示す図である。
【図２１】ソースブロック分類テーブルの一例を示す図である。
【図２２】共通ソースブロックリストの一例を示す図である。
【図２３】ソースブロックグループ化テーブルの一例を示す図である。
【発明を実施するための形態】
【００２１】
以下、本発明の一実施形態を図１～図２３を参照して詳細に説明する。
【実施例】
【００２２】
図１は開発者が既存のソフトウェア資産を部品化する際に用いる、ソフトウェア資産整理装置１００が提供する入力画面の構成例である。ソフトウェア資産整理装置１００の入力画面はソースファイル選択窓１１０と条件設定窓１２０とソースコードプレビュー窓１３０と分割実行ボタン１４０と処理対象変数選択窓１５０と部品化判定条件設定窓１６０とグループ化実行ボタン１７０から構成される。
【００２３】
ソースファイル選択窓１１０はファイル表示窓１１１とソースファイル追加ボタン１１２とソースファイル除去ボタン１１３とソースファイル決定窓１１４から構成される。ファイル表示窓１１１にはファイルシステム内におけるディレクトリ構造が表示されている。図１ではディレクトリｒｏｏｔの下にディレクトリＤｉｒｅｃｔｏｒｙＡ、ＤｉｒｅｃｔｏｒｙＢが表示されている。またＤｉｒｅｃｔｏｒｙＡの下にはＤｉｒｅｃｔｏｒｙＡ１、ＤｉｒｅｃｔｏｒｙＡ２が、ＤｉｒｅｃｔｏｒｙＡ１の下にはソースファイルｈｏｇｅ．ｃ、ｆｕｇａ．ｃが表示されている。開発者はソフトウェア資産整理装置１００においてソフトウェア資産整理を行いたいソースファイルをファイル表示窓１１１から選択した後、ソースファイル追加ボタン１１２を押下することによりソースファイル決定窓１１４へと当該ソースファイルを登録することができる。また、ソースファイル決定窓１１４へ登録されたソースファイルを除去したい場合は、ソースファイル決定窓１１４において除去したいソースファイルを選択した後、ソースファイル除去ボタン１１３を押下することにより除去することができる。ソースファイルの追加、除去の際には、ソースコードプレビュー窓１３０により選択したソースコードの内容を確認することができる。ソースファイル決定窓１１４では、登録されたソースファイルの中から、ソフトウェア資産整理装置１００においてソフトウェア資産部品化の解析対象とするソースファイルをチェックボックスにより選択する。図１ではソフトウェア資産部品化の解析対象とするソースファイルとしてｈｏｇｅ．ｃ、ｆｕｇａ．ｃ、ｈｅａｄ．ｈが選択された状態となっている。
【００２４】
条件設定窓１２０ではソフトウェア資産整理装置１００の解析に用いる条件を選択することができる。解析に用いる条件とは、例えば分析対象のソースコードの記述に用いられているプログラミング言語である。解析に用いる条件は変更ボタン１２１を押下することにより変更することができる。
【００２５】
ソースファイル選択窓１１０と条件設定窓１２０の入力が完了した後、分割実行ボタン１４０を押下されると、ソフトウェア資産整理装置１００によりソースコード分割処理が

開始される。

【００２６】

ソースコード分割処理が完了した後、ソフトウェア資産整理装置１００により処理対象変数選択窓１５０に処理対象データ（以下、処理対象変数ともいう。）の候補が表示される。処理対象変数選択窓１５０は処理対象データ候補表示窓１５１と処理対象データ追加ボタン１５２と処理対象データ除去ボタン１５３と処理対象データ決定窓１５４から構成される。図１ではｒｏｏｔの下に構造体ＳｔｒｕｃｔＡ、ＳｔｒｕｃｔＢが表示されている。またＳｔｒｕｃｔＡの下にはＳｔｒｕｃｔＡ１、ＳｔｒｕｃｔＡ２が、ＳｔｒｕｃｔＡ１の下にはデータｄａｔａＡ、ｄａｔａＢが表示されている。開発者はソフトウェア資産整理装置１００においてソースコードの分類を行うために用いる処理対象データを処理対象データ候補表示窓１５１から選択した後、処理対象データ追加ボタン１５２を押下することにより処理対象データ決定窓１５４へと当該処理対象データを登録することができる。また、処理対象データ決定窓１５４へ登録された処理対象データを除去したい場合は、処理対象データ決定窓１５４において除去したい処理対象データを選択した後、処理対象データ除去ボタン１５３を押下することにより除去することができる。処理対象データ決定窓１５４では、登録された処理対象データの中から、ソフトウェア資産整理装置１００においてソースコードの分類を行うために用いる処理対象データをチェックボックスにより選択する。図１ではｄａｔａＡ、ｄａｔａＢが選択された状態となっている。

【００２７】

ここで、処理対象データとは、当該データの生成が対象プログラムの実行目的となっているものであり、ソースコード内においては、プログラムの返り値やテーブルへと格納された変数、当該プログラム内で書き換えが行われた後に他プログラムで入力として利用されている変数などの形で判別することができる。また、仕様書からは、プログラムの出力として明示されているデータとして判別することができる。一例として、鉄道運行管理システムにおける、運行制御用の情報テーブルから旅客案内用の情報を生成するプログラムの場合について説明する。前記プログラムにおいては、目的となる出力は旅客案内用の情報であり、処理対象データは例えば列車の行先や停車駅となる。

【００２８】

処理対象変数選択窓１５０の入力が完了した後は、部品化判定条件設定窓１６０から二つの部品を同一部品と判定するための条件として、例えばソースコード一致率の下限を入力する。その後、グループ化実行ボタン１７０を押下されると、ソフトウェア資産整理装置１００によりソフトウェア資産として部品化可能なソースコード群のグループ化処理が開始する。

【００２９】

図２はソフトウェア資産整理装置１００が提供する出力画面の構成例である。ソフトウェア資産整理装置１００の出力画面はソースブロックグループ化結果表示窓２１０とソースブロック分類結果表示窓２２０とソースコード比較結果表示窓２３０とメトリクス表示窓２４０と再表示ボタン２５０から構成される。

【００３０】

ソースブロックグループ化結果表示窓２１０はグループ表示窓２１１とグループ内のソースブロック表示窓２１２から構成される。グループ表示窓２１１には、一つのソフトウェア部品として資産化されるソースブロック群をグループ化したものの一覧が表示される。図２では、グループとしてＧｒｏｕｐ１、Ｇｒｏｕｐ２、Ｇｒｏｕｐ３、Ｇｒｏｕｐ４が表示されている。また、Ｇｒｏｕｐ１が選択状態となっている。グループ内のソースブロック表示窓２１２にはグループ表示窓２１１において選択中のグループに属するソースブロック群が表示される。図２では、Ｇｒｏｕｐ１に属するＢｌｏｃｋ２、Ｂｌｏｃｋ５、Ｂｌｏｃｋ９、Ｂｌｏｃｋ１５が表示されている。また、グループ内のソースブロック表示窓２１２では選択グループ内のソースコード比較結果表示窓２３０に表示されるソースブロックをチェックボックスにより選択することができる。図２では、Ｂｌｏｃｋ２、Ｂｌｏｃｋ５、Ｂｌｏｃｋ９が選択状態となっている。開発者は、ソースブロックグルー

プ化結果表示窓２１０、ソースコード比較結果表示窓２３０の表示結果より、一つのソフトウェア部品として資産化されたソースコード群が何であるかを知ることができる。また、開発者は部品化結果について手動で修正を加えることもできる。

【００３１】

ソースブロック分類結果表示窓２２０はソースブロック名表示欄２２１と処理対象データ表示欄２２２とソースブロックと処理対象データの対応関係表示欄２２３から構成される。図２では、ソースブロックと処理対象データの対応関係表示欄２２３中の黒塗りとなっているセルは、ソースブロック名表示欄２２１のうち当該セルの左に表示されているソースブロック名が、処理対象データ表示欄２２２のうち当該セルの上に表示されている処理対象データに対応付けられていることを示している。開発者は、ソースブロック分類結果表示窓２２０の結果より、ソースコードの部品化作業時に取り扱っているソースブロックがどの処理対象データに影響を与えうるかを確認することができる。

【００３２】

ソースコード比較結果表示窓２３０はグループ内のソースブロック表示窓２１２において選択中のソースブロックのソースコード一致部分と相違部分を表示する。図２では黒塗り部が一致部分、非黒塗り部が相違部分を表す。開発者は、ソースコード比較結果表示窓２３０の結果より、ソースコードの部品化作業において、重複しているソースコードの除去や、ソースブロック内のソースコード群を統合する際に各々の行を挿入する場所を判別することができる。

【００３３】

メトリクス表示窓２４０はソフトウェア資産の整理による効果を表示する。ソフトウェア資産の整理による効果とは、例えば部品化により削減可能なソースコード行数やソースコード削減率である。また、ソースブロック総数や部品化後の部品点数、ソースコード総行数などの情報も表示する。図２では、ソースコード削減率として６４％が表示されており、これは部品化により削減可能なソースコード行数である１８、５５２行をソースコード総行数である２８、９８８行で除したものである。開発者は、メトリクス表示窓２４０の結果より、部品化作業の作業規模や部品化に伴うソースコードの削減効果を定量的に評価することができる。

【００３４】

再表示ボタン２５０は、ソースブロックグループ化結果表示窓２１０において選択中のグループ、ソースブロックを変更した際に押下することにより、ソースブロック分類結果表示窓２２０、ソースコード比較結果表示窓２３０、メトリクス表示窓２４０を更新するためのボタンである。

【００３５】

図３はソフトウェア資産整理装置１００のハードウェア構成例である。ソフトウェア資産整理装置１００のハードウェアは、中央処理装置３１０と主記憶装置３２０と内部バス３３０とバスインタフェース３４０と外部バス３５０と入出力装置３６０と入出力装置インタフェース３６１と大容量記憶装置３７０と大容量記憶装置インタフェース３７１と通信装置３８０と通信装置インタフェース３８１から構成される。

【００３６】

中央処理装置３１０はプログラム実行等の演算を行うためのプロセッサである。主記憶装置３２０はプログラム実行時にプログラムや演算結果などのデータを一時的に格納するための装置である。中央処理装置３１０と主記憶装置３２０は内部バス３３０により接続されており、内部バス３３０はバスインタフェース３４０を介して外部バス３５０に接続されている。

【００３７】

入出力装置インタフェース３６１は入出力装置３６０と接続される。入出力装置３６０はディスプレイやキーボード、マウスなど、開発者とのインターフェースデバイスである。図１および図２に示す入力画面、出力画面は入出力インタフェース３６１から提供される。大容量記憶装置インタフェース３７１は大容量記憶装置３７０に接続される。大容量

記憶装置３７０は基本プログラムや処理結果などを永続的に記憶するための装置である。通信装置インタフェース３８１は通信装置３８０と接続される。通信装置３８０は外部サーバ装置などと接続するための装置であり、処理対象のソースコードや処理結果をネットワークを通じて外部サーバ装置へと提供することができる。通信装置３８０を用いることにより、例えばソフトウェア資産整理装置１００の機能をネットワークを通じた先に接続されている計算機に対してサービスとして提供することが可能となる。

【００３８】

図４はソフトウェア資産整理装置１００のソフトウェア構成例である。ソフトウェア資産整理装置１００のソフトウェアは、ソースコード分割部４１１とソースブロック分類部４１２とグループ化用マーカ抽出部４１３とソースブロックグループ化部４１４とコントローラ部４２１と画面管理部４３１と入力インタフェース４４１と出力インタフェース４５１とソースコード分割条件リスト４６１とインライン展開ソースファイル４６２と制御構文情報テーブル４６３とソースコード分割結果テーブル４６４と変数情報テーブル４６５とソースブロック依存関係テーブル４６６と処理対象データ候補テーブル４６７と処理対象データテーブル４７１とソースブロック分類テーブル４７２と共通ソースブロックリスト４８１とソースブロックグループ化テーブル４９１から構成される。

【００３９】

ソースコード分割部４１１、ソースブロック分類部４１２、グループ化用マーカ抽出部４１３、ソースブロックグループ化部４１４、コントローラ部４２１、画面管理部４３１、入力インタフェース４４１、出力インタフェース４５１を実現するプログラムは実行時に大容量記憶装置３７０から主記憶装置３２０へと展開され、中央処理装置３１０により演算が行われる。また、プログラムの実行により生成されたソースコード分割条件リスト４６１、インライン展開ソースファイル４６２、制御構文情報テーブル４６３、ソースコード分割結果テーブル４６４、変数情報テーブル４６５、ソースブロック依存関係テーブル４６６、処理対象データ候補テーブル４６７、処理対象データテーブル４７１、ソースブロック分類テーブル４７２、共通ソースブロックリスト４８１、ソースブロックグループ化テーブル４９１は主記憶装置３２０へと展開される。開発者はキーボードやマウス、ディスプレイなどの入出力装置３６０を介して入力インタフェース４４１へと指示を与えることによりプログラムの実行を制御することができる。また、入出力装置３６０として着脱可能で可搬性のある記憶媒体を用いることによりデータの入出力を行うことができる。通信装置３８０を通じ、他の装置との間で入出力を行うこともできる。各機能部を実現するプログラムやソースファイル決定窓１１４で選択できるソースファイルは、予め大容量記憶装置３７０に格納されていてもよいし、利用可能な媒体を介して他の装置から大容量記憶装置３７０や主記憶装置３２０に導入されてもよい。媒体とは、例えば、入出力装置３６０としては着脱可能な記憶媒体、または、通信装置３８０としてはネットワーク、ネットワークを伝搬する搬送波やデジタル信号などの通信媒体を指す。

【００４０】

ソースコード分割部４１１は既存ソースコードをインライン展開し、制御構文によりソースコードを分割する部分である。その際、ソースブロック間の依存関係導出やソースコード内で利用されているデータの抽出も行う。ソースコード分割部４１１では、まず条件設定窓１２０に入力された解析条件を読み込み、ソースコード分割条件リスト４６１へと格納する。

【００４１】

ソースコード分割条件リスト４６１の詳細例を図１４に示す。ソースコード分割条件リスト４６１には、ソースコード言語設定、ソースコード分割条件が格納されている。ソースコード言語設定にはＣ言語やＪＡＶＡ（登録商標）など、ソースコードを記述している言語が格納されている。これは、条件分岐の探索や実行順序の判断などの構文解析を行う際に用いる。ソースコード分割条件には、入力ソースコードをソースブロックへと分割する際に用いる条件が格納されている。

【００４２】

次に、ソースコード分割部４１１はソースファイル決定窓１１４において選択されたソースファイル名で指定されるソースファイルからソースコードを読み込み、当該ソースコードをインライン展開したものをインライン展開ソースファイル４６２へと格納する。そしてソースコード分割部４１１はインライン展開ソースファイル４６２に対して構文解析処理を行い、抽出された制御構文の詳細を制御構文情報テーブル４６３へと格納する。ここで、制御構文とは、例えばＣ言語であれば条件分岐を表すｉｆ文や反復処理を表すｆｏｒ文のことである。制御構文情報テーブル４６３はソースコード分割の際に用いる。

【００４３】

制御構文情報テーブル４６３の詳細例を図１５に示す。制御構文情報テーブル４６３はソースファイル中に出現した制御構文のそれぞれについて、制御構文ＩＤ、開始位置、終了位置、制御構文種類、実行条件、親階層制御構文ＩＤが格納されている。制御構文ＩＤには対象の制御構文を一意に識別するためのキーが格納されている。制御構文ＩＤの割り当ては重複がなければ任意で構わない。開始位置には制御構文により制御されるソースコード範囲（以下、制御範囲と記述）のうち、一番初めの行の行番号が格納されている。終了位置には制御範囲の一番最後の行の行番号が格納されている。尚、行番号はインライン展開ソースファイル４６２におけるものを用いる。制御構文には制御構文の種類が格納されている。実行条件には制御構文が実行される際に満たすべき条件が格納されている。親階層制御構文ＩＤには、制御構文Ａが他の制御構文Ｂの制御範囲に包含されていた場合、ＢをＡの親階層制御構文と定義し、対象としている制御構文の親階層制御構文の制御構文ＩＤが格納されている。ここで、制御構文Ａが複数の制御構文Ｂ、Ｃに包含されている場合、すなわち、制御構文Ｂが制御構文Ｃにさらに包含されているような場合は、Ａの直上の制御構文、すなわち制御構文Ｂを親階層制御構文とする。

【００４４】

制御構文情報テーブル４６３の作成が終了した後は、ソースコード分割部４１１はソースコード分割条件リスト４６１のソースコード分割条件を参照し、条件に該当する制御構文を制御構文情報テーブル４６３から検索してソースコードを複数のソースブロックに分割する。作成結果はソースコード分割結果テーブル４６４に格納する。ソースコード分割結果テーブル４６４はソースコード分割部４１１における主要な成果物である。

【００４５】

ソースコード分割結果テーブル４６４の詳細を図１６に示す。ソースコード分割結果テーブル４６４にはソースブロックＩＤ、開始位置、終了位置、実行条件、親階層ソースブロックＩＤが格納されている。ソースブロックＩＤには対象のソースブロックを一意に識別するためのキーが格納されている。ソースブロックＩＤの割り当ては重複がなければ任意で構わない。開始位置にはソースブロックの開始行の行番号が格納されている。終了位置にはソースブロックの終了行の行番号が格納されている。尚、行番号はインライン展開ソースファイル４６２におけるものを用いる。実行条件にはソースブロックが実行される際に満たすべき条件が格納されている。親階層ソースブロックＩＤには、当該ソースブロックに対応する制御構文について、制御構文情報テーブル４６３の親階層制御構文ＩＤに値が格納されていた場合、親階層制御構文ＩＤが表す制御構文に対応するソースブロックのソースブロックＩＤが格納されている。

【００４６】

インライン展開ソースファイル４６２からソースコード分割結果テーブル４６４が生成される様子を概念的に図５に示す。図５では、ソースコード分割部４１１によって、インライン展開ソースファイル４６２の１行目がソースブロック１に、２～５行目がソースブロック２に、６～９行目がソースブロック３に、１０～１３行目がソースブロック４に、１４行目がソースブロック５に分割されている。

【００４７】

ソースコード分割結果テーブル４６４の作成が終了した後は、ソースコード分割部４１１は処理対象データの抽出とソースブロック間の関連を導出する。まずソースコード分割部４１１はインライン展開ソースファイル４６２から、当該インライン展開ソースファイ

ル４６２内に出現する変数を抽出し、変数情報テーブル４６５へと格納する。変数情報テーブル４６５はソースブロック間の依存関係を判別するために用いられる。

【００４８】

変数情報テーブル４６５の詳細例を図１７に示す。変数情報テーブル４６５には変数ＩＤ、変数名、スコープ、変数型、出現位置、対応分割ソースコードＩＤ（以下、分割ソースコードをソースブロックとも呼ぶ）、利用種別フラグが格納されている。変数ＩＤには対象の変数を一意に識別するためのキーが格納されている。変数ＩＤの割り当ては重複がなければ任意で構わない。変数名にはインライン展開ソースファイル４６２内で使用されている変数の名称が格納されている。スコープにはソースコード上において当該変数に読み書きが行える範囲について、インライン展開ソースファイル４６２における開始行と終了行の行番号が格納されている。ここで、変数情報テーブル４６５においては同一変数名でもスコープが異なれば別々の変数として扱われる。変数型には対象としている変数の型が格納されている。ここで、変数の型とは、例えばＣ言語ではｉｎｔやｆｌｏａｔである。本実施例では、記述の煩雑さを避けるため、出現位置、対応ソースブロックＩＤ、利用種別フラグは変数ごとに別表としている。出現位置には、インライン展開ソースファイル４６２上で変数が出現する行が格納されている。対応ソースブロックＩＤには、変数の出現行を範囲に含んでいるソースブロックのソースブロックＩＤが格納されている。利用種別フラグには、当該行で変数に値が書き込まれている場合は”書き込み”が、変数の値を読み込んでいる場合は”読み込み”が格納されている。ここで、当該行で書き込みが行われていない場合は読み込みであると判別している。

【００４９】

変数情報テーブル４６５の作成が終了した後は、ソースコード分割部４１１は変数情報テーブル４６５の対応分割ソースブロックＩＤ、利用種別フラグからソースブロック間のデータの読み書き関係を判別し、結果をソースブロック依存関係テーブル４６６に格納する。ソースブロック依存関係テーブル４６６はソースブロックと処理対象データを関連付ける際に用いる。

【００５０】

ソースブロック依存関係テーブル４６６の詳細例を図１８に示す。ソースブロック依存関係テーブル４６６には変数ＩＤ、読み込み分割ソースコードＩＤ、書き込み分割ソースコードＩＤ、実行条件依存フラグが格納されている。変数ＩＤは変数情報テーブル４６５に用いられている変数ＩＤであり、二つのソースブロックで各々読み込み若しくは書き込みが行われている変数の変数ＩＤが格納されている。読み込みソースブロックＩＤには当該変数を読み込んでいるソースブロックＩＤが格納されている。書き込みソースブロックＩＤには当該変数に書き込みが行われているソースブロックＩＤが格納されている。なお、ソースブロック間のデータの読み書き関係を判別する際に、書き込みソースブロックＩＤで表されるソースブロックで書き込みが行われた後、読み込みソースブロックＩＤで表されるソースブロックで読み込みが行われる前に他のソースブロックにおいて必ず上書きが行われるものについては除外してある。実行条件依存フラグには、当該変数を読み込んでいる場所が読み込みソースブロックＩＤで表されるソースブロックの実行条件部分であった場合に”ＴＲＵＥ”が、それ以外の場合は”ＦＡＬＳＥ”が格納されている。実行条件依存フラグは、ソースコード分割結果テーブル４６４の親階層ソースブロックＩＤに値が格納されている場合に処理対象データとソースブロックの対応付けを正しく行うために用いられる。

【００５１】

ソースブロック依存関係テーブル４６６の作成が終了した後は、ソースコード分割部４１１はソースコード内に出現している変数から処理対象データの候補を取り出す。結果は処理対象データ候補テーブル４６７に格納される。

【００５２】

処理対象データ候補テーブル４６７の詳細例を図１９に示す。処理対象データ候補テーブル４６７には変数ＩＤ、ビット数、構造体直下の変数ＩＤが格納されている。変数ＩＤ

は変数情報テーブル４６５の変数ＩＤと同一のものであり、変数情報テーブル４６５への参照に用いられる。ビット数は変数のデータサイズをビットを単位として表したものが格納されている。構造体直下の変数ＩＤには、当該変数が構造体である場合、直下に存在している変数の変数ＩＤが格納されている。ここで、構造体Ａが要素として構造体Ｂを持つ場合、構造体Ｂは構造体Ａの内部に含まれているとし、また、構造体Ａの内部に他の構造体Ｂがあったとき、構造体Ａの内部に含まれている変数のうち構造体Ｂの内部に含まれている変数を除いたものを構造体直下の変数とする。優先順位には処理対象データ候補表示窓１５１に表示される順番が格納されている。処理対象データ候補テーブル４６７の作成が終了した後は、ソースコード分割部４１１は結果を処理対象データ候補表示窓１５１に出力し、ユーザの応答待ちとなる。

【００５３】

ソースブロック分類部４１２はユーザが入力した処理対象データに従ってソースブロックを分類する部分である。まずは、処理対象データ決定窓１５４よりユーザが指定した処理対象データを読み込み、処理対象データテーブル４７１に格納する。

【００５４】

処理対象データテーブル４７１の詳細例を図２０に示す。処理対象データテーブル４７１には処理対象データＩＤ、変数ＩＤが格納されている。処理対象データＩＤには対象の変数を一意に識別するためのキーが格納されている。処理対象データＩＤの割り当ては重複がなければ任意で構わない。変数ＩＤには当該変数に対して変数情報テーブル４６５内で割り当てられている変数ＩＤが格納されている。

【００５５】

処理対象データテーブル４７１の作成が終了した後は、ソースブロック分類部４１２は処理対象データとソースブロックの対応関係を導く。ここで、処理対象データとソースブロックが対応関係を持つとは、ソースブロックの除去により処理対象データの値が変化する場合が存在することとする。処理対象データとソースブロックの対応関係判別にはソースコード分割結果テーブル４６４、ソースブロック依存関係テーブル４６６、処理対象データテーブル４７１を用いる。結果はソースブロック分類テーブル４７２に格納する。ソースブロック分類テーブル４７２はソースブロック分類部４１２の主要な成果物である。

【００５６】

ソースブロック分類テーブル４７２の詳細例を図２１に示す。ソースブロック分類テーブル４７２は処理対象データごとに用意されるテーブルで、処理対象データごとにソースブロックＩＤ、関係処理フラグ、条件処理フラグが格納されている。ソースブロックＩＤには処理対象データと対応関係を持つソースブロックのソースブロックＩＤが格納されている。関係処理フラグには、当該ソースブロックに対し関係処理がまだ行われていない場合”未処理”が、関係処理が行われている最中は”処理中”が、関係処理が終了した後には”処理済”が格納されている。ここで、関係処理とは、ソースブロック依存関係テーブル４６６から依存関係を持つソースブロックを連鎖的に探索していく処理である。条件処理フラグには、当該ソースブロックに対し条件処理がまだ行われていない場合”未処理”が、条件処理が行われている最中は”処理中”が、条件処理が終了した後には”処理済”が格納されている。ここで、条件処理とは、あるソースブロックの実行条件に変数が含まれているとき，当該変数を書き換える別のソースブロックを探索する処理である。条件処理は、ソースコード分割結果テーブル４６４の当該ソースブロックに対応する行の親階層ソースブロックＩＤに値が格納されており、かつソースブロック依存関係テーブル４６６の読み込みソースブロックＩＤに親階層ソースブロックＩＤに対応するソースブロックが格納されており、かつ当該行の実行条件フラグが”ＴＲＵＥ”である場合に行われる。

【００５７】

ソースブロック分類部４１２の処理の概念を図５を用いて説明する。図５では、ソースブロック分類部４１２によってソースブロック１、２、５は処理対象データｄａｔａＡと、ソースブロック１、３、５は処理対象データｄａｔａＢと、ソースブロック１、４、５は処理対象データｄａｔａＣと対応関係を持っていると判定されている。ソースブロック

分類テーブル４７２の作成が終了した後は、グループ化用マーカ抽出部４１３が実行される。

【００５８】

グループ化用マーカ抽出部４１３は、処理対象データとソースブロックの対応関係に基づき、一つのソフトウェア部品として資産化が可能なソースブロック群をグループ化する際に用いるマーカを抽出する部分である。本実施例では、マーカとしてすべての処理対象データと対応関係を持つソースブロック（以下、共通ソースブロックと記述）を用いる。共通ソースブロックの判別にはソースブロック分類テーブル４７２を用い、結果は共通ソースブロックリスト４８１に格納される。共通ソースブロックリスト４８１はグループ化用マーカ抽出部４１３の主要な成果物である。

【００５９】

共通ソースブロックリスト４８１の詳細例を図２２に示す。共通ソースブロックリスト４８１には共通ソースブロックと判定されたソースブロックのソースブロックＩＤが格納されている。グループ化用マーカ抽出部４１３の処理概念を図５を用いて説明する。図５では、全ての処理対象データ（ｄａｔａＡ、ｄａｔａＢ、ｄａｔａＣ）と対応関係を持つソースブロック１、５が共通ソースブロックと判定されている。共通ソースブロックリスト４８１の作成が終了した後は、ソースブロックグループ化部４１４が実行される。

【００６０】

ソースブロックグループ化部４１４は、グループ化用マーカ抽出部４１３で抽出されたマーカを用いて一つのソフトウェア部品として資産化が可能なソースブロック群をグループ化する部分である。また、グループ総数や、グループ内に含まれるソースブロック群のソースコードの一致行数を算出し、部品化による部品点数やソースコード量削減効果などの指標をユーザに提示する。グループの判別には、共通ソースブロックリスト４８１に格納されている二つの共通ソースブロックの組み合わせを用いる。二つの共通ソースブロックの組み合わせに対し、それらを結ぶプログラム実行経路のうち他の共通ソースブロックを含まないものを判別し、実行経路上に存在するソースブロック群を一つのグループとする。結果はソースブロックグループ化テーブル　４９１に格納する。ソースブロックグループ化テーブル　４９１はソースブロックグループ化部４１４の主要な成果物である。

【００６１】

ソースブロックグループ化テーブル　４９１の詳細を図２３に示す。ソースブロックグループ化テーブル　４９１にはグループＩＤ、共通ソースブロックＩＤ（前）、共通ソースブロックＩＤ（後）、グループ内ソースブロックＩＤが格納されている。グループＩＤには対象のグループを一意に識別するためのキーが格納されている。グループＩＤの割り当ては重複がなければ任意で構わない。共通ソースブロックＩＤ（前）はグループ化のマーカとなっている二つの共通ソースブロックのうち、プログラム上の実行順序が前であるものの共通ソースブロックＩＤが格納されている。共通ソースブロックＩＤ（後）はグループ化のマーカとなっている二つの共通ソースブロックのうち、プログラム上の実行順序が後であるものの共通ソースブロックＩＤが格納されている。グループ内ソースブロックＩＤには、対象グループに属するソースブロックのソースブロックＩＤが格納されている。

【００６２】

ソースブロックグループ化テーブル　４９１の作成が終了した後は、ソースブロックグループ化部４１４はグループ化結果に対する分析処理を行い、結果をメトリクス表示窓２４０に出力する。分析処理では、例えばソースブロックの総数、グループ数、ソースコードの総行数、部品化により削減可能なソースコード行数、部品化により削減可能なソースコード行数をソースコードの総行数で除したものであるソースコード削減率などを算出する。分析処理の終了後は、ソースブロックグループ化部４１４が結果を画面管理部４３１に受け渡し、ユーザの応答待ちとなる。ソースブロックグループ化部４１４の処理概念図を図５を用いて説明する。図５では、共通ソースブロックであるソースブロック１、５を結ぶプログラム実行経路上にあるソースブロックであるソースブロック２、３、４が一つ

のグループと判定されている。

【００６３】

コントローラ部４２１は画面管理部４３１からユーザの入力情報を受け取り、ソースコード分割部４１１、ソースブロック分類部４１２、グループ化用マーカ抽出部４１３、ソースブロックグループ化部４１４を制御する部分である。画面管理部４３１は入力インタフェース４４１からユーザの入力情報を受け取り、コントローラ部４２１へと受け渡す。また、ソースコード分割部４１１、ソースブロック分類部４１２、グループ化用マーカ抽出部４１３、ソースブロックグループ化部４１４の処理結果を受け取り、画面出力を出力インタフェース４５１へと受け渡す。入力インタフェース４４１はユーザからの入力を受け取り、画面管理部４３１へと受け渡す。出力インタフェース４５１は画面管理部から受け取った画面出力をディスプレイなどのユーザインタフェースへと出力する。

【００６４】

図６はソフトウェア資産整理装置１００の処理の流れを表している。処理６００は分割実行ボタン１４０の押下をトリガとして開始され、処理７００、８００、９００、１０００が実行される。

【００６５】

処理７００は、対象ソースコードをインライン展開した後、構文解析により制御構文を解析してソースコードの分割を行う処理であり、ソースコード分割部４１１によって実行される。処理７００の詳細な流れを図７に示す。７００における成果物はソースコード分割条件リスト４６１、インライン展開ソースファイル４６２、制御構文情報テーブル４６３、ソースコード分割結果テーブル４６４であり、特に主要な成果物はソースコード分割結果テーブル４６４である。処理７００は入力の読み込み処理である７１１、７１２、構文解析処理である７２１、７２２、７２３、７２４、７２５、７２６、ソースコード分割処理である７３１、７３２、７３３、７３４から構成される。

【００６６】

７１１では、条件設定窓１２０に入力された解析条件を読み込む。ここで、解析条件とは例えば分析対象のソースコードの記述言語やソースコード分割条件である。ソースコード分割条件は例えば分割時の区切りとして用いる制御構文として与えられる。

【００６７】

７１２では、７１１で読み込んだ解析条件をソースコード分割条件リスト４６１に格納する。

【００６８】

７２１では、ユーザによりソースファイル決定窓１１４に設定されたソースファイルを読み込む。読み込み対象のソースファイルは図１のソースファイル決定窓１１４においてチェックボックスにレ点が入っているソースファイルである。

【００６９】

７２２では、読み込んだソースファイルをインライン展開し、インライン展開ソースファイル４６２に格納する。

【００７０】

７２３では、インライン展開ソースファイル４６２に対し、ソースコード分割条件リスト４６１内の記述言語に格納されている情報を用いることでインライン展開ソースファイル４６２に出現する制御構文を検出する。例えば記述言語がＣ言語であれば、ｉｆ文やｆｏｒ文、ｗｈｉｌｅ文、ｓｗｉｔｃｈ文などを検出する。検出された制御構文には制御構文ＩＤを割り当てる。制御構文ＩＤの割り当ては重複がなければ任意で構わない。

【００７１】

７２４では、７２３で検出された各制御構文について、インライン展開ソースファイル４６２における位置、制御範囲、実行条件を判定する。ここで、位置とは制御構文のインライン展開ソースファイル４６２内での出現行である。制御範囲とは、例えばＣ言語では制御構文後に”｛“と”｝”が出現する場合、それらに囲まれた範囲のことである。実行条件とは制御範囲内のソースコードがプログラム内で実行されるために必要な条件である

。例えばＣ言語のｉｆ文では、直後の”（“と”）”に囲まれた式の値が１となる条件である。

【００７２】

７２５では、制御構文間の階層関係を判定する。ここで、階層関係とは、制御構文Ａの内部に別の制御構文Ｂが出現する場合、すなわちＡ⊃Ｂの場合、ＢはＡを親階層として持つとする。

【００７３】

７２６では、７２４、７２５で判定された結果を制御構文情報テーブル４６３に格納する。

【００７４】

７３１では、制御構文情報テーブル４６３に格納されている制御構文のうち、ソースコード分割条件リスト４６１内の分割条件と一致する制御構文を検出する。

【００７５】

７３２では、７３１で検出された制御構文からソースブロックを決定する。検出された制御構文の制御範囲を一つのソースブロックとする。制御構文Ａの内部に別の制御構文Ｂが出現する場合、すなわちＡ⊃Ｂの場合、ＢおよびＡからＢを除いたもの、すなわちＢ、Ａ∩￢Ｂをソースブロックとする。いずれのソースブロックにも属さないソースコード、例えばメイン関数直下に記述されているソースコードは、常に真である実行条件を持つ条件文の内部に存在するとみなし、前後に出現する別のソースブロックを区切りとするソースブロックとする。

【００７６】

７３３では、７３２で検出されたソースブロックに識別用のＩＤとしてソースブロックＩＤを割り当てる。ソースブロックＩＤの割り当ては重複がなければ任意で構わない。

【００７７】

７３４では、ソースブロックＩＤ、ソースブロックの位置、範囲、実行条件をソースコード分割結果テーブル４６４に格納する。

【００７８】

図６に戻って、処理７００が終了した後はソースコード分割部４１１によって処理８００が実行される。処理８００はインライン展開ソースファイル４６２から変数を抽出し、各変数とソースブロックを対応付ける処理である。

【００７９】

処理８００の詳細な流れを図８に示す。処理８００の成果物は変数情報テーブル４６５である。処理８００はデータ読み込み処理である８１１、８１２、８１３、変数抽出処理である８２１、８２２、変数とソースブロックの対応付け処理である８３１、８３２、８３３、８３４、８３５から構成される。処理８００ではソースコード中に出現する変数を検出し、各箇所において読み込みと書き込みのどちらが行われているかを判定する。ここで、変数とは構造体も含む。

【００８０】

８１１では、ソースコード分割条件リスト４６１の記述言語に格納されている情報を読み込む。

【００８１】

８１２では、インライン展開ソースファイル４６２からインライン展開されたソースコードを読み込む。

【００８２】

８１３では、ソースコード分割結果テーブル４６４の情報を読み込む。

【００８３】

８２１では、インライン展開ソースファイル４６２に出現するすべての変数を検出する。この際、変数定義を参照することにより変数のスコープも同時に判定する。変数のスコープとは、ソースコード上において当該変数に読み書きが行える範囲である。検出された変数には変数ＩＤを割り当てる。変数ＩＤの割り当ては重複がなければ任意で構わない。

【００８４】
８２２では、８２１で検出された変数の変数ＩＤ、変数名、スコープ、変数型、出現位置を変数情報テーブル４６５に格納する。
【００８５】
８３１は反復処理であり、変数情報テーブル４６５に格納されている各変数のすべての出現場所について８３２、８３３、８３４、８３５、８３６を実行することを示している。
【００８６】
８３２では、対象としている出現場所でデータへの書き込みが行われているかを判定する。書き込みが行われていた場合は、８３３で当該箇所における利用種別フラグを“書き込み”に設定する。書き込みが行われていなかった場合は読み込みと判断し、８３６で当該箇所における利用種別フラグを“読み込み”に設定する。
【００８７】
８３４では、ソースコード分割結果テーブル４６４の開始行と終了行を参照することにより、当該出現場所を内部に含むソースブロックを検索し、ソースブロックＩＤを得る。８３５では、変数情報テーブル４６５に出現場所、対応ソースブロックＩＤ、利用種別フラグを格納する。
【００８８】
図６に戻って、処理８００が終了した後は処理９００が実行される。処理９００はソースコード分割部４１１によって実行され、各ソースブロック内部における変数の読み書きに基づいて、ソースブロック間の依存関係を求める処理である。
【００８９】
処理９００の詳細な流れを図９に示す。処理９００の成果物は処理対象データ候補テーブル４６７である。処理９００は９１１、９１２、９１３、９１４、９１５、９１６、９１７から構成される。
【００９０】
９１１は反復処理であり、ソースコード分割結果テーブル４６４に格納されている各ソースブロックについて９１２、９１３、９１４、９１５、９１６、９１７、９１８を実行することを示している。また、９１２も反復処理であり、ソースブロック内部に出現する変数のうち、利用種別フラグが”読み込み”となる出現位置が存在するものについて９１３、９１４、９１５、９１６、９１７、９１８を実行することを示している。
【００９１】
９１３では、変数情報テーブル４６５のうち、反復処理９１２において現在処理対象となっている変数に該当する部分を参照し、利用種別フラグが“書き込み”となっているソースブロックをすべて検出する。
【００９２】
９１４では、９１３で検出されたソースブロックのうち、反復処理９１１で現在処理対象となっているソースブロックとの間に依存関係があるものを抽出する。依存関係の判定は、反復処理９１１の処理対象ソースブロックと９１３で検出されたソースブロックを結ぶプログラムの実行経路を考えることにより行う。すなわち、９１３で検出されたソースブロックの中から一を選択し、反復処理９１１で現在処理対象となっているソースブロックと選択された当該一のソースブロックとの間に、９１３で検出された他のソースブロックが含まれない実行経路が存在した場合にこれら２つのソースブロック間に依存関係ありとする。
【００９３】
９１５では、９１４で依存関係ありと判定された２つのソースブロックについて、ソースブロック依存関係テーブル４６５に読み込み分割ソースコードＩＤ、書き込み分割ソースコードＩＤと、反復処理９１２で現在処理対象としている変数を格納する。
【００９４】
９１６では、反復処理９１２で現在処理対象としている変数が反復処理９１１で現在処

理対象としているソースブロックの実行条件内に出現するかどうかを判定する。ソースブロックの実行条件内に出現していた場合は、９１７でソースブロック依存関係テーブル４６５内の当該項目の実行条件依存フラグを”ＴＲＵＥ”とする。そうでない場合は、９１８で実行条件依存フラグを”ＦＡＬＳＥ”とする。実行条件依存フラグは、ソースコード分割結果テーブル４６４の親階層ソースブロックＩＤに値が格納されている場合に、処理対象データとソースブロックの対応付けを正しく行うために以降の処理で用いる。

【００９５】

図６に戻って、処理９００が終了した後は処理１０００が実行される。処理１０００はソースコード分割部４１１によって実行され、ユーザに提示する処理対象データの候補を算出する処理である。

【００９６】

処理１０００の詳細な流れを図１０に示す。処理１０００の成果物は処理対象データ候補テーブル４６７である。１０００は１０１１、１０１２、１０１３、１０１４、１０１５から構成される。

【００９７】

１０１１では、変数情報テーブル４６５から変数情報を読込む。

【００９８】

１０１２では、１０１１で読み込まれた変数から、プログラム全体をスコープに含んでいる変数を抽出する。これにはソースコード内に出現しているグローバル変数も含める。変数情報テーブル４６５と同様に構造体も一つの変数として扱う。

【００９９】

１０１３では、１０１２で抽出された変数について、階層構造、変数間の参照・被参照の関係を解析する。ここで、変数が構造体である場合、構造体Ａが要素として構造体Ｂを持つなら、構造体Ｂは構造体Ａの内部に含まれているとする。また、構造体Ａの内部に他の構造体Ｂがあったとき、構造体Ａの内部に含まれている変数のうち構造体Ｂの内部に含まれている変数を除いたものを構造体直下の変数とする。変数Ｃ、Ｄが構造体Ａの直下にあった場合、ＣとＤは並列関係にあるとする。

【０１００】

１０１４では、１０１２で抽出された変数のデータサイズをビット数で表したものを算出する。

【０１０１】

１０１５では、１０１２で抽出された変数の優先順位を算出する。優先順位の算出は、例えば並列関係にある変数の数をカウントし、数が多いものから昇順に優先順位を割り当てていくことにより決定する。この際、並列関係にある変数の数が同一であった場合、優先順位も同一とする。ここで、優先順位とは、処理対象データ候補のうち、ユーザに処理対象データとして選択される可能性の高さを表す指標であり、本実施例の方法以外にも、プログラム内における変数の出現回数、構造体のサイズ、プログラム末尾から辿った書き込みが順序などを用いて算出してもよい。

【０１０２】

１０１６では、１０１２、１０１３、１０１４、１０１５の結果に基き処理対象データ候補テーブル４６７に変数ＩＤ、ビット数、構造体直下の変数ＩＤ、優先順位を格納する。

【０１０３】

１０１６の終了後は、処理は一時停止しユーザの入力待ち状態となる。この際、画面管理部４３１は処理対象データ候補テーブル４６７の内容をもとに出力インタフェース４５１を介して処理対象変数候補表示窓１５１に処理対象変数の候補を表示する。この際、処理対象変数の候補は処理対象データ候補テーブル４６７の優先順位に従って表示する。

【０１０４】

図６に戻って、ユーザによる処理対象変数の選択後、グループ化実行ボタン１７０の押下をトリガとして処理１１００、１２００、１３００が実行される。処理１１００、１２

００はソースブロック分類部４１２によって実行され、ユーザが選択した処理対象変数とソースブロックの関係を、ソースブロック依存関係テーブル４６６を用いて判定する処理である。

【０１０５】

処理１１００、１２００の詳細な流れを図１１、図１２に示す処理。１１００、１２００の成果物はソースブロック分類テーブル４７２である。処理１１００、１２００は、直接処理対象データを書き換えるソースブロックを判定する処理である１１１１、１１１２、１１１３、ソースブロック間の依存関係から処理対象データの書き換えに影響しうるソースブロックを判定する処理である１１３１、１１３２、１１３３、１１３４、１１３５、１１３６、親階層ソースブロックが存在した場合に親階層ソースブロックの実行に影響を与えるソースブロックを検出する処理である１１２１、１２１１、１２１２、１２１３、１２１４、１２１５、１２１６、１２１７から構成される。　１１１１では、処理対象データ決定窓１５４においてユーザが選択した処理対象変数を読み込み、処理対象変数テーブル４７１に格納する。

【０１０６】

１１１２では、ソースブロック依存関係テーブル４６６を参照し、１１１１で読み込んだ処理対象変数の利用種別フラグが”書き込み”となっているソースブロックを検索する。

【０１０７】

１１１３では、１１１２で検索されたソースブロックについて、ソースブロック分類テーブル４７２の処理対象データ項に当該処理対象変数を追加し、関係処理フラグを”処理中”にセットする。ここで、関係処理フラグには、当該ソースブロックに対し関係処理がまだ行われていない場合”未処理”が、関係処理が行われている最中は”処理中”が、関係処理が終了した後には”処理済”が格納されている。関係処理とは、ソースブロック依存関係テーブル４６６から依存関係を持つソースブロックを連鎖的に探索していく処理であり、１１３１、１１３２、１１３３、１１３４、１１３５、１１３６を指す。

【０１０８】

１１２１は無限ループであり、１２１７の条件が満たされるまで反復される。１１３１も無限ループであり、１１３６の条件が満たされるまで反復される。

【０１０９】

１１３２では、ソースブロック分類テーブル４７２から、関係処理フラグが”処理中”であるソースブロックを検索する。

【０１１０】

１１３３では、ソースブロック依存関係テーブル４６６から、１１３２で検索されたソースブロックＩＤを読込みソースブロック項に持っている行を検索し、当該行の書込みソースブロック項にあるソースブロックＩＤを取り出す。

【０１１１】

１１３４では、１１３３で取り出されたソースブロックＩＤをソースブロック分類テーブル４７２に追加し、関係処理フラグを”処理中”にセットする。

【０１１２】

１１３５では、１１３２で検索されたソースブロックの関係処理フラグを”処理済”にセットする。

【０１１３】

１１３６では、１１３２で関係処理フラグが”処理中”であるソースブロックが一つ以上検出されたかを判定する。一つ以上検出されていた場合、１１３２、１１３３、１１３４、１１３５を再び実行する。一つも検出されたかった場合、処理１２００へと処理を進める。

【０１１４】

１２１１では、ソースブロック分類テーブル４７２から、関係処理フラグが”処理済”であり、かつ条件処理フラグが”未処理”であるソースブロックを検索する。ここで、条

件処理フラグには、当該ソースブロックに対し条件処理がまだ行われていない場合”未処理”が、条件処理が行われている最中は”処理中”が、条件処理が終了した後には”処理済”が格納されている。条件処理とは、ソースコード分割結果テーブル４６４の当該ソースブロックに対応する行の親階層ソースブロックＩＤに値が格納されており、かつソースブロック依存関係テーブル４６６の読み込みソースブロックＩＤに親階層ソースブロックＩＤに対応するソースブロックが格納されており、かつ当該行の実行条件フラグが”ＴＲＵＥ”である場合に行われる処理であり、１１２１、１２１１、１２１２、１２１３、１２１４、１２１５、１２１６、１２１７を指す。

【０１１５】

１２１２では、ソースコード分割結果テーブル４６４から１２１１で検索されたソースブロックを検索する。

【０１１６】

１２１３では、１２１２で検索されたソースブロックに対し、親階層ソースブロックを判定する。その後、判定された親階層ソースブロックについて、さらにその親階層ソースブロックを判定するという処理を親階層ソースブロックがなくなるまで繰り返す。最後に、それぞれの親階層ソースブロックのソースブロックＩＤを抽出する。

１２１４では、ソースブロック依存関係テーブル４６６を参照し、１２１３で検索されたソースブロックを読込みソースブロックとして持ち、かつ実行条件依存フラグが”ＴＲＵＥ”である行を検索する。すなわち、１２１３で検索されたソースブロックの実行条件に影響を及ぼしうるソースブロックを検索する。

【０１１７】

１２１５では、１２１４で検索された行の書込みソースブロック項に格納されているソースブロックＩＤをソースブロック分類テーブル４７２に追加し、関係処理フラグを”処理中”にセットする。

【０１１８】

１２１６では、１２１１で検索されたソースブロックの条件処理フラグを”処理済”にセットする。

【０１１９】

１２１７では、１２１１でソースブロックが一つ以上検出されたかどうかを判定する。一つ以上検出されていた場合、１１３１の処理を再び実行する。一つも検出されたかった場合、１２００を終了する。

【０１２０】

図６に戻って、処理１２００が終了した後は処理１３００が実行される。処理１３００はグループ化用マーカ抽出部４１３とソースブロックグループ化部４１４によって実行され、グループ化用マーカとして共通ソースブロックを抽出し、共通ソースブロックの位置からソフトウェア部品として資産化可能なソースブロック群をグループ化する処理である。

【０１２１】

処理１３００の詳細な流れを図１３に示す。処理１３００の成果物は共通ソースブロックリスト４８１、ソースブロックグループ化テーブル４９１である。処理１３００は共通ソースコードを判別する処理である１３１１、１３１２、共通ソースブロックを用いてソースブロックをグループ化する処理である１３２１、１３２２、１３２３、１３２４、１３２５、グループに関する統計情報を算出し結果を出力する１３３１、１３３２、１３３３から構成される。

【０１２２】

１３１１では、ソースブロック分類テーブル４７２から、すべての処理対象変数と対応関係を持っているソースブロックを検索する。

【０１２３】

１３１２では、１３１１で検索されたソースブロックを共通ソースブロックリスト４８１に追加する。

【０１２４】
１３２１では、共通ソースブロックリスト４８１から、二つの共通ソースブロックを結ぶプログラムの実行経路として他の共通ソースブロックを経由しないものが存在する組み合わせを検索する。
【０１２５】
１３２２では、１３２１で検索された共通ソースブロックの組をソースブロックグループ化テーブル４９１に追加する。
【０１２６】
１３２３では、１３２２で検索された共通ソースブロックの組について、両者を結ぶプログラム実行経路上に存在するソースブロックをすべて抽出する。これらのソースブロック群は一つのソフトウェア部品として資産化をするための候補となる。
【０１２７】
１３２４では、１３２３で抽出されたソースブロック間のソースコード比較を行い、一致率が部品化判定条件設定窓１６０から設定された閾値を上回ったものを同一グループと判定する。一致率は、例えばソースブロックをトークン(語彙素)の列に分割し、２つのソースブロックで一致したトークン数を求めた後、２つのソースブロックの総トークン数のうち小さいもので除すことにより算出する。ここで、一致したトークン数は３つの手順により求める。手順１として、それぞれのソースブロックから生成されたトークン列に対し、任意の位置にあるトークンを除去することにより部分トークン列を作成する。例えば、トークン列[a,b,c]の部分トークン列は[]、[a]、[b]、[c]、[a,b]、[a,c]、[b,c]、[a,b,c]の８個存在する。手順２として、２つのソースブロックのそれぞれから部分トークン列を１つずつ選び、同一位置にあるトークンが一致しているものの数をカウントする。手順３として、すべての部分トークン列の集合に対し手順２を繰り返し、最大の値を一致したトークン数とする。
【０１２８】
１３２５では、１３２４で同一グループと判定されたソースブロック群をソースブロックグループ化テーブル４９１に格納する。
【０１２９】
１３３１では、ソースブロックグループ化テーブル４９１より、各グループに属するソースブロック間のソースコード比較を行い、ソースコードの一致部分と相違部分を判定する。
【０１３０】
１３３２では、統計情報として、例えばソースコード削減率などの指標を算出する。ソースコード削減率はソースコードの一致部分から重複ソースコードを判定し、重複ソースコードの行数の総和をソースコードの総行数で除したものとして算出できる。
【０１３１】
１３３３では、ソースブロックグループ化テーブル４９１から同一グループと判定されたソースブロック群を統合しファイルに出力する。
【０１３２】
１３３３が終了した後は、ソースブロック分類テーブル４７２、共通ソースブロックリスト　４８１、ソースブロックグループ化テーブル４９１を画面管理部４３１に受け渡し、処理は終了となる。
【０１３３】
以上のような実施例によれば、既存のソフトウェア資産のうち、類似性の高いソースコード群を抽出し部品化することができる。また、ユーザはＧＵＩへの提示情報に基づいて部品化結果の修正を行うことができる。
【０１３４】
、更に本実施形態によれば、あらかじめ処理の区切りに従ってソースコードを分割し、分割により生成されたソースブロックの間で部品化判定を行うため、細かい一致コード断片といったノイズが発生せず、計算資源の浪費を抑えることが出来る。また、ソースコー

ドの分割およびソースブロックの関連付けは、処理対象データや部品化判定の条件を変更しても再実行する必要がないため、条件のチューニングによる再計算のコストを大幅に減らすことが出来る。さらに、制御構文によりソースコードを分割し、処理の実行順序が近いソースブロック間で部品化判定を行うため、プログラムの意味的なまとまりを崩すことなく部品化が可能である。他にも、処理対象のデータに固有な処理を検出できるため、データと処理や機能の関連性分析や、改修による処理対象データ追加時の影響範囲推定といった用途に用いることも出来る。

【０１３５】

鉄道運行管理システムでは、サブシステムである旅客案内システムひとつをとってみても、列車の行先や停車駅、両数など実にさまざまなデータが存在する。また、東京圏の鉄道では一日に数千本といった単位で列車が運行されており、データの種類だけではなく量も膨大である。このようなシステムにおいて、処理対象のデータごとに作成されたソースコードを部品化するためには多大な労力を要するが、本発明によればプログラム実行順序をほとんど崩すことなく部品化が実現でき、かつノイズ判定といった追加工数も発生しないため、極めて効率的に作業を行うことが出来る。

【符号の説明】

【０１３６】

１００：ソフトウェア資産整理装置
４１１：ソースコード分割部
４１２：ソースブロック分類部
４１３：グループ化用マーカ抽出部
４１４：ソースブロックグループ化部
４３１：画面間理部

【図１】

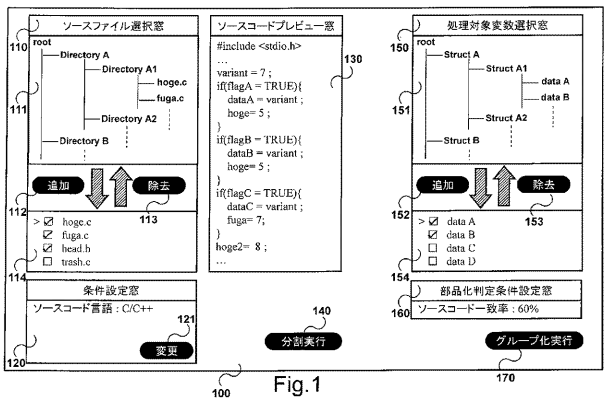


Fig.1

【図２】

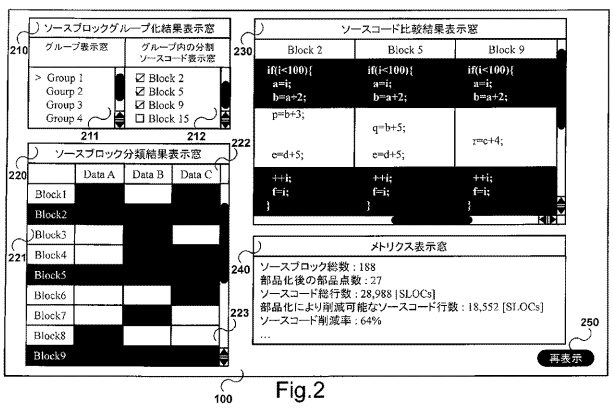


Fig.2

【図３】

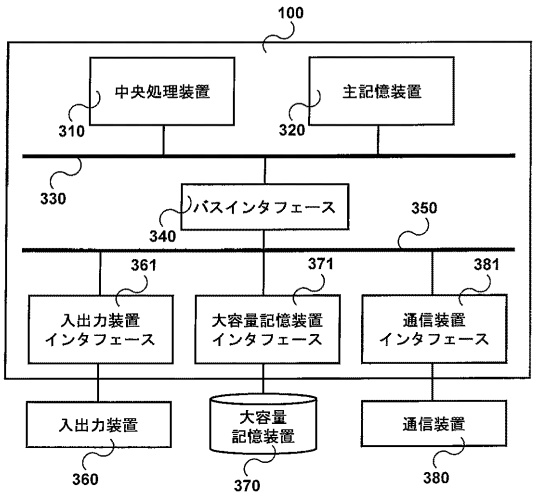


Fig.3

【図４】

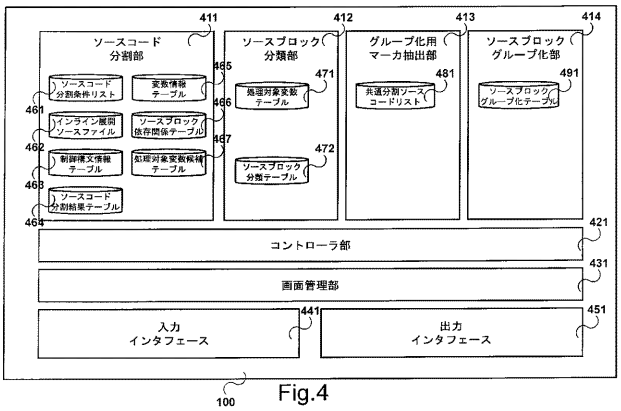


Fig.4

【図５】

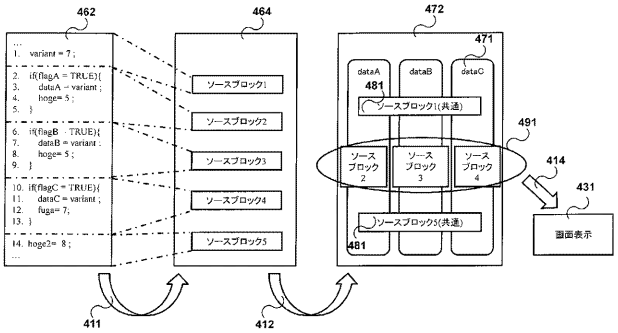


Fig.5

【図６】

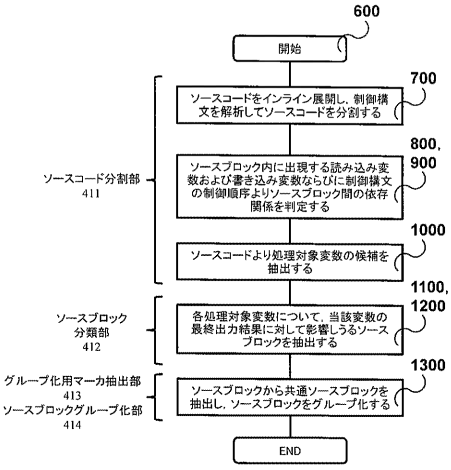


Fig.6

【図７】

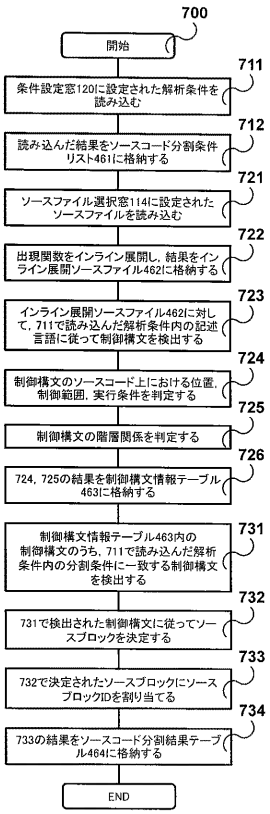


Fig.7

【図８】

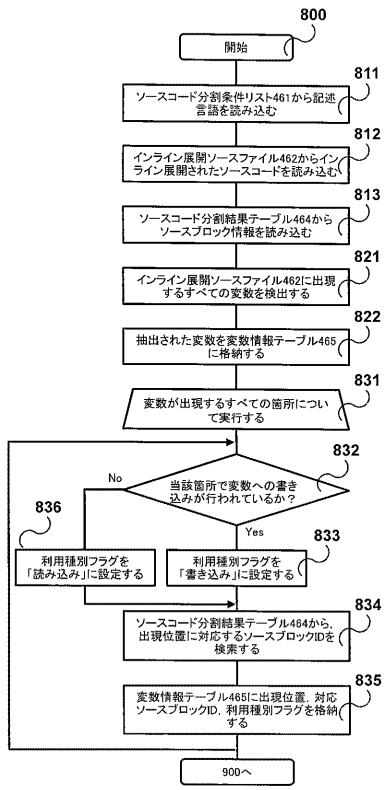


Fig.8

【図９】

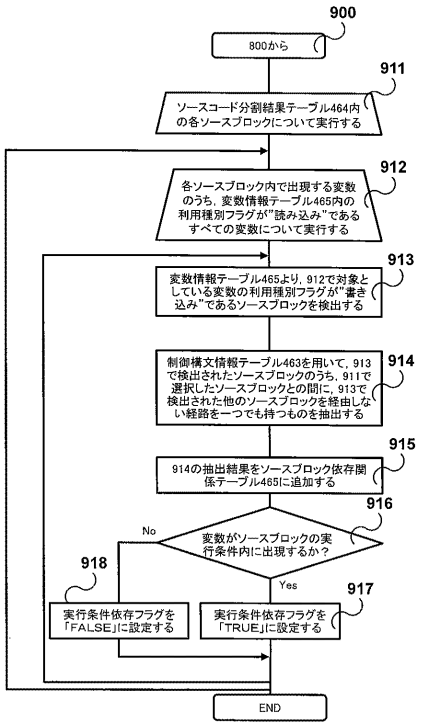


Fig.9

【図１０】

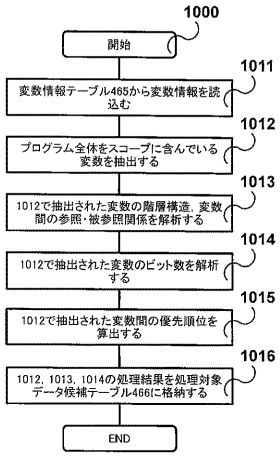


Fig.10

【図１１】

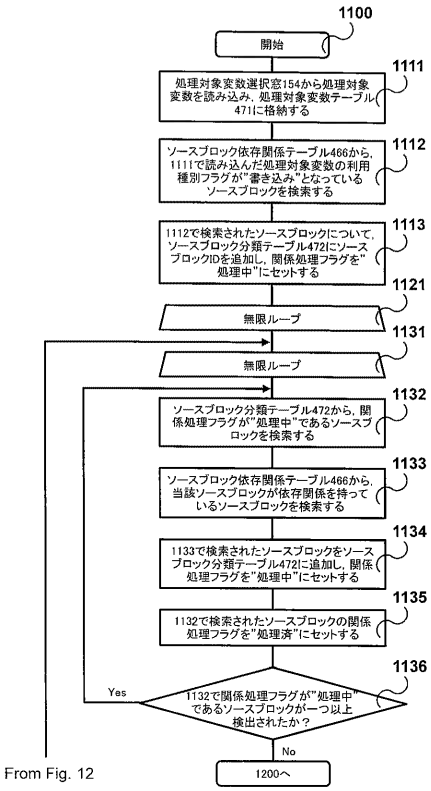


Fig.11

【図１２】

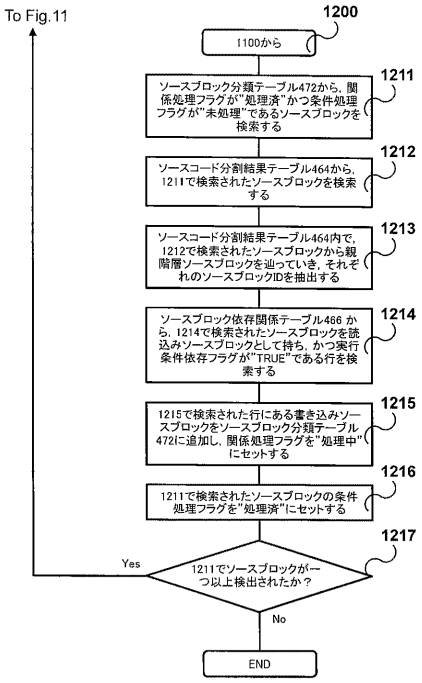


Fig.12

【図１３】

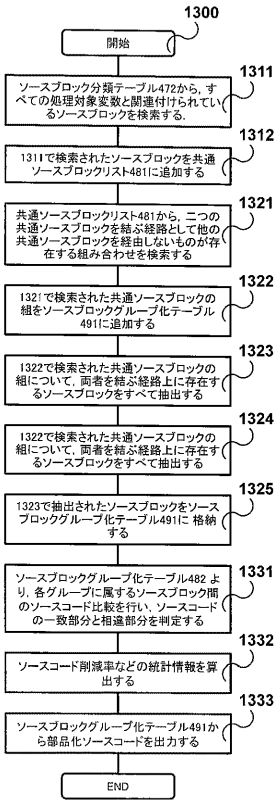


Fig.13

【図１４】

ソースコード分割条件リスト 461

ソースコード言語設定	ソースコード分割条件
C/C++	条件分岐
…	

Fig.14

【図１５】

制御構文情報テーブル 463

制御構文ID	開始位置	終了位置	制御構文	実行条件	親階層制御構文ID
1	LINE 72	LINE 77	if	i < j	7
2	LINE 85	LINE 124	for	j=0; j<10; ++j	11
…	…	…	…	…	…

Fig.15

【図１６】

ソースコード分割結果テーブル 464

ソースブロックID	開始位置	終了位置	実行条件	親階層ソースブロックID
1	LINE 172	LINE 177	i < j	7
2	LINE 186	LINE 224	hoge == 10	11
…	…	…	…	…

Fig.16

【図１７】

変数情報テーブル 465

変数ID	変数名	スコープ	変数型
1	i	LINE 55 - LINE 83	int
2	f	LINE 3 - LINE 982	float
…	…	…	…

変数ID：1

出現位置	対応分割ソースコードID	利用種別フラグ
LINE 272	3	読み込み
LINE 285	8	書き込み
…	…	…

Fig.17

【図１８】

ソースブロック依存関係テーブル 466

読込み分割ソースコードID	書込み分割ソースコードID	変数ID	実行条件依存フラグ
3	6	12	TRUE
6	8	14	FALSE
…	…	…	…

Fig.18

【図１９】

処理対象データ候補テーブル 467

変数ID	ビット数	構造体直下の変数ID	優先順位
3	128	4,5,6	5
4	16	–	1
…	…	…	…

Fig.19

【図２０】

処理対象データテーブル 471

処理対象データID	変数ID
1	15
2	22
…	…

Fig.20

【図２１】

ソースブロック分類テーブル 472

処理対象変数ID：5

ソースブロックID	関係処理フラグ	条件処理フラグ
5	処理済	処理中
8	処理中	未処理
…	…	…

Fig.21

【図２２】

共通ソースブロックリスト 481

ソースブロックID
5
9
…

Fig.22

【図２３】

ソースブロックグループ化テーブル 491

グループID	共通ソースブロックID(前)	共通ソースブロックID(後)	グループ内のソースブロックID
1	5	9	5, 7, 8
2	9	13	10, 11, 12
…	…	…	…

Fig.23

フロントページの続き

(72)発明者　志村　明俊
神奈川県横浜市戸塚区吉田町２９２番地　株式会社日立製作所システム開発研究所内
(72)発明者　武沢　隆之
茨城県日立市大みか町五丁目２番１号　株式会社日立製作所情報制御システム社内

Ｆターム(参考) 5B376 BC12　BC13　DA21　DA24　FA13