(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4924976号
(P4924976)
(45)発行日　平成24年4月25日(2012.4.25)　(24)登録日　平成24年2月17日(2012.2.17)

(51)Int.Cl.　F I
G06F 9/44 (2006.01)　G06F 9/06 620A

請求項の数 4 (全 21 頁)

(21)出願番号　特願2005-83931(P2005-83931)
(22)出願日　平成17年3月23日(2005.3.23)
(65)公開番号　特開2006-268299(P2006-268299A)
(43)公開日　平成18年10月5日(2006.10.5)
審査請求日　平成20年3月21日(2008.3.21)

前置審査

(73)特許権者　000191076
新日鉄ソリューションズ株式会社
東京都中央区新川２丁目２０番１５号
(74)代理人　100091269
弁理士　半田　昌男
(72)発明者　辰巳　哲
東京都中央区新川二丁目２０番１５号　新日鉄ソリューションズ株式会社内
(72)発明者　田尾　公紀
東京都中央区新川二丁目２０番１５号　新日鉄ソリューションズ株式会社内
(72)発明者　斉藤　康弘
東京都中央区新川二丁目２０番１５号　新日鉄ソリューションズ株式会社内

最終頁に続く

(54)【発明の名称】ソフトウェア開発支援システム

(57)【特許請求の範囲】
【請求項１】
開発者が表示手段の画面上に表示されたソースコードの内容を追加・変更しながらソフトウェアの開発を行うために使用されるソフトウェア開発支援システムにおいて、
設計情報に基づいて、一又は複数の挿入マーカがそれぞれ所定の箇所に挿入されたソースコードを生成するソースコード生成手段と、
前記ソースコードのうち前記各挿入マーカで指定される範囲に含まれるコードを挿入コードとして抽出する挿入コード抽出手段と、
前記挿入コード抽出手段によって抽出された前記各挿入コードを、当該挿入コードを含むソースコード及び当該挿入コードの挿入マーカを識別するための識別情報と関連付けて記憶する挿入コード記憶手段と、
前記ソースコード生成手段が以前生成した前記ソースコードについて再度前記ソースコードを生成したときに、前記挿入コード記憶手段に記憶されている前記挿入コードのうち、前記再度生成したソースコード及び当該ソースコードに存在する挿入マーカを識別するための識別情報と同一の識別情報に関連付けられた挿入コードを特定し、前記特定した挿入コードを、当該再度生成したソースコード中に存在する当該挿入マーカで指定される範囲に結合する処理を行うコード結合手段と、
を具備し、
前記ソースコード生成手段は、前記設計情報に基づいて、前記ソースコードを生成するときに、そのソースコードがどの設計情報に基づいて生成されたのかを識別する情報を付

するものであり、
前記コード結合手段は、前記ソースコード生成手段が以前生成した前記ソースコードについて再度前記ソースコードを生成したときに、前記挿入コード記憶手段に記憶されている前記挿入コードの中に、当該挿入コードについての前記識別情報によって特定される前記ソースコードを識別する前記情報と当該ソースコードを識別する前記情報とが一致するが、当該識別情報によって特定される前記挿入マーカが当該ソースコードに存在しないものが含まれていれば、当該ソースコードに結合されない前記挿入コードが存在することを検出することを特徴とするソフトウェア開発支援システム。
【請求項２】
前記ソースコードを編集すると共に、前記ソースコードを編集する際に前記挿入マーカで指定される範囲以外の範囲におけるコードの入力を制限するコード編集手段を具備することを特徴とする請求項１記載のソフトウェア開発支援システム。
【請求項３】
コンピュータを利用して、開発者が表示手段の画面上に表示されたソースコードの内容を追加・変更しながらソフトウェアの開発を行うためのソフトウェア開発支援プログラムを記録したコンピュータ読み取り可能な記録媒体において、
設計情報に基づいて、一又は複数の挿入マーカがそれぞれ所定の箇所に挿入されたソースコードを生成すると共に、前記設計情報に基づいて、前記ソースコードを生成するときに、そのソースコードがどの設計情報に基づいて生成されたのかを識別する情報を付するソースコード生成機能と、
前記ソースコードのうち前記各挿入マーカで指定される範囲に含まれるコードを挿入コードとして抽出し、その抽出した前記各挿入コードを、当該挿入コードを含むソースコード及び当該挿入コードの挿入マーカを識別するための識別情報と関連付けて挿入コード記憶手段に記憶させる挿入コード抽出機能と、
前記ソースコード生成機能により、以前生成した前記ソースコードについて再度前記ソースコードが生成されたときに、前記挿入コード記憶手段に記憶されている前記挿入コードのうち、前記再度生成したソースコード及び当該ソースコードに存在する挿入マーカを識別するための識別情報と同一の識別情報に関連付けられた挿入コードを特定し、前記特定した挿入コードを、当該再度生成したソースコード中に存在する当該挿入マーカで指定される範囲に結合すると共に、前記ソースコード生成機能により、以前生成した前記ソースコードについて再度前記ソースコードが生成されたときに、前記挿入コード記憶手段に記憶されている前記挿入コードの中に、当該挿入コードについての前記識別情報によって特定される前記ソースコードを識別する前記情報と当該ソースコードを識別する前記情報とが一致するが、当該識別情報によって特定される前記挿入マーカが当該ソースコードに存在しないものが含まれていれば、当該ソースコードに結合されない前記挿入コードが存在することを検出するコード結合機能と、
を前記コンピュータに実現させるためのソフトウェア開発支援プログラムを記録したコンピュータ読み取り可能な記録媒体。
【請求項４】
コンピュータを利用して、開発者が表示手段の画面上に表示されたソースコードの内容を追加・変更しながらソフトウェアの開発を行うためのソフトウェア開発支援プログラムにおいて、
設計情報に基づいて、一又は複数の挿入マーカがそれぞれ所定の箇所に挿入されたソースコードを生成すると共に、前記設計情報に基づいて、前記ソースコードを生成するときに、そのソースコードがどの設計情報に基づいて生成されたのかを識別する情報を付するソースコード生成機能と、
前記ソースコードのうち前記各挿入マーカで指定される範囲に含まれるコードを挿入コードとして抽出し、その抽出した前記各挿入コードを、当該挿入コードを含むソースコード及び当該挿入コードの挿入マーカを識別するための識別情報と関連付けて挿入コード記憶手段に記憶させる挿入コード抽出機能と、

前記ソースコード生成機能により、以前生成した前記ソースコードについて再度前記ソースコードが生成されたときに、前記挿入コード記憶手段に記憶されている前記挿入コードのうち、前記再度生成したソースコード及び当該ソースコードに存在する挿入マーカを識別するための識別情報と同一の識別情報に関連付けられた挿入コードを特定し、前記特定した挿入コードを、当該再度生成したソースコード中に存在する当該挿入マーカで指定される範囲に結合すると共に、前記ソースコード生成機能により、以前生成した前記ソースコードについて再度前記ソースコードが生成されたときに、前記挿入コード記憶手段に記憶されている前記挿入コードの中に、当該挿入コードについての前記識別情報によって特定される前記ソースコードを識別する前記情報と当該ソースコードを識別する前記情報とが一致するが、当該識別情報によって特定される前記挿入マーカが当該ソースコードに存在しないものが含まれていれば、当該ソースコードに結合されない前記挿入コードが存在することを検出するコード結合機能と、

を前記コンピュータに実現させることを特徴とするソフトウェア開発支援プログラム。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、開発者（プログラマ）によるソフトウェアの開発を支援するソフトウェア開発支援システムに関する。

【背景技術】

【０００２】

現在、ソフトウェアの開発においては、ソースコードジェネレータの機能を備えたソフトウェア開発支援ツールを用いることが多くなっている。このソースコードジェネレータは、入力された設計情報（仕様情報）に基づいて、一定の規則にしたがいソースコード（プログラム）を自動的に生成するツールである。

【０００３】

そのようなソフトウェア開発支援システムを用いたソフトウェアの開発では、まず、ソースコードジェネレータで生成されたソースコードに対して動作確認等を行う。次に、その結果等にしたがって、機能の追加や処理の変更等、ソースコードそのものの変更や設計情報の変更が行われる。設計情報の変更が行われた場合には、再度、設計情報に基づいてソースコードジェネレータでソースコードを生成する。このような作業は一回に限らず、ソフトウェアが完成するまでに何度となく繰り返される。

【０００４】

ところで、開発者が一回目に生成されたソースコードに変更を加えても、再度ソースコードジェネレータでソースコードの自動生成を行うと、その変更を加えた部分が上書き削除されてしまうケースが多い。例えば、一回目に生成されたソースコードにコメント文を入れたとする。そして、再度、同じ設計情報に基づいてソースコードを生成すると、その二回目に生成されたソースコードには、そのコメント文が含まれていない。この場合には、設計情報そのものには何ら変更が加えられていないので、ソースコードジェネレータは一回目であろうが二回目であろうが同じソースコードを生成するのである。このように、従来のソースコードジェネレータを用いたソフトウェア開発では、ソースコードジェネレータでソースコードを生成する度に開発者がそのソースコードに同じ変更を加えなければならないことが多いので、ソフトウェア開発を効率よく行うことができないという問題があった。

【０００５】

かかる問題を解決するために、ソースコードジェネレータで自動生成された後に開発者によって変更されたソースコードを、実装情報抽出装置で設計図に変換して実装情報としてデータベースに格納しておき、その実装情報を設計情報に反映するというソフトウェア開発支援システムが提案されている（例えば、特許文献１参照。）。また、変更前のソースコードＡと変更後のソースコードＢとの桁数や言語仕様に起因する特有の記述の差分を抽出し、この差分の情報を基にして変更後の設計情報を作成するというシステムも提案さ

れている（例えば、特許文献２参照。）。これらのシステムでは、開発者がソースコードに変更を加えた内容を設計情報に反映させ、その反映させた新たな設計情報をソースコードジェネレータに入力することにより、上記の問題を解決している。

【０００６】

【特許文献１】特開平７－１２９３８２号公報

【特許文献２】特開２０００－６６８８９号公報

【発明の開示】

【発明が解決しようとする課題】

【０００７】

ところで、このような特許文献１や特許文献２に示す先行技術では、開発者がソースコードに加えた変更の内容に基づいて、設計情報そのものを変更している。このため、例えば、単なるコメント文の追加等、設計情報に定義する程ではない変更や設計情報に該当する部分がない変更が行われた場合には、これらの先行技術では対応することができず、開発者による変更内容がソースコードに反映されない。

【０００８】

また、ソースコードジェネレータには定型的なソースコードを生成させ、ビジネスロジック等の非定型なソースコードについては開発者が自ら実装する、というように、ソースコードジェネレータと開発者とで分担してソフトウェアの開発を行う場合がある。このように、ソースコードジェネレータと開発者とで作業を分担することが前提となっている場合には、そもそも、開発者が実装した部分を設計情報に反映させる必要はないので、上述した先行技術の適用自体が考えられていない。

【０００９】

また、開発者によるソースコードの変更部分を正確に抽出することが技術的に困難な場合もある。例えば、図１１の左図に示すように、ソースコードジェネレータによってａａ、ｂｂ、ｃｃという内容のソースコードが生成され、その後、図１１の右図に示すように、開発者がそのソースコードにａａ、ｂｂを追加したとする。この場合、特許文献２に示す先行技術のように二つのソースコードを比較して差分を取る手法を用いたのでは、開発者によって追記されたのが上のａａ、ｂｂであるのか、下のａａ、ｂｂであるのかを判別できず、或いは誤って判別されてしまうことがある。

【００１０】

このように、従来は、上述の先行技術を用いても、開発者がソースコードに追加した内容を、再度生成したソースコードに反映させることができない場合があった。すなわち、開発者がソースコードに加えた変更の内容に基づいて設計情報を変更するという先行技術の適用範囲は狭く、かかる先行技術を用いてもソフトウェア開発を効率よく行うことができない場合があった。また、開発者によるソースコードの変更とは別に、設計情報にも追加が行われた場合、両方の変更を反映したソースコードを生成することができるソフトウェア開発支援システムの実現が望まれている。

【００１１】

本発明は上記事情に基づいてなされたものであり、設計情報を変更することなく、開発者がソースコードに追加した内容を、再度生成したソースコードに反映させることができるソフトウェア開発支援システムを提供することを目的とするものである。

【課題を解決するための手段】

【００１２】

上記の目的を達成するための本発明は、開発者が表示手段の画面上に表示されたソースコードの内容を追加・変更しながらソフトウェアの開発を行うために使用されるソフトウェア開発支援システムにおいて、設計情報に基づいて、一又は複数の挿入マーカがそれぞれ所定の箇所に挿入されたソースコードを生成するソースコード生成手段と、前記ソースコードのうち前記各挿入マーカで指定される範囲に含まれるコードを挿入コードとして抽出すると共に、その抽出した前記各挿入コードついて当該挿入コードが含まれる前記ソースコード及び前記挿入マーカを識別するための識別情報を取得する挿入コード抽出手段と

、前記挿入コード抽出手段によって抽出された前記各挿入コードを、当該挿入コードについての前記識別情報と関連付けて記憶する挿入コード記憶手段と、前記ソースコード生成手段が前記ソースコードを生成したときに、前記挿入コード記憶手段に記憶されている前記挿入コードのうち、当該挿入コードについての前記識別情報によって特定される前記ソースコードが当該ソースコードと一致し且つ当該識別情報によって特定される前記挿入マーカが当該ソースコードに存在するものについて、当該挿入コードを当該ソースコードのうち当該挿入マーカで指定される範囲に結合する処理を行うコード結合手段と、を具備することを特徴とするものである。

【００１３】

本発明のソフトウェア開発支援システムにおいて、前記コード結合手段は、前記ソースコード生成手段が前記ソースコードを生成したときに、前記挿入コード記憶手段に記憶されている前記挿入コードの中に、当該挿入コードについての前記識別情報によって特定される前記ソースコードが当該ソースコードと一致するが、当該識別情報によって特定される前記挿入マーカが当該ソースコードに存在しないものが含まれていれば、当該ソースコードに結合されない前記挿入コードが存在することを検出することが望ましい。

【００１４】

本発明のソフトウェア開発支援システムにおいて、前記ソースコードを編集すると共に、前記ソースコードを編集する際に前記挿入マーカで指定される範囲以外の範囲におけるコードの入力を制限するコード編集手段を具備することが望ましい。

【００１５】

上記の目的を達成するための本発明に係るコンピュータ読み取り可能な記録媒体は、上記のソフトウェア開発支援システムの機能をコンピュータに実現させるためのプログラムを記録したものである。

【００１６】

上記の目的を達成するための本発明に係るプログラムは、上記のソフトウェア開発支援システムの機能をコンピュータに実現させるためのものである。

【発明の効果】

【００１７】

本発明では、ソースコード生成手段が、設計情報に基づいて、一又は複数の挿入マーカがそれぞれ所定の箇所に挿入されたソースコードを生成し、挿入コード抽出手段が、ソースコードのうち各挿入マーカで指定される範囲に含まれるコードを挿入コードとして抽出した後、その抽出した各挿入コードを、当該挿入コードについての識別情報と関連付けて挿入コード記憶手段に記憶しておく。そして、コード結合手段は、ソースコード生成手段がソースコードを生成したときに、挿入コード記憶手段に記憶されている挿入コードのうち、当該挿入コードについての識別情報によって特定されるソースコードが当該ソースコードと一致し且つその識別情報によって特定される挿入マーカが当該ソースコードに存在するものについて、当該挿入コードを当該ソースコードのうち当該挿入マーカで指定される範囲に結合する処理を行う。これにより、設計情報を変更することなく、開発者がソースコードに追加した内容を、再度生成したソースコードに反映させることができる。したがって、開発者はソースコードが生成される度に、同じコードの入力を行う必要がなくなるので、開発効率の向上を図ることができる。

【００１８】

また、本発明では、コード結合手段は、ソースコード生成手段がソースコードを生成したときに、挿入コード記憶手段に記憶されている挿入コードの中に、当該挿入コードについての識別情報によって特定されるソースコードが当該ソースコードと一致するが、その識別情報によって特定される挿入マーカが当該ソースコードに存在しないものが含まれていれば、当該ソースコードに結合されない挿入コードが存在することを検出する。このため、かかる検出結果を開発者に知らせることにより、開発者は、当該挿入コードの存在を認識し、当該挿入コードが本当に必要かどうかを再検討することができるので、開発効率がさらに向上する。

【００１９】
更に、本発明では、ソースコードを編集すると共に、ソースコードを編集する際に挿入マーカで指定される範囲以外の範囲におけるコードの入力を制限するコード編集手段を備えることにより、ソースコードの中で開発者がコードの追加・変更を行うべき範囲を明確にすることができる。
【発明を実施するための最良の形態】
【００２０】
以下に、図面を参照して、本願に係る発明を実施するための最良の形態について説明する。図１は本発明の一実施形態であるソフトウェア開発支援システムの概略ブロック図である。
【００２１】
本実施形態のソフトウェア開発支援システムは、開発者（プログラマ）がソースコード（プログラム）の内容を追加・変更しながらソフトウェアの開発を行うために使用されるものである。具体的に、開発者は、本実施形態のソフトウェア開発支援システムを用いて、設計情報に基づいてソースコードを自動生成し、そのソースコードに対してコードの追加・変更を行うという作業を繰り返すことにより、ソフトウェアを完成させる。
【００２２】
かかるソフトウェア開発支援システムは、図１に示すように、入力装置１０と、表示装置２０と、記憶部３０と、制御部４０とを備える。
【００２３】
入力装置１０は、例えば、設計情報やソースコードを入力するためのものである。入力装置１０としては、キーボードやマウス等が用いられる。表示装置２０は、設計情報の入力画面を表示したり、ソースコードの内容を表示したりするものである。
【００２４】
記憶部３０には、設計情報やソースコード等の各データや、各種のアプリケーションプログラムが格納されている。また、記憶部３０には、ソースコード保存部３１と、挿入コード保存部３２とが含まれる。ソースコード保存部３１は、制御部４０によって生成されたソースコードを一時的に保存する作業用のメモリである。挿入コード保存部（挿入コード記憶手段）３２は、後述する挿入コードを格納するものである。
【００２５】
制御部４０は、本システムの各部を統括して制御するものである。この制御部４０は、図１に示すように、ソースコードジェネレータ（ソースコード生成手段）４１と、コード編集部４２と、挿入コード抽出部４３と、コード結合部４４とを有する。ソースコードジェネレータ４１は、設計情報に基づいてソースコードを自動生成するものである。
【００２６】
制御部４０の各部の説明を行う前に、設計情報について詳しく説明する。設計情報とは、どのようなソースコードを作るのかという内容を、人間に分かりやすい形で書いたものである。この設計情報自体はプログラムではない。開発者は、表示装置２０に表示された設計情報の入力画面上で、設計情報を入力したり、その内容を変更したりする。
【００２７】
ところで、開発者が設計情報を入力する際に、あらゆる内容を事細かに入力することにすると、その入力作業にとても手間がかかる。このため、本実施形態では、設計情報には、定型的に表しやすい内容だけを含めることにしている。例えば、データの種類、データの定義、データ間の包含関係等は、定型的に表しやすい内容である。これに対し、処理の内容等は、定型的に表しにくい内容である。一般に、ある事柄が定型的に表しやすい内容であるかどうかは、開発者が開発しようとしているソフトウェアの内容に依存する。また、定型的に表しやすい内容については、ソースコードへの変換を規則的に且つ容易に行うことができる。これに対し、処理の内容等の定型的に表しにくい内容については、ソースコードに自動変換しても、大抵の場合、所望のプログラムが得られない。したがって、本実施形態のソフトウェア開発支援システムでは、開発者は、定型的に表しやすい内容を設

計情報として入力し、定型的に表しにくい内容をソースコードの中に直接入力することになる。すなわち、ソースコードジェネレータ４１と開発者とで分担してソフトウェアの開発を行うことにより、ソフトウェアの開発効率の向上を図っている。

【００２８】

設計情報の入力形式にはさまざまなものがある。図２（ａ）は表形式で入力された設計情報の一例を示す図、図２（ｂ）は同図（ａ）の設計情報においてデータの追加が行われた場合の例を示す図である。図３（ａ）はテキスト形式で入力された設計情報の一例を示す図、図３（ｂ）は同図（ａ）の設計情報においてデータの追加が行われた場合の例を示す図である。

【００２９】

ここでは、例えば名簿管理システムを実現するためのソフトウェアを作成する場合を考えることにする。この場合、表形式の設計情報の入力画面には、図２（ａ）に示すように、データ型の定義（data definition）を行うためのテーブル１１０と、データの属性（attributes list）を定義するためのテーブル１２０とが含まれている。データ型の定義を行うためのテーブル１１０としては、“data type”、“description”、“package”の各項目名が記載された書式テンプレートが予め用意されている。名簿管理システムでは「人」のデータが扱われるので、例えば、“data type”の項目には“Person”が、“description”の項目には“人を表すデータ”が、“package”の項目には“jp.co.ns_sol.sysrdc”が、それぞれ開発者によって入力されている。また、データの属性を定義するためのテーブル１２０としては、“symbol”、“type”、“access limitation”の各項目名が記載された書式テンプレートが予め用意されている。例えば、“symbol”の項目には、“name（名前）”、“address（住所）”、“dataOfBirth（誕生日）”、“age（年齢）”、“phoneNo（電話番号）”が、開発者によって入力されている。これにより、「人」のデータには、これら五つの変数が含まれることになる。そして、これらの変数は、ソースコードジェネレータ４１が生成するソースコードの中ではその入力したシンボルで表現されることになる。また、“type”の項目には、各変数の型が入力される。ここでは、“name”、“address”、“phoneNo”の各シンボルに対しては“String”、すなわち文字列型を指定し、“dataOfBirth”のシンボルに対しては“Data”を指定し、そして、“age”のシンボルに対しては“Integer”、すなわち整数型を指定している。また、“access limitation”の項目には、“age”のシンボルに対してのみ、“readonly”が指定されている。この指定が行われると、開発者はソースコードにおいて“age”のシンボルの内容を変更することができない。これは、“age”の内容は“dataOfBirth”から計算できるからである。この計算のロジックは、開発者によってソースコード中に書かれることになる。

【００３０】

また、上記と同じ名簿管理システムを実現するためのソフトウェアを作成する場合にテキスト形式で入力された設計情報が、図３（ａ）である。図３（ａ）の内容は、入力形式の違いを除けば、図２（ａ）におけるものと全く同じである。開発者や開発環境等によっては、設計情報をテキスト形式で表示した方が便利な場合がある。

【００３１】

ところで、設計情報の内容は、ソフトウェアの開発中に変更されることがある。例えば、顧客から、「人」のデータにメールアドレスを追加してほしいという要求が出されることがある。このような場合、例えば、開発者は、図２（ｂ）又は図３（ｂ）に示すように、「人」のデータに文字列型の“mailAddress”という変数を追加する。通常、ソースコード内においては、変数が多くの箇所に現れるので、開発者が変数の追加をソースコード内で直接行うことにすると、とても手間がかかる。これに対し、設計情報内で変数の追加を行い、その追加がなされた設計情報に基づいて再度、ソースコードを生成すれば、当該変数が追加されたソースコードを容易に得ることができる。

【００３２】

次に、ソースコードジェネレータ４１について説明する。ソースコードジェネレータ４１は、設計情報に基づいてソースコードを自動生成する。図４から図９までの各図は、ソ

ースコードジェネレータ４１によって作成されたソースコードの例を示す図である。具体的に、図２（ａ）又は図３（ａ）に示す設計情報に基づいてソースコードジェネレータ４１によって作成されたソースコードが、図４、図５及び図６に示す一連のソースコードである。また、図２（ｂ）又は図３（ｂ）に示す設計情報に基づいてソースコードジェネレータ４１によって作成されたソースコードが、図７、図８及び図９に示す一連のソースコードである。すなわち、ここでは、図面作成上の都合から、ソースコードジェネレータ４１によって作成されたソースコードを三つに分割し、その分割された各ソースコードをそれぞれ、図４、図５、図６（又は、図７、図８、図９）に示している。したがって、図面の番号の順に、その分割された三つのソースコードを繋ぎ合わせたものが、全体のソースコードである。

【００３３】

ソースコードジェネレータ４１によって生成されたソースコードには所定のファイル名が付けられる。例えば、図４、図５及び図６に示す一連のソースコードには、図４の上から二行目に示すように、“Person.java”というファイル名が付けられている。すなわち、ソースコードジェネレータ４１は、図２又は図３に示す設計情報の“data type”の項目に入力された“Person”という情報を利用して、ソースコードのファイル名を生成する。このため、ソースコードのファイル名が分かれば、当該ソースコードがどの設計情報に基づいて生成されたのかを容易に知ることができる。

【００３４】

また、一つの設計情報からは一つのソースコードだけでなく、複数のソースコードが生成されることがある。例えば、画面表示用のソースコード、計算用のソースコード、データベース用のソースコードのように、三つのソースコードが生成される場合がある。かかる三つのソースコードは完全に独立したものではなく、それらのソースコードが協調して動作することにより、画面上で入力されたデータを用いて所定の計算を行い、その結果をデータベースに格納するというように、システムが全体として機能することになる。このように複数のソースコードが生成される場合には、各ソースコードには、例えば、共通の名称とそれらを識別する符号や文字列等とを組み合わせて得られるファイル名が付けられる。具体例として、図２及び図３に示す設計情報に基づいて複数のソースコードが生成される場合には、それらのソースコードに対してそれぞれ、“Person_gui.java”、“Person_calc.java”、“Person_db.java”、・・・のようなファイル名が付けられる。

【００３５】

尚、上述の例では、“Person”という情報を利用して、生成されたソースコード毎のファイル名を生成することにより、ソースコードを識別する場合について説明したが、そのような方法以外に、生成されたソースコードを識別する方法としては、格納場所を変えるという方法を用いることができる。すなわち、ソースコードの格納場所名とファイル名とを合わせたもの（パス名などと呼ばれる。）によって、ソースコードを識別することができる。このため、例えば、上述の三つのソースコードが生成される場合には、それらのソースコードに対してはいずれも、“Person.java”というファイル名を付け、それらのソースコードをそれぞれ、“gui/Person.java”、“calc/Person.java”、“db/Person.java”のように、別々のディレクトリ（フォルダ）に格納しておけばよい。

【００３６】

また、本実施形態では、ソースコードジェネレータ４１に、一又は複数の挿入マーカをそれぞれ、ソースコードの所定の箇所に挿入する機能を付加している。挿入マーカは、開発者によってコードの追加・変更が行われる可能性のある箇所（範囲）を明示する表示である。例えば、開発者がプログラムを入力しなければならない箇所や、プログラム作成の便宜上、コメントを挿入してもよい箇所等に、挿入マーカが挿入されている。一般に、ソースコードジェネレータ４１は、ソースコードを生成する際に、開発者によってコードの追加・変更が行われる可能性があるすべての箇所に、挿入マーカを挿入しておくことが望ましい。

【００３７】

開発者は、入力装置１０を用いて、挿入マーカで指定される範囲、及び当該範囲以外の範囲にコードを入力することができる。特に、ソースコードのうち挿入マーカで指定される範囲に含まれるコードのことを、「挿入コード」と称する。かかる挿入コードは、挿入コード保存部３２に保存される。後に詳述するが、再度、ソースコードが生成されたときには、当該挿入コードの内容がそのソースコードに反映される。尚、挿入マーカで指定される範囲以外の範囲にコードを入力した場合には、その入力したコードの内容が記憶部３０に保存されたり、その後に生成されるソースコードに反映されたりすることはない。また、挿入マーカは、後述するように、挿入コード抽出部４３がソースコードから挿入コードを抽出する際に、その抽出する範囲を特定するという役割をも有している。

【００３８】

ソースコードジェネレータ４１がソースコードを生成したとき、通常、そのソースコードに含まれる各挿入マーカで指定される範囲には何もコードが含まれていない。開発者が必要に応じて挿入マーカで指定される範囲にコードを入力することになる。しかし、ソースコードジェネレータ４１は、ソースコードを生成したときに、挿入マーカで指定される範囲に、デフォルトの挿入コードを生成して入れておくこともできる。デフォルトの挿入コードは、挿入マーカで指定される範囲に含まれているので、開発者が変更等をしなければ、そのままの内容で挿入コード保存部３２に記憶される。一方、開発者がデフォルトの挿入コードを変更等すると、その変更等された内容で挿入コードが抽出され、挿入コード保存部３２に格納される。

【００３９】

デフォルトの挿入コードは次のような場合に生成される。例えば、ソースコードの中のある部分に含めるべきプログラムとして、大抵、特定のコードが使用されるが、稀なケースでは、当該特定コード以外のコードを使用する必要があるという場合がある。このような場合には、ソースコードのうち当該特定コードを含める箇所に挿入マーカを付加すると共に、その挿入マーカで指定される範囲に当該特定コードをデフォルトの挿入コードとして含めておく。これにより、開発者は、通常、当該特定コードをそのまま使用し、変更する必要があると判断した場合だけ、当該特定コードを別のコードに書き換えればよいので、ソフトウェアの開発を効率よく行うことができる。また、あるプログラムが書きやすい部分と書きにくい部分とを含んでいる場合には、そのプログラムのうち書きやすい部分だけをデフォルトの挿入コードとすることができる。

【００４０】

図４、図５及び図６に示す一連のソースコードには、六つの挿入マーカM_1，M_2，M_3，M_4，M_5，M_6が含まれている。すなわち、そのソースコードにおいて、“// NSStella mixin point (*****) BEGIN”、“// NSStella mixin point (*****) END”と記載された一組の文字列が、挿入マーカである。ここで、“*****”には、その挿入マーカを識別するためのラベルが記載される。すなわち、各挿入マーカはラベルによって識別される。ラベルを除けば、どの挿入コードも、その文字列の内容が同じである。また、挿入マーカで指定される範囲とは、“// NSStella mixin point (*****) BEGIN”と“// NSStella mixin point (*****) END”とで囲まれた範囲のことである。開発者は、“// NSStella mixin point (*****) BEGIN”と“// NSStella mixin point (*****) END”との間に、プログラムやコメント等を追加することができる。

【００４１】

具体的に、第一の挿入マーカM_1には、図４に示すように、“imports”というラベルが付されている。この第一の挿入マーカM_1は、デフォルトで生成された五つのImport文（当該マーカM_1の上に記載された五行）だけでは、Import文が足りない場合に、開発者が、Import文を追加することができるように設けられたものである。第二の挿入マーカM_2には、“comment”というラベルが付されている。この第二の挿入マーカM_2は、開発者がソースコードの中にさらにコメントを挿入することができるように設けられたものである。第三の挿入マーカM_3には、“fields”というラベルが付されている。この第三の挿入マーカM_3は、開発者が、フィールドを追加することができるように設けられたもの

である。第四の挿入マーカＭ$_4$には、図５に示すように、“age_readonly”というラベルが付されている。この第四の挿入マーカＭ$_4$は、設計情報において“readonly”の指定がある変数“age”については、ソースコードジェネレータ４１がその算出のプログラムを自動生成しないので、そのプログラムを開発者に入力してもらうために設けられたものである。第五の挿入マーカＭ$_5$には、図６に示すように、“methods”というラベルが付されている。この第五の挿入マーカＭ$_5$の間には、特定のメソッドがデフォルトの挿入コードとして含まれている。開発者は、デフォルトの挿入コードを変更したほうがよいと判断すると、例えば、当該デフォルトの挿入コードを削除し、その代わりに、自己の作成したメソッドを当該挿入マーカＭ$_5$の間に入力することになる。そして、第六の挿入マーカＭ$_6$には、“file_tail”というラベルが付されている。この第六の挿入マーカＭ$_6$はソースコードの最後に設けられている。この挿入マーカＭ$_6$を設けたのは、開発者が他に何かを追加したい場合もあるだろうということを考慮したものである。

【００４２】

また、図７、図８及び図９に示す一連のソースコードは、図４、図５及び図６に示す一連のソースコードとほとんど同じである。特に、それらの一連のソースコードにおいて挿入マーカの数やラベルは全く同じである。図７、図８及び図９に示す一連のソースコードと図４、図５及び図６に示す一連のソースコードとの違いは、図７、図８及び図９に示す一連のソースコードには、設計情報において「人」のデータにメールアドレスを追加したことに起因して、メールアドレスについてのprivate文と、メールアドレスの生成プログラムとが追加されている点だけである。

【００４３】

尚、上述の例では、挿入マーカ“// NSStella mixin point (*****) BEGIN”、“// NSStella mixin point (*****) END”に含まれるラベル“*****”としては、開発者にとって分かりやすいように、“imports”や“fields”など、意味ある文字列を用いている。このようなラベルを各挿入マーカに対して付与する作業をコンピュータに自動で行わせるには、記憶部３０に所定のラベル付与ルールを格納しておき、ソースコードジェネレータ４１がそのラベル付与ルールにしたがってラベルを付与すればよい。また、ラベルとしては、単純にシリアル番号などを用いてもよい。

【００４４】

ソースコードは、ソースコードジェネレータ４１又は後述するコード結合部４４によって作成されると、ソースコード保存部３１に一時記憶される。また、開発者によってソースコードの内容が追加・変更されると、その追加等された後のソースコードがソースコード保存部３１に一時記憶される。このとき、以前に作成されたソースコードがソースコード保存部３１に記憶されていれば、その以前のソースコードは削除される。プログラムの開発においてはソースコードが何度も作成・変更されるが、作業用のメモリであるソースコード保存部３１には、常に一つのソースコードしか存在しない。

【００４５】

尚、ソースコードは、いつでも新たなファイル名を付して記憶部３０に保存しておくことができる。これにより、開発者は、過去のある時点で生成したソースコードを読み出して、その内容を参照することができる。

【００４６】

コード編集部４２は、ソースコードを編集するものである。具体的に、コード編集部４２としては、通常のエディタを使用することができる。開発者は、かかるコード編集部４２の機能を利用して、表示装置２０に表示された設計情報の入力画面上で設計情報を入力したり、表示装置２０に表示されたソースコードの内容を追加・変更したりする。

【００４７】

挿入コード抽出部４３は、ソースコードの中から挿入マーカを見つけ出し、その見つけ出した挿入マーカの間に含まれる挿入コードを抽出するものである。また、挿入コード抽出部４３は、その抽出した各挿入コードについて、当該挿入コードが含まれるソースコード及び挿入マーカを識別するための識別情報を取得する。ここでは、識別情報として、当

該ソースコードのファイル名と当該挿入マーカのラベルとで構成される情報を用いることにする。挿入コード抽出部４３によって抽出された各挿入コードは、当該挿入コードについての識別情報と関連付けて挿入コード保存部３２に記憶される。例えば、かかる関連付けの方法としては、挿入コードについての識別情報をその挿入コードのファイル名とする方法を用いることができる。例えば、ソースコードのファイル名が“Person.java”で、挿入マーカのラベルが“imports”である場合には、当該挿入コードのファイル名は、“Person.java_imports”とされる。したがって、挿入コード保存部３２に記憶された各挿入コードについては、そのファイル名が分かれば、当該挿入コードがどのソースコードの中の、どの挿入マーカの間に含まれていたものであるかを容易に知ることができる。尚、一つのソースコードについては、それに含まれるすべての挿入コードをまとめて一つのファイル（例えば、そのファイル名を“Person.java.mixin”とする。）に保存しておくことも可能である。各挿入コードが識別情報と関連付けられていれば、当該ファイルの中に含まれる識別情報によって挿入コードを識別することができるからである。

【００４８】

挿入コード抽出部４３が抽出するのは、挿入マーカの間に含まれるコード（挿入コード）だけであり、それ以外の範囲に挿入されたコードは抽出されない。したがって、開発者によって挿入マーカで指定される範囲以外の範囲に入力されたコードは、ソースコードジェネレータ４１が再度、同じファイル名を有する設計情報に基づいてソースコードを生成したときに、そのソースコードに反映されることはない。ここで、「同じファイル名を有する設計情報」には、ファイル名が同じである限り、内容が全く同一の設計情報だけでなく、内容の追加・変更がなされた設計情報も含まれる。

【００４９】

また、挿入コード抽出部４３は、挿入コードの抽出処理を行う前に、ソースコードのタイムスタンプに基づいて当該ソースコードが最新のものであるかどうかを判断してもよい。そして、当該ソースコードが最新ものであると判断したときのみ、挿入コードの抽出処理を行う。例えば、開発者は、以前作成した古いソースコードを記憶部３０から読み出して、表示装置２０の画面上に表示することがある。このような場合には、挿入コード抽出部４３は、そのソースコードのタイムスタンプに基づいて当該ソースコードが最新のものでないと判断するので、当該ソースコードから挿入コードを抽出する処理を行わない。

【００５０】

コード結合部４４は、ソースコードジェネレータ４１がソースコードを生成したときに、挿入コード保存部３２に記憶されている挿入コードのうち、当該挿入コードのファイル名（識別情報）によって特定されるソースコードが当該ソースコードと一致し且つそのファイル名によって特定される挿入マーカが当該ソースコードに存在するものについて、当該挿入コードを当該ソースコードのうち当該挿入マーカで指定される範囲に結合する処理を行い、その処理後のソースコードを表示装置２０に出力するものである。例えば、設計情報に追加がなされ、その追加後の設計情報に基づいてソースコードジェネレータ４１がソースコードを生成すると、その生成したソースコードには、開発者が追加前の設計情報に基づいて生成されたソースコードに入力した内容が反映されていない。コード結合部４４は、開発者が以前に生成されたソースコードに入力した内容のうち、挿入マーカの間に入力した挿入コードを、今回生成されたソースコードに反映させるという役割を果たす。尚、ソースコードジェネレータ４１が新たな設計情報に基づいて最初にソースコードを生成したときには、当然、そのソースコードと関連のある挿入コードは挿入コード保存部３２に記憶されていない。このため、この場合には、コード結合部４４は結合処理を行わない。

【００５１】

ところで、設計情報に追加がなされ、その追加後の設計情報に基づいて生成されたソースコードでは、追加前の設計情報に基づいて生成されたソースコードに比べて、いくつかの挿入マーカが新たに追加され、挿入マーカの数が増えていることがある。このような場合であっても、ソースコードジェネレータ４１はそれらの新たな挿入マーカに対しては異

なるラベルを付与する。したがって、コード結合部４４は、挿入コードの結合処理を行う際に、挿入コードをソースコード内で間違った箇所に結合することはない。

【００５２】

また、設計情報の一部が削除され、その削除後の設計情報に基づいて生成されたソースコードでは、削除前の設計情報に基づいて生成されたソースコードに比べて、挿入マーカの数が減ってしまうことがある。この場合、いくつかの挿入コードについては、そのソースコードに結合する箇所がなくなってしまうこともありうるが、ソースコードジェネレータ４１は当該挿入コードをそのソースコードに結合しないだけであり、このことは何ら問題とならない。このような挿入コードは、今回作成したソースコードにおいては必要がなくなったものであると考えられるが、場合によっては、他の箇所に入力すべきものであるかもしれない。このため、本実施形態では、コード結合部４４は、ソースコードジェネレータ４１がソースコードを生成したときに、挿入コード保存部３２に記憶されている挿入コードの中に、当該挿入コードのファイル名によって特定されるソースコードが当該ソースコードと一致するが、そのファイル名によって特定される挿入マーカが当該ソースコードに存在しないものが含まれていれば、当該ソースコードに結合されない挿入コードが存在する旨を示す警告画面を表示装置２０に表示させるようにしている。これにより、開発者は、そのような挿入コードの存在を認識し、当該挿入コードが本当に必要かどうかを再検討することができるので、開発効率が一層向上する。

【００５３】

ソースコードジェネレータ４１、挿入コード抽出部４３、コード結合部４４は、互いに連携し合って処理を行う。各部は所定のタイミングで他にコマンド等を出すことにより連携している。具体的には、ソースコードジェネレータ４１は設計情報が送られてきたときに、挿入コード抽出部４３に所定のコマンドを出力する。挿入コード抽出部４３はそのコマンドを受け取ると、現在のソースコードから挿入コードを抽出し、挿入コード保存部３２に記憶する。その後、挿入コード抽出部４３は、所定のコマンドをソースコードジェネレータ４１に出力する。ソースコードジェネレータ４１はそのコマンドを受け取ると、当該設計情報に基づいてソースコードを生成する。そして、コード結合部４４は、ソースコードジェネレータによってソースコードが生成されると、そのソースコードと挿入コードとの結合処理を行う。

【００５４】

次に、本実施形態のソフトウェア開発支援システムの処理手順について説明する。図１０は本実施形態のソフトウェア開発支援システムの処理手順を説明するための図である。

【００５５】

まず、ソフトウェアの開発者は、設計情報を所定の入力画面上で入力する。具体的に、その入力画面は、所定のエディタを起動したときのエディタ画面内に表示される。入力が完了した後、開発者は、その入力画面上に設けられたソースコード生成ボタンを押すと、その入力した設計情報がソースコードジェネレータ４１に送られる（Ｓ１）。

【００５６】

次に、ソースコードジェネレータ４１は、その設計情報に基づいてソースコードを生成する（Ｓ２）。具体的には、ソースコードジェネレータ４１は設計情報が送られてきたときに、挿入コード抽出部４３に所定のコマンドを出力する。挿入コード抽出部４３は、そのコマンドを受け取ると、挿入コードの抽出処理を行うことになるが、このとき、ソースコード保存部３１にはソースコードが記憶されていないので、挿入コードの抽出処理を行わずに、所定のコマンドをソースコードジェネレータ４１に出力する。そして、ソースコードジェネレータ４１はそのコマンドを受け取ると、当該設計情報に基づいてソースコードを生成し、ソースコード保存部３１に一時記憶する。ここで、このソースコードには一又は複数の挿入マーカが含まれている。こうしてソースコードが生成されると、コード結合部４４は、そのソースコードと挿入コードとの結合処理を行うことになるが、この時点では、挿入コード保存部３２に挿入コードが格納されていないので、コード結合部４４は、そのソースコードと挿入コードとの結合処理を行わずに、ソースコードをそのまま表示

装置２０に出力する。図１０では、こうして生成されたソースコードを「第ｎのソースコード」と称している。

【００５７】

本実施形態では、設計情報に定型的に表しやすい内容だけを含めているので、その設計情報に基づいて生成された第ｎのソースコードは必ずしも完全なものではない。このため、開発者は、エディタ（コード編集部４２）の機能を利用して、第ｎのソースコードの中にプログラムやコメント等を追加したり、その内容を変更したりする（Ｓ３）。このとき、その追加等する内容は、通常、挿入マーカで指定された箇所に入力される。こうして、開発者による追加等が行われたソースコードが得られる。図１０では、こうして得られたソースコードを「第ｎ＋１のソースコード」と称している。このとき、ソースコード保存部３１には、第ｎのソースコードに上書きされた第ｎ＋１のソースコードが一時記憶されている。

【００５８】

その後、開発者は、第ｎ＋１のソースコードをコンパイルしてテストを実行する（Ｓ４）。その結果、プログラムがうまく動作しなかった場合には、開発者は、その原因を追求する。例えば、データの数が少ないことが原因であると判断すると、設計情報にデータを追加することを決定する。また、プログラムがうまく動作した場合であっても、顧客からの要求によりデータを追加する必要が生ずることがある。このような場合、開発者は、エディタ画面内に設計情報を読み出し、設計情報に対してデータの追加を行う（Ｓ５）。設計情報に対する追加・変更の作業が完了した後、開発者は、その入力画面上でソースコード生成ボタンを押すと、その入力した設計情報がソースコードジェネレータ４１に送られる（Ｓ６）。

【００５９】

ソースコードジェネレータ４１は、設計情報が送られると、挿入コード抽出部４３に所定のコマンドを送る。挿入コード抽出部４３は、かかるコマンドを受け取ると、ソースコード保存部３１に記憶されている第ｎ＋１のソースコードの中から挿入マーカを検索し、各挿入マーカで指定された範囲に含まれる挿入コードを見つけ出す。そして、こうして見つけ出した挿入コードを抽出した後、その抽出した挿入コードに所定のファイル名を付して、その挿入コードを挿入コード保存部３２に記憶する（Ｓ７）。その後、挿入コード抽出部４３は、挿入コードの抽出処理が終了した旨のコマンドをソースコードジェネレータ４１に送る。

【００６０】

ソースコードジェネレータ４１は、挿入コード抽出部４３からコマンドを受けると、ステップＳ６で送られた設計情報に基づいてソースコードを生成する（Ｓ８）。図１０では、このソースコードを「第ｎ＋２のソースコード」と称している。また、このとき、ソースコード保存部３１には、第ｎ＋１のソースコードに上書きされた第ｎ＋２のソースコードが一時記憶されている。

【００６１】

次に、コード結合部４４は、ソースコードジェネレータ４１が第ｎ＋２のソースコードを生成すると、その第ｎ＋２のソースコードと挿入コードとの結合処理を行う（Ｓ９）。具体的に、コード結合部４４は、挿入コード保存部３２に記憶されている挿入コードのうち、当該挿入コードのファイル名によって特定されるソースコードが第ｎ＋２のソースコードと一致し且つそのファイル名によって特定される挿入マーカが第ｎ＋２のソースコードに存在するものを見つけ出す。そして、その見つけ出した挿入コードを当該第ｎ＋２のソースコードのうち当該挿入マーカで指定される範囲に結合する。こうして結合処理が行われたソースコードは、ソースコード保存部３１に一時記憶されると共に、表示装置２０に出力される。図１０では、こうして得られるソースコードを「第ｎ＋３のソースコード」と称している。ここで、第ｎ＋２のソースコードと第ｎ＋３のソースコードとの違いは次の通りである。すなわち、第ｎ＋２のソースコードと第ｎ＋３のソースコードはともに、ステップＳ５における設計情報の追加が考慮された内容となっている。しかし、第ｎ＋

２のソースコードには、第ｎ＋１ソースコードにおいて開発者によって入力された挿入コードの内容が含まれていないが、第ｎ＋３ソースコードには、その入力された挿入コードの内容が含まれている。このように、本実施形態のソフトウェア開発支援システムでは、開発者が以前生成されたソースコードに入力した内容のうち、挿入マーカの間に入力した挿入コードを、今回生成されたソースコードに反映させることができる。

【００６２】

その後、開発者は、そのソースコードをコンパイルしてテストを実行する（Ｓ１０）。その結果、プログラムがうまく動作しなかった場合等には、開発者は、エディタの機能を利用して、設計情報の内容やソースコードの内容を追加・変更する。そして、ステップＳ３からステップＳ１０までの処理を繰り返すことにより、所望のソースコードを完成させることになる。

【００６３】

上述したように、開発者は、本実施形態のソフトウェア開発支援システムを使用してソースコードを生成し、そのソースコードのテストを行いながら、ソフトウェアを完成させていく。特に、本実施形態のソフトウェア開発支援システムでは、あえて間違った内容のソースコードを用いてテストを行うという使い方をすることができる。すなわち、本実施形態のソフトウェア開発支援システムでは、開発者がソースコードの中でコードを追加したり、変更したりしても、その内容が設計情報に反映されることはない。このため、テストを行う際に、テスト用のコードやバグのあるコードをわざとソースコードに付加し、そのようなソースコードを用いてわざと間違った答えを出すようなテストを行うことができる。これらのテスト用コードやバグのあるコードは、当然、設計には関係のないものである。従来の技術では、ソースコードに追加・変更した内容が設計情報に自動的に反映されてしまうので、そのようなテストを行うことができなかった。これに対して、本実施形態では、かかるテストにも容易に対応することができる。テスト用のコードやバグのあるコードは、ソースコードのどの場所に入力してもよい。テスト用のコード等を挿入マーカで指定される範囲以外の範囲に入れた場合には、同じファイル名を有する設計情報に基づいて再度、ソースコードを生成すれば、テスト用のコード等が挿入されていない状態のソースコードが得られる。一方、テスト用のコード等を挿入マーカで指定される範囲に入れた場合には、同じファイル名を有する設計情報に基づいて再度、ソースコードを生成すれば、そのテスト用のコード等が挿入された情報のソースコードが得られる。この場合には、例えば、そのテスト用のコード等の内容を変更することにより、さらに条件等を一部変更したテストを行うことができる。

【００６４】

また、開発者は挿入マーカ自体をソースコードの中で適当な箇所に入力することができる。このときには、その挿入マーカのラベルを、他の挿入マーカのラベルと異なるものにする必要がある。開発者が自ら挿入マーカを入力した場合でも、挿入コード抽出部４３は、その挿入マーカを認識し、その挿入マーカの間に入力された挿入コードを抽出する。その後、同じファイル名を有する設計情報に基づいてソースコードが作成されたときには、当然、そのソースコードには当該挿入マーカは存在しないので、その抽出した挿入コードはそのソースコードと結合されない。しかし、コード結合部４４は、その結合されない挿入コードが存在する旨の警告画面を表示装置２０に表示させる。このため、例えば、開発者は、自ら挿入マーカを入力し、その挿入マーカの間にテスト用のコード等を入力しておけば、その後、同じファイル名を有する設計情報に基づいてソースコードが作成され、警告画面が表示されたときに、その挿入コードを所定の位置に挿入して、再度、同じテストを行うことが可能である。ここで、コード結合部４４は、開発者がその警告画面上で当該挿入コードを挿入する旨のボタンを押したときに、挿入位置の指示画面を表示装置２０に表示させることが望ましい。これにより、開発者がその指示画面上で挿入位置を指示すれば、当該挿入コードはコード結合部４４によって所望の位置に自動的に挿入される。

【００６５】

本実施形態のソフトウェア開発支援システムでは、ソースコードジェネレータが、設計

情報に基づいて、一又は複数の挿入マーカがそれぞれ所定の箇所に挿入されたソースコードを生成し、挿入コード抽出部が、ソースコードのうち各挿入マーカの間に含まれるコードを挿入コードとして抽出して、その抽出された各挿入コードに所定のファイル名を付して、各挿入コードを挿入コード保存部に記憶しておく。そして、コード結合部は、ソースコードジェネレータがソースコードを生成したときに、挿入コード保存部に記憶されている挿入コードのうち、当該挿入コードのファイル名によって特定されるソースコードが当該ソースコードと一致し且つそのファイル名によって特定される挿入マーカが当該ソースコードに存在するものについて、当該挿入コードを当該ソースコードのうち当該挿入マーカで指定される範囲に結合する処理を行い、その処理後のソースコードを表示装置に出力する。これにより、設計情報を変更することなく、開発者がソースコードに追加した内容を、再度生成したソースコードに反映させることができる。したがって、開発者はソースコードが生成される度に、同じコードの入力を行う必要がなくなるので、開発効率の向上を図ることができる。

【００６６】

尚、本発明は上記の実施形態に限定されるものではなく、その要旨の範囲内において種々の変形が可能である。

【００６７】

例えば、上記の実施形態では、挿入マーカを特定の文字列で表示する場合について説明したが、その挿入マーカの文字列を所定の色で表示するようにしてもよい。これは、コード編集部に、ソースコードを表示装置の画面上に表示する際に挿入マーカの文字列を検索し、その検索で得られた文字列を所定の色で表示するという機能を付加することにより実現することができる。このように色が付された挿入マーカを表示することにより、開発者は挿入マーカを容易に識別することができる。

【００６８】

また、上記の実施形態において、開発者は、ソースコードのどの箇所にもコードを入力することができる場合について説明したが、ソースコードのうち挿入マーカで指定される範囲以外の範囲にはコードを入力することができないようにしてもよい。これは、コード編集部に、ソースコードにおいて挿入コードの文字列を検索し、その検索で得られた挿入マーカで指定される範囲でだけコードの入力を許可し、その挿入マーカで指定される範囲以外の範囲ではコードの入力を制限する機能を付加することにより実現することができる。これにより、ソースコードの中で開発者がコードの追加・変更を行うべき範囲を明確にすることができるので、開発効率をより一層高めることができる。

【００６９】

更に、上記の実施形態では、開発者が表示装置に表示された画面を参照しながら一連の処理を行う場合について説明したが、本発明はこれに限定されるものではなく、例えば、コード結合部が挿入コードの結合処理を行った後に、その結合処理により得られたソースコードを表示装置に表示することなく、そのソースコードについてはそのままコンパイル等の処理を行うようにしてもよい。少なくとも表示手段への表示が必要になるのは、開発者が設計情報を入力するときと、ソースコードジェネレータが出力したソースコードに対して開発者がコード編集部の機能を利用して追加・変更等を行うとき（図１０における第ｎのソースコードから第ｎ＋１のソースコードを作成するとき）であり、その他の処理においては、目的に応じて任意のタイミングでデータや情報を表示装置に表示すればよい。

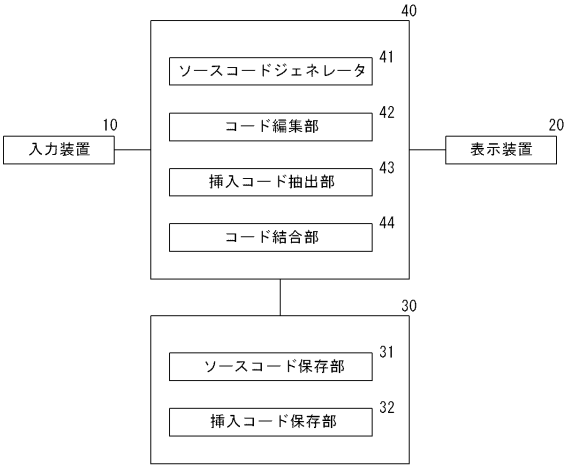
【００７０】

本発明の目的は、上述した各実施形態の装置の機能を実現するソフトウェアのプログラムコード（実行形式を含む）を、その全体あるいは一部を記録した記録媒体により、各実施形態の装置に供給し、その装置のコンピュータ（又はＣＰＵ、ＭＰＵ）が記録媒体に格納されたプログラムコードを読み出して、動作の全部あるいは一部を実行することによっても達成されることは言うまでもない。この場合、記録媒体から読み出されたプログラムコード自体が各実施形態の機能を実現することになり、そのプログラムコードを記録した記録媒体は本発明を構成することになる。

【００７１】
プログラムコードを供給するための記録媒体としては、ＲＯＭ、フロッピー（登録商標）ディスク、ハードディスク、光ディスク、光磁気ディスク、ＣＤ－ＲＯＭ、ＣＤ－Ｒ、ＤＶＤ－ＲＯＭ、磁気テープ、不揮発性のメモリカード等を用いることができる。さらに、通信回線を介してダウンロードすることによってプログラムコードを供給するようにしてもよいし、ＪＡＶＡ（登録商標）などの技術を利用してプログラムコードを供給して実行するようにしてもよい。
【００７２】
また、コンピュータが読み出したプログラムコードを実行することにより、各実施形態の機能が実現されるだけでなく、そのプログラムコードの指示に基づき、コンピュータ上で稼動しているＯＳなどが実際の処理の一部又は全部を行い、その処理によって各実施形態の機能が実現される場合も本発明に含まれることは言うまでもない。
【００７３】
更に、記録媒体から読み出されたプログラムコードが、コンピュータに挿入された機能拡張ボードやコンピュータが接続された機能拡張ユニットに備わるメモリに書き込まれた後、そのプログラムコードの指示に基づき、その機能拡張ボードや機能拡張ユニットに備わるＣＰＵなどが実際の処理の一部又は全部を行い、その処理によって各実施形態の機能が実現される場合も本発明に含まれることは言うまでもない。
【００７４】
加えて、本発明はコンピュータに上記の各実施形態の装置の機能を実現させるためのプログラムを含むプログラム・プロダクトであってもよい。ここで、プログラム・プロダクトというのは、コンピュータ・プログラムだけでなく、プログラムを記録した記録媒体あるいはコンピュータを含むものである。
【産業上の利用可能性】
【００７５】
以上説明したように、本発明のソフトウェア開発支援システムでは、ソースコード生成手段が、設計情報に基づいて、一又は複数の挿入マーカがそれぞれ所定の箇所に挿入されたソースコードを生成し、挿入コード抽出手段が、ソースコードのうち各挿入マーカで指定される範囲に含まれるコードを挿入コードとして抽出して、その抽出された各挿入コードを、当該挿入コードについての識別情報と関連付けて挿入コード記憶手段に記憶しておく。そして、コード結合手段は、ソースコード生成手段がソースコードを生成したときに、挿入コード記憶手段に記憶されている挿入コードのうち、当該挿入コードについての識別情報によって特定されるソースコードが当該ソースコードと一致し且つその識別情報によって特定される挿入マーカが当該ソースコードに存在するものについて、当該挿入コードを当該ソースコードのうち当該挿入マーカで指定される範囲に結合する処理を行う。これにより、設計情報を変更することなく、開発者がソースコードに追加した内容を、再度生成したソースコードに反映させることができる。このため、開発者はソースコードが生成される度に、同じコードの入力を行う必要がなくなるので、開発効率の向上を図ることができる。したがって、本発明は、開発者によるソフトウェアの開発を支援するさまざまなシステムに適用することができる。
【図面の簡単な説明】
【００７６】
【図１】本発明の一実施形態であるソフトウェア開発支援システムの概略ブロック図である。
【図２】（ａ）は表形式で入力された設計情報の一例を示す図、（ｂ）は同図（ａ）の設計情報においてデータの追加が行われた場合の例を示す図である。
【図３】（ａ）はテキスト形式で入力された設計情報の一例を示す図、（ｂ）は同図（ａ）の設計情報においてデータの追加が行われた場合の例を示す図である。
【図４】ソースコードジェネレータによって作成されたソースコードの例を示す図である。

【図５】ソースコードジェネレータによって作成されたソースコードの例を示す図である。
【図６】ソースコードジェネレータによって作成されたソースコードの例を示す図である。
【図７】ソースコードジェネレータによって作成されたソースコードの例を示す図である。
【図８】ソースコードジェネレータによって作成されたソースコードの例を示す図である。
【図９】ソースコードジェネレータによって作成されたソースコードの例を示す図である。
【図１０】本実施形態のソフトウェア開発支援システムの処理手順を説明するための図である。
【図１１】ソースコードの内容を追加する場合の例を説明するための図である。
【符号の説明】
【００７７】
１０　入力装置
２０　表示装置
３０　記憶部
３１　ソースコード保存部
３２　挿入コード保存部
４０　制御部
４１　ソースコードジェネレータ
４２　コード編集部
４３　挿入コード抽出部
４４　コード結合部
１１０　データ型定義用テーブル
１２０　データ属性用テーブル

【図１】



【図２】

（ａ）

110

data definition	
data type	Person
description	人を表すデータ
package	jp.co.ns_sol.sysrdc

120

attributes list		
symbol	type	access limitation
name	String	
address	String	
dateOfBirth	Date	
age	Integer	readonly
phoneNo	String	

（ｂ）

110

data definition	
data type	Person
description	人を表すデータ
package	jp.co.ns_sol.sysrdc

120

attributes list		
symbol	type	access limitation
name	String	
address	String	
dateOfBirth	Date	
age	Integer	readonly
phoneNo	String	
mailAddress	String	

【図３】

（ａ）

```
package jp.co.ns_sol.sysrdc;

definition Person {
    attribute String    name;
    attribute String    address;
    attribute Date      dateOfBirth;
    attribute Integer   age (readonly);
    attribute String    phoneNo;
}
```

（ｂ）

```
package jp.co.ns_sol.sysrdc;

definition Person {
    attribute String    name;
    attribute String    address;
    attribute Date      dateOfBirth;
    attribute Integer   age (readonly);
    attribute String    phoneNo;
    attribute String    mailAddress;
}
```

【図４】

```
/*
 * Person.java
 *
 *    Copyright (C) 2005 NS Solutions Corporation.
 *    All Rights Reserved.
 *
 */

package jp.co.ns_sol.sysrdc.model;

import jp.co.ns_sol.sysrdc.common.*;
import jp.co.ns_sol.sysrdc.util.*;
import jp.co.ns_sol.sysrdc.value.*;
import jp.co.ns_sol.sysrdc.interfaces.*;
import jp.co.ns_sol.sysrdc.model.*;
// NSStella mixin point (imports) BEGIN  ] <---- M1
// NSStella mixin point (imports) END    ]

/**
 *      ユーザ定義データクラス: 人を表すデータ
// NSStella mixin point (comment) BEGIN  ] <---- M2
// NSStella mixin point (comment) END    ]
 */
public class Person extends BaseClass implements java.lang.Cloneable

    private String    name;
    private String    address;
    private Date    dateOfBirth;
    private Integer age;

    private String     phoneNo;
// NSStella mixin point (fields) BEGIN  ] <---- M3
// NSStella mixin point (fields) END    ]

    public String getName() throws ApplicationException {
```

【図５】

```
        return name;
    }

    public void setName(String name) throws ApplicationException {
        this.name = name;
    }

    public String getAddress() throws ApplicationException {
        return address;
    }

    public void setAddress(String address) throws ApplicationException {
        this.address = address;
    }

    public Data getDateOfBirth() throws ApplicationException {
        return dateOfBirth;
    }

    public void setAddress(Data dateOfBirth) throws ApplicationException
        this.dateOfBirth = dateOfBirth;
    }

    public Integer getAge() throws ApplicationException {
// NSStella mixin point (age_readonly) BEGIN
// NSStella mixin point (age_readonly) END
    }

    public String getPhoneNo() throws ApplicationException {
        return phoneNo;
    }

    public void setPhoneNo(String phoneNo) throws ApplicationException {
        this.phoneNo = phoneNo;
    }
```

M4

【図６】

```
    public void setBaseData() {
// NSStella mixin point (methods) BEGIN
        try {
            if (getBaseObject().getBaseData() != null) {
                setBaseData(convert(getBaseObject().getBaseData(), this))
            }
        } catch (BaseObjectException e) {
            throw new ApplicationException(e);
        }
// NSStella mixin point (methods) END
    }

}

// NSStella mixin point (file_tail) BEGIN
// NSStella mixin point (file_tail) END
```

M5

M5

M6

【図７】

```
/*
 * Person.java
 *
 *    Copyright (C) 2005 NS Solutions Corporation.
 *    All Rights Reserved.
 *
 */

package jp.co.ns_sol.sysrdc.model;

import jp.co.ns_sol.sysrdc.common.*;
import jp.co.ns_sol.sysrdc.util.*;
import jp.co.ns_sol.sysrdc.value.*;
import jp.co.ns_sol.sysrdc.interfaces.*;
import jp.co.ns_sol.sysrdc.model.*;
// NSStella mixin point (imports) BEGIN
// NSStella mixin point (imports) END

/**
 *      ユーザ定義データクラス: 人を表すデータ
// NSStella mixin point (comment) BEGIN
// NSStella mixin point (comment) END
 */
public class Person extends BaseClass implements java.lang.Cloneable

    private String    name;
    private String    address;
    private Date      dateOfBirth;
    private Integer   age;
    private String    phoneNo;
    private String    maileAddress;
// NSStella mixin point (fields) BEGIN
// NSStella mixin point (fields) END


    public String getName() throws ApplicationException {
```

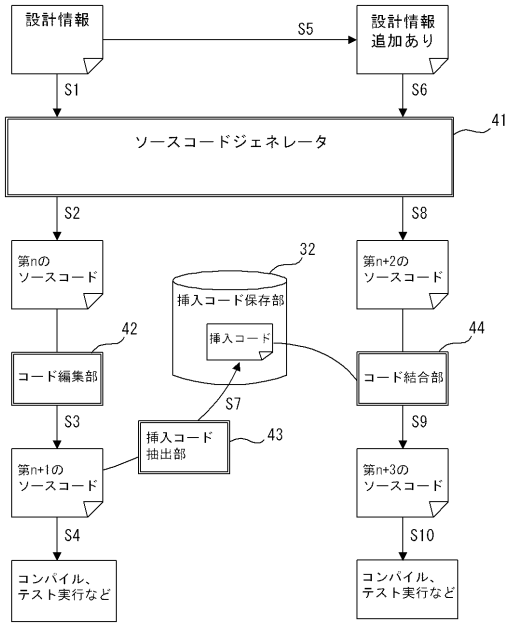
【図８】

```
        return name;
    }

    public void setName(String name) throws ApplicationException {
        this.name = name;
    }

    public String getAddress() throws ApplicationException {
        return address;
    }

    public void setAddress(String address) throws ApplicationException {
        this.address = address;
    }

    public Data getDateOfBirth() throws ApplicationException {
        return dateOfBirth;
    }

    public void setAddress(Data dateOfBirth) throws ApplicationException
        this.dateOfBirth = dateOfBirth;
    }

    public Integer getAge() throws ApplicationException {
// NSStella mixin point (age_readonly) BEGIN
// NSStella mixin point (age_readonly) END
    }

    public String getPhoneNo() throws ApplicationException {
        return phoneNo;
    }

    public void setPhoneNo(String phoneNo) throws ApplicationException {
        this.phoneNo = phoneNo;
    }
```

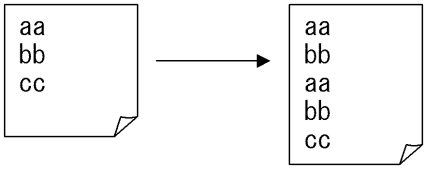
【図９】

```
    public String getMailAddress() throws ApplicationException {
        return mailAddress;
    }

    public void setMailAddress(String mailAddress) throws ApplicationException {
        this.mailAddress = mailAddress;
    }


     public void setBaseData() {
// NSStella mixin point (methods) BEGIN
        try {
            if (getBaseObject().getBaseData() != null) {
                setBaseData(convert(getBaseObject().getBaseData(), this));
            }
        } catch (BaseObjectException e) {
            throw new ApplicationException(e);
        }
// NSStella mixin point (methods) END
    }

}

// NSStella mixin point (file_tail) BEGIN
// NSStella mixin point (file_tail) END
```

【図１０】



【図１１】



フロントページの続き

審査官　林　毅

(56)参考文献　特開２００３－２７１３８２（ＪＰ，Ａ）
特開２００１－２８２５２０（ＪＰ，Ａ）
特開平０４－１７１５３２（ＪＰ，Ａ）
特開平０６－１０３０４２（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／４４