2004－SE－145　(1)
2004／8／19

制約に基づくソフトウェア開発計画自動立案システム -システムを実装するための戦略とその評価-

木下大輔[†1]　林雄一郎[†1]　八重樫理人[†1]
上之薗和弘[†1]　橋浦弘明[†1]　古宮誠一[†1]

あらまし　著者らは、プロジェクト管理者の負担を軽減すると共に、高度なプロジェクト管理ができるようにするために、ソフトウェア開発計画を自動的に立案するシステムを研究開発中である。そこでは、様々な戦略に基づいてソフトウェア開発計画を立案できるようなシステムの構築を目指している。そのためには、柔軟にシステムを拡張できるようなシステム構造になっていなければならない。この論文では、システム実装のために採用した戦略を明らかにするとともに、実装したシステムの評価を述べている。

An Automatic Schedule Planning System: Strategy and Evaluation for Implementing the System

Daisuke Kinoshita[†1]　Yuichiro Hayashi[†1]　Rihito Yaegashi[†1]
Kazuhiro Uenosono[†1]　Hiroaki Hashiura[†1]　Seiichi Komiya[†1]

Abstract The authors are studying and developing an automatic schedule planning system for software development so that it is possible for project managers to manage software projects in high level and to reduce the work load of the managers. They are aiming at developing a system to create a software development schedule plan based on various strategies. In order to achieve this purpose、 it is required that the system is implemented so as to be able to enhance the system easily. This paper clarifies strategies for implementing the system and evaluates the system implemented based on the strategies.

1. はじめに

インターネットや携帯電話などの普及等による情報通信技術の急速な社会への浸透、そのニーズの増加、更に技術の進歩に伴いソフトウェアの開発要求が高まっている。しかも、開発されるソフトウェアの大規模化･複雑化傾向は益々すすみ、ソフトウェア開発の高生産性・高信頼性が求められている。このような状況を反映して最近はプロジェクトマネジメントのための技術がますます重要視されてきている。

大規模なソフトウェアの開発は労働力を結集するためにプロジェクトを組んで行われるのが一般的である。どのようなライフサイクルモデルを採用しようとも、ソフトウェア開発プロジェクトには、開発計画（＝開発のための作業スケジュールや各作業への要員割当などに関する計画）というものが必ず存在する。従って、プロジェクトを成功へと導くためには、ソフトウェア開発計画を基に管理目標を設定し、その達成度をフォローするという方法が効果的である。しかし、ソフトウェア開発計画に則って進捗をフォローすることは、実は容易なことではない。なぜなら、ソフトウェア開発作業は頭の中で行われるので、検討の深さが他人からは見えないからである。このため、機械化できる部分はできるだけ機械化し、プロジェクト管理者が担当者と対話する事によって、検討の深さを確認するための時間を確保する事が必要である。また、ソフトウェア開発においては、作業量の見積もりやプロジェクトにおけるリスクの予測等が困難なために、ソフトウェア開発計画の立案が困難だからである。したがって、機械化によってプロジェクト管理者の負担を軽減すると共にソフトウェア開発計画の立案を支援する、ソフトウェア開発計画自動立案システムを研究開発する。

PMDB[7]、Design-Net[4][5]、KyotoDB[6]、PROMX[8]などのように、これまでにもソフトウェア開発プロジェクトの作業構造を表現する種々のモデルが数多く提案されている。しかし、これらのモデルは作業の階層構造や先行後続関係の表現に焦点を当てているので、プロジェクトの作業構造を表現するモデルとしては有用であるが、ソフトウェア開発における作業とその実行を可能にするリソースとの関連や、リソースの割当条件や割当可能期間に関する制約を示的に扱ってはいない。（PMDB では、実体として人（Person）をあげているが、その割当可能期間に関する制約は扱っていない。）このため、これらはソフトウェア開発プロジェクトのプロジェクト管理のためのモデルとしては不十分である。

既に文献[1][2]において、遺伝的アルゴリズム(Genetic Algorithm: GA)を用いて構築したシステムの枠組みがソフト

†1 芝浦工業大学大学院
〒337-8570 埼玉県さいたま市見沼区深作 307

ウェア開発計画の立案に有効であることをシステムの実行例を用いて示した。ところで、遺伝的アルゴリズムを使用するシステムでは、使用する遺伝的オペレータ(染色体を操作する演算)を具体的にどのような演算にするかはシステム設計者に任されているが、どのようなアプリケーションにどのような演算が適しているかということが未だよく分かっていない。このように、遺伝的アルゴリズムを使用するシステムは、実用的なシステムを構築するには実装レベルで未だ不明な部分が多い。遺伝的アルゴリズムを用いて実用的なシステムを構築するには、複数の演算やアルゴリズムをシステムに実装して、それらの適用実験をすることにより、それらの中から最適なものを選択するしか方法はない。このため、遺伝的アルゴリズムを使用するシステムは、柔軟にシステムを拡張できるようなシステム構造になっていなければならない。しかし、文献[1][2]のシステムでは、そのような柔軟なシステム構造にはなっていなかった。そこで、われわれは遺伝的アルゴリズムの特徴を活かすと同時に、ソフトウェア開発計画立案問題の性質をうまく活用するための枠組みを考え、オブジェクト指向言語 Java を用いて開発を行った。本稿では遺伝的アルゴリズムの特徴とソフトウェア開発計画立案問題の制約を明らかにするとともに、遺伝的アルゴリズムを使用したソフトウェア開発計画自動立案システムの特徴から現れる問題点を解決するためのシステム実装の戦略について述べる。さらに、行った実装に対して評価を行い、その結果を考察する。

本稿の構成をいかに示す。2.1.ではソフトウェア開発計画立案問題における制約の種類について述べる。2.2.ではソフトウェア開発計画立案問題を解くことは、制約条件を満足するリソースの割り当て最適な問題を解くことであるということを示し、遺伝的アルゴリズムを導入した理由について述べる。3.では遺伝的アルゴリズムを使用したソフトウェア開発計画自動立案システムの特徴より現れる問題を挙げ、その問題を解決するために採った実装のための戦略について述べる。4.では実装のための戦略に基づいて実装されたシステムのアーキテクチャについて説明する。5.では実装されたシステムについて評価を行い、その結果を考察する。6.では本稿のまとめについて述べる。

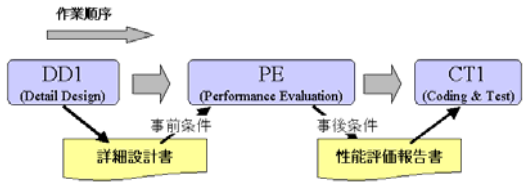

図 1:中間成果物による作業順序制約

2. 開発計画立案問題の制約と GA の導入理由

2.1 開発計画立案問題における制約の種類

本研究では、ソフトウェア開発計画立案の際に生じる問題を制約として捉えた。ソフトウェア開発計画立案問題が持つ制約の内容を具体的に示す。

(1) 作業順序に関する制約

ソフトウェア開発の各工程には、中間成果物を介して、それらの実施順序が決定する。図 1 を例に挙げて説明する。PE(性能見積もり)を実施するには DD1(詳細設計の一部)の工程が行われ、その成果物(中間成果物)である詳細設計書が作成されなければならない。故に、これを工程 PE の**事前条件**と呼ぶ。工程 PE が終了すると、その成果物として性能評価報告書が生成されなければならない。故に、これを PE の**事後条件**と呼ぶ。性能評価報告書は、これを基に CT1 (コーディング&テストの一部)が行われるので、同時に CT1 の事前条件でもある。このような制約を**作業順序に関する制約**と呼ぶ。

(2) リソースの割り当て条件に関する制約

ソフトウェア開発の各作業には、その作業を行うのに必要なスキル、資格、機能などを持つ要員(人的リソース)しか割り当てることができない。これを**リソースの割り当て条件に関する制約**と呼ぶ。例えば、コーディング&テストの作業では、当該プロジェクトで必要となるプログラミング言語を使いこなす能力を持たない人は割り当てることができない，などが制約となる。

(3) リソースの割り当て可能期間に関する制約

ソフトウェア開発の各作業には、リソースの割り当て条件を満たすリソースでもその期間にスケジュールが空いてなければ割り当てることができない。これを**リソースの割り当て可能期間に関する制約**と呼ぶ。非人的リソースでも同様のことが言える。それ故、ソフトウェア開発計画は人的リソースと非人的リソースの割り当て可能期間に依存する。

(4) リソースの実行効率に関する制約

リソースが 1 日で実施可能な作業量には限界がある。例えば、要員がある 1 日にＡという作業とＢという作業をそれぞれ４時間実施するという作業計画は、この要員の作業時間が合計で８時間になってしまう。１日に割り当て可能な時間を最大で８時間と考えたとき、合計で８時間となる複数の作業を８時間でこなすことは実は不可能である。何故なら，誰もがどのプロジェクトにも属さない作業を抱えているからである。（その人の職位が高くなればなるほど，それぞれのプロジェクトの作業に専念できる割合は低くなるという傾向にある。）そこで、リソースごとに異なる値を持つ **efficiency (実行効率)** という概念を導入し、どんなに効率よく働いても、例えば８時間の 80%しか有効に働けないと考える。このような制約を**リソースの実行効率に関する制約**と呼ぶ。話を簡単にするために，本稿ではすべての要員の実行効率を 100%とする。

2.2 遺伝的アルゴリズムの導入理由

ソフトウェア開発計画を立案するとき，プロジェクト管理者は前節で述べたような制約を考慮しながら立案しなければならない。このときに考慮すべき制約の数は，割り当ての対象となる要員の候補者(その組織が抱えている開発要員)の数が多ければ多いほど多くなる。しかも，リソースの割り当て可能期間に関する制約に至っては，プロジェクトの管理サイクル(通常は毎週１回の決まった日)に合わせて，考慮すべき制約の内容が動的に変化する。特に，その組織で複数のプロジェクトが並列に進行する状況を想定してみよ。立案すべきプロジェクトは１つでも，ソフトウェア開発計画の立案時にプロジェクト管理者が考慮しなければならない制約の数は膨大なものとなる。ソフトウェア開発計画の立案問題はこのような状況下での組み合わせ最適化問題である。このような問題に，制約を満足させながらリソース割り当てを試行錯誤する方法を採用するのでは，立案作業を行うプロジェクト管理者の負担は計り知れない。このような状況では，制約を考慮せずに次から次へと計画案を立案してから，それぞれの計画案がそれぞれ制約を満足するかどうかをチェックする方法(生成検査法)を採用するほうが得策である。遺伝的アルゴリズムは生成検査法による問題解決に適している。

遺伝的アルゴリズム（Genetic Algorithm）は生物が環境に適応して進化していく過程を工学的に模倣したアルゴリズムである。対象となる解を GA で扱うために、文字列の並びで表現された染色体という形でコーディングする。この染色体の集合を母集団と呼び、母集団に対して**「交叉」「突然変異」**などの遺伝的操作を行うことによって次世代の母集団を生成する。この母集団の中から適合度により解の取捨選択を行う。これを繰り返すことによって適合度の高い解を探索する。なお、遺伝的操作を遺伝的オペレータと呼ぶ。

GA の計算モデルの根本的な仕組みは‘Generate & Test’である。これは、解の候補を次々に生成し、それに対する評価(満足の度合)によって解の取捨を決定する方法である。すなわち、GA は手当たり次第に組み合わせを生成するので、染色体同士の組み合わせの数が増えると指数関数的に計算時間が増大する。その結果、組み合わせの数が増えると計算機時間が膨大になるという欠点がある。しかし、ソフトウェア開発計画立案問題において、すべての制約を満たす解というのはわずかな数でしかない。また、制約や解の評価関数の数が増加しても計算時間にはほとんど影響を与えないという利点があるので、ソフトウェア開発計画の立案問題に適用した場合、その解答に要する時間は長くないと考えられる。さらに、GA では、解の評価関数を問題における制約という形で与えるので、対象領域の知識があれば解法の知識がなくても容易に近似値を得ることができるという利点をもっている。本研究はこれらの利点ゆえに GA を採用した。

3. 実装のための戦略

本システムはソフトウェア開発のプロジェクト管理に利用するシステムである。メンバは進捗報告や実績データなどの入力、管理者はメンバの状況確認作業を、システムを使用して行う必要がある。しかし、プロジェクトによっては、担当者がプロジェクト管理者から離れた場所(例えば客先)で開発している場合がある。このような場合に、いちいち専用回線を引いて使用するシステムでは利用しにくい。そこで、プロジェクト管理者やメンバが場所や時間を選ばずに利用できるように、Web アプリケーションシステムの形態で実現した。Web アプリケーションシステムとして実現する場合、JSP+JavaBeans+Servlet でシステム構築すると、他の方法よりもオーバヘッドが少ない。これらの理由により、JSP+JavaBeans＋Servlet+Java でシステム構築する。本章では、本システムのソフトウェア開発を立案する処理を行うモジュールに関して説明を行う。

ソフトウェア開発計画立案問題を解決するための手段として GA が適しているということは既に述べた。GA を利用したソフトウェア開発計画自動立案システムは以下のような特徴を持つ。

(1)解の候補を絞り込むために多様な評価関数が与えられる
(2)遺伝的オペレータの多様な定義が考えられる
(3)状況に応じたパラメータ設定が必要である
(4)インスタンスの生成と破棄が大量に発生する

上記の特徴はソフトウェア開発計画立案問題の解決には非常に有効であるが、ソフトウェアの実装には大きな問題を引き起こす原因となっている。この問題を解決しなければ、実用的なシステムを構築することはできない。そこで、我々はソフトウェア開発計画立案問題の制約と遺伝的アルゴリズムの特徴を活かせるような設計を行った。その際に GoF が提案したデザインパターン(文献[3] 参照)をいくつか採用した。デザインパターンとはオブジェクト指向ソフトウェア開発において、過去に何度も設計された経験を蓄積し、カタログとしてまとめられたものである。デザインパターンを使用することで、成功した設計やアーキテクチャの再利用が容易にできる。また、デザインパターンはクラスやオブジェクト間の通信の仕様やそれの意図するところを明確に示すので、既存のシステムの文書化や保守性の向上、設計者や開発者の相互理解を助ける設計の語彙としても有効である。これらの理由により我々が開発をしたソフトウェアの設計にデザインパターンを採用した。以下で、前述した特徴の問題について詳しく述べるとともに、その問題を解決するために採用したデザインパターンについて詳しく述べる。

(1) 解の候補を絞り込むために多様な評価関数が与えられる

ソフトウェア開発計画立案問題では，解の候補の選択方法はプロジェクトごとに異なる。開発計画全体で要するトータルコストの最小なものを選択する，開発期間が最短のものを選ぶ､各要員の信頼性を**ソフトウェア信頼性指数** (各要員が過去 1 年間に作成したプログラムの規模を，ソフトウェア出荷後のバグ発生数で割ったもの)で表したときに，最も信頼性が高くなるようなプロジェクト・メンバの構成を選ぶ (文献[1]参照)，などの方法がある。そこで、選択方法を複数用意し、使い分けられるようにする必要がある。

(2) 遺伝的オペレータの多様な定義が考えられる

遺伝的アルゴリズムでは、様々な遺伝的オペレータというものが存在する。遺伝的オペレータとは、選択、交叉、突然変異など遺伝的操作を行うオペレータのことである。これらの実現方法のそれぞれがどのような問題領域に対して有効であるのかが明らかになっていない。そのため、実験を通じてどの遺伝的オペレータが有効であるか試行錯誤する必要がある。そこで、遺伝的オペレータを問題領域に応じて使い分けることができる仕組みが必要である。

「交叉」という遺伝的オペレータは 2 つの染色体を組み合わせて次世代の染色体を生成する処理である。**「突然変異」**という遺伝的オペレータは染色体の一部をまったく違う遺伝子に書き換えることによって、次世代の染色体を生成するという処理である。**「選択」**という遺伝的オペレータはある評価値によって染色体の集合から、その部分集合を選択する処理である。したがって、染色体の集合をそれぞれ A、B、C とするとき、以下のような関数と定義できる。

交叉　$f : A \times B \to C$
突然変異　$g : A \to B$
選択　$h : A \to B \quad (A \supset B)$

しかし、$f(a), g(a), h(a)$をそれぞれ集合 C、B、B 上のどの要素に対応させるかはシステム設計者に任されている。これまでにも対応のさせ方に関してはさまざまなアルゴリズムが提案されている。例えば、**「選択」**という遺伝的オペレータの具体的な実現方法には以下のようなものがある。ルーレット選択、期待値選択、ランキング選択、トーナメント選択、エリート選択などである。しかし、これらのアルゴリズムがそれぞれどのような問題領域に適しているかはまだわかっていない。ここで、我々はソフトウェア開発計画立案問題に適したアルゴリズムをこれらの中から選択したい。そこで、これらをあらかじめシステムに備えておき、自由に選択できるようなソフトウェア構造が必要である。また、これらのアルゴリズムで不十分な場合には新しく提案できるようなソフトウェア構造が必要である。

(1)(2)の問題を解決するために Strategy Pattern を利用する。Strategy パターンとは、戦略を利用するためのインターフェースを規定し、アルゴリズムはそのインターフェースにそった形でひとつのクラスとしてカプセル化する。そして、クライアント側はインターフェースの具体的な実装クラスを使用する。これにより、利用するクライアント側はアルゴリズムを独立に変更できる。それは、クライアントと戦略を利用するためのインターフェースは委譲(Delegation) という緩やかな結びつきとなっているためである。

(3) 状況に応じたパラメータ設定が必要である

GA では、個体数や交叉率、突然変異率など設定すべきパラメータが多い。また、これらのパラメータは解の探索能力に大きな影響を及ぼす。さらに、対象とする問題によって最適なパラメータが大きく異なるので、最適なパラメータを知るために多くの予備実験をする必要がある。つまり、パラメータを容易に変更できる枠組みが必要である。そこで、Template Pattern を利用することでこの問題を解決する。

Template Method パターンとは 1 つのオペレーションにアルゴリズムのスケルトンを定義しておき、その中のいくつかのステップについては、サブクラスでの定義に任せるというものである。これにより、アルゴリズムの構造を変えずに、アルゴリズム中にあるステップをサブクラスで再定義することが可能である。GA ではパラメータにより、部分的なアルゴリズムは変更するが、システムの全体的な処理の流れ(ソフトウェアのアーキテクチャ)は変わらない。つまり、Template Method パターンを用いることで、システムの部分的なアルゴリズムを容易に変更できるシステムの枠組みを構築することができる。具体的には、システムの一連の処理の流れはスーパークラスに定義しておき、パラメータを初期化する処理に関してはサブクラスで実装する。これにより、システムの全体のアルゴリズムを変更することなくシステムの部分的なアルゴリズムを容易に変更することが可能になる。

(4) インスタンスの生成と破棄が大量に発生する

遺伝的アルゴリズムの根本的な計算モデルは“Generate & Test”である。これは、解の候補を次々に生成し、それに対する評価(満足の度合)によって解の取捨を決定する方法である。そのため、解の候補が増えれば増えるほどインスタンスの数が増える。インスタンス数の増加はメモリの消費量が増えると同時に、メモリ領域の確保や開放に伴ったパフォーマンスの低下も同時に引き起こす。この問題を解決するために Flyweight Pattern を利用する。

flyweight とは、複数の箇所から同時に利用されうる共有オブジェクトのことである。flyweight は複数の箇所で独立した情報として振舞う。flyweight と、共有されていないオブジェクトを区別することはできない。また、flyweight は利用される箇所に関して何も仮定をおくことができない。ここで重要となる概念は intrinsic 状態と extrinsic 状態の区別である。intrinsic 状態は flyweight の内部に格納される。この状態は、flyweight オブジェクトが利用されるものには依存しない情報から成る。したがって、intrinsic 状態は共有できる。一方、extrinsic 状態は flyweight が利用されるものに依存し、そのものによって異なる。したがって、extrinsic 状態を共有することはできない。flyweight が extrinsic 状態を利用するときには、クライアントオブジェクトがそれを flyweight に渡さなければならない。

開発計画案を解として表現するには工程を遺伝子とし、工程の並びを染色体として表現するのがオブジェクトモデルとしては自然である。しかし、工程を遺伝子として利用すると、工程の表現する情報は非常に大きいので、多くのメモリを必要とする。そこで、遺伝子を数字で表現し、その数字を工程オブジェクトと対応付ける。そして、対応するオブジェクトを基に染色体の意味を解釈する。これにより、遺伝子を intrinsic 状態とし、工程オブジェクトを extrinsic 状態として扱うことができる。これにより、遺伝子を共有することができる。また、工程オブジェクトと遺伝子の対応関係を変えるだけで、解の表現する情報を容易に変更することができる。

図 2 を例に挙げて説明する。今 1302 と 0132 という数字の並びの染色体があるとする。この数字は遺伝子を表しており、同じ数字であるなら、どの染色体に存在しても同じ情報として扱われる。この情報を Pool しておき、1302 の染色体の 1 と 0132 の染色体の 1 が Pool されているインスタンスを両方とも参照することで共有することができる。また、数字は問題領域によって変化する情報と対応している。その情報を基に染色体が何を表現しているかを解釈するのである。図では 0 に対して工程 SA の情報が対応している。もし、0 に対応する情報を工程 DR に変更するとしても、染色体の構造は変えずに染色体の表現する意味だけを変更することが可能である。

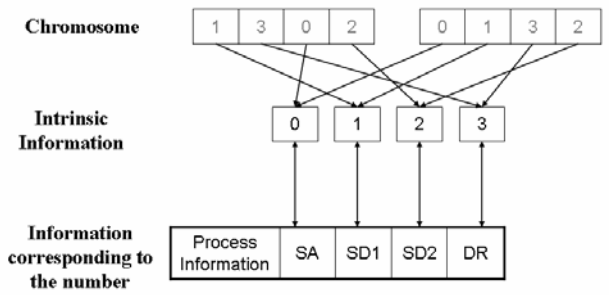

図 2:共有されているオブジェクトの例

4. 戦略に基づくアーキテクチャ

我々は、上記で述べた戦略を反映した設計を行い、その戦略に基づきシステムの開発を行った。システムは GA をフレームワーク(GAFramework)として実現した。GAFramework には Flyweight パターンと Strategy パターン、Template Method パターンが使用されている。GAFramework は図 3 のように構成される。

AbstractGAOperator クラスは評価(Evaluation)、交叉(Crossover)、選択(Selection)、突然変異(Mutation)という Java 言語におけるインターフェースを保持している。AbstractGAOperator クラスはインターフェースを実装したクラスを必要な処理の際に染色体を扱うクラス GeneticPool に渡す。GeneticPool クラスは渡された変数を使用することで処理を行う。Java ではインターフェースの型として定義した変数を作成することができる。その変数は必ずインターフェースを継承する必要があるが、そのインターフェースで定義されたメソッドの実装が保証されている。例えば、Selection インターフェースを実装したクラス、Roulette クラス(ルーレットホイールアルゴリズムを実装したクラス)と Elite クラス(エリート選択を実装したクラス)があるとする。これらは具体的なアルゴリズムこそ異なるが、選択の処理をするということに関しては保証されている。つまり、AbstractGAOperator クラスで Roulette クラスを使用するか、Elite クラスを使用するかを指定すれば、GeneticPool クラスは実装を意識することなくその処理を実現することができる。これにより、遺伝的オペレータを容易に変更することを可能にした。

AbstractGAOperator クラスには GA の一連の処理の流れが記述されている。しかし、使用するパラメータに関しては定義されていない。パラメータの定義はサブクラス MyGAOperator クラスに記述することで、パラメータに依存せずに GA を実行することが可能となった。また、染色体の生成は AbstractChromFactory クラスで行われているが、その生成の方法は問題によって異なる。そこで、生成の方法をサブクラスの MyChromFactory クラスが染色体を生成することによって、問題に応じて Chrom オブジェクトの生成方法を容易に変更できるようにした。これらは Template Method パターンによって実現されている。

AbstractGAFactory クラスは intrinsic な情報である Gene オブジェクトを保持している。Gene オブジェクトは共有可能な情報であるので、生成の必要がない。そこで、新しく Gene オブジェクトを生成せず、オブジェクトの参照を Chrom オブジェクトに渡す。これにより、無駄なインスタンス生成を防ぐことができる。これは Flyweight パターンを用いることで実現されている。

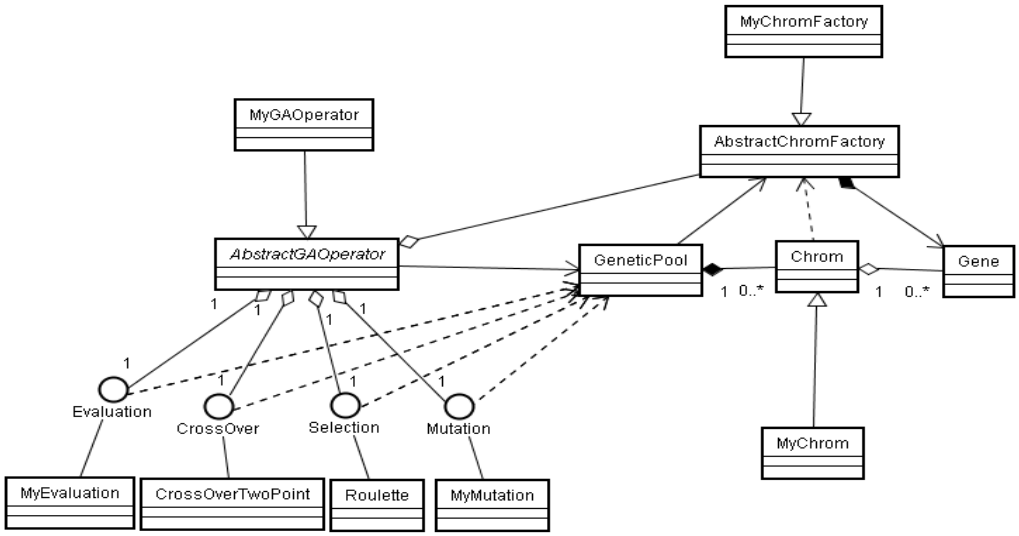

図 3：GAFramework のクラス図

5. 実験

パターンというものは、ソースコードの生成規則(この場合は作成規則)であって、ソースコードの固まりで示されるものではない、ということを Biggerstaff はソフトウェア再利用技術の分類(文献[9]参照)の中で示した。それ故、パターン物理的な形を持たないものだから、例でしか示すことができない。デザインパターンも設計の要素、それらの関連および責任、協調関係のみを高い抽象レベルで表現している。そのため、特定の具体的な設計や実装の方法は具体的に記述できない。GoF のデザインパターンでは，部分的にはサンプルコードは示されているものの，具体的なインプリメンテーションの形では殆ど示されていない。結城氏によるデザインパターンの本(文献[10]参照)で示されたインプリメンテーションは，GoF のデザインパターンに対する彼の解釈であり，GoF のメンバが示したインプリメンテーションではない。そのため、デザインパターンを利用して開発された実用的なシステムという粒度ではその有効性を検証している報告を我々は知らない。そこで、開発しているシステムを用いてその有効性を確認した。

本章では、利用した 3 つのデザインパターンの中から Flyweight Pattern を適用した効果を確認するための実験を行った。

5.1 アプローチ

GA の計算モデルは生成検査法である。インスタンスを生成し、評価関数を基に取捨を選択する。これを繰り返すことで解の探索を行う。つまり、その過程において、インスタンスの生成と破棄が多く行われる。そこで、Flyweight パターンを採用し、無駄なインスタンスの生成を抑えるという実装の戦略を立てた。

5.2 実験方法

実験には以下のようなプログラムを使用した。

- Flyweight パターンを採用したプログラムと採用していないプログラム
- 個体数が 100 個、1000 個、10000 個
- 遺伝子の種類が 10、20、30、40、50 種類

計 30 種類のプログラムに対して計測を行った。

Flyweight パターンを採用したプログラムでは、初めに Gene のインスタンスの生成を遺伝子の種類の数だけ行う。そして、Chrom オブジェクトはすでに生成されている Gene オブジェクトを使用して生成される。これに対して、Flyweight パターンを採用しないプログラムでは、Chrom オブジェクトを生成するときに必要とされる Gene オブジェクトを再度生成する。個体数と遺伝子の種類の数を変えることで、この効果を測定できると考えた。

5.3 考察

まず、実行時間を見てみる。個体数が 100 個の場合、染色体の種類の数がどの場合でも実行時間に変化はなかった。しかし、個体数が 1000 個、10000 個の場合、実行時間が 10%ほど速くなった(表 2、3 参照)。

実行時間が早くなった要因を調べるためにメソッドの実行回数と実行時間を計測した。Flyweight パターンを採用したプログラムでは Gene オブジェクトの生成の回数は遺伝子の種類の数だけ行われたが、Flyweight パターンを採用しなかったプログラムでは個体数×遺伝子の種類の数だけ行われた。実行時間は個体数が 10000 個の時には遺伝子の種類が多くなるほど時間は増えた。しかし、インスタンスの生成処理にかかる時間は Flyweight Pattern を採用したプログラムで処理全体の 0.01%、Flyweight Pattern を採用しなかったプログラムでも処理全体の 0.15%にしか

すぎなかった。そこで、Gene オブジェクトと Chrom オブジェクトを母集団に生成する前と後で使用されているメモリ領域の測定、ガーベッジコレクションの実行時間とその処理時間全体に対する割合を調べた。ガーベッジコレクションは JVM(Java Virtual Machine)のメモリ管理機能である。

図 4、5、6 を見ればわかるように Flyweight パターンを採用したプログラムと採用しなかったプログラムでは、メモリの使用量に大きな差が出た。これは Flyweight パターンを採用しないプログラムでは Chrom オブジェクトを生成するたびに Gene オブジェクトを生成するため、多くのメモリ領域が必要だからと考えられる。

個体数が 100 個のときは、ガーベッジコレクションにかかった時間は差がなかった。しかし、個体数が 1000 個以上ではガーベッジコレクションの実行時間に明らかな差が出た(表 4 参照)。さらに、Flyweight パターンを採用したプログラムではガーベッジコレクションの実行時間は処理全体の 3%~6%にしか過ぎなかったが、Flyweight パターンを採用していないプログラムでは個体数 1000 個の場合では処理全体の 10%、10000 個では処理全体の 7%～8%ぐらいを占めている(表 5 参照)。Flyweight パターンを採用していないプログラムでは多くの Gene オブジェクトが破棄される。このため、Flyweight パターンを採用したプログラムと採用しないプログラムでは、ガーベッジコレクションの実行時間に差がでたと考えられる。また、解の評価の処理で Flyweight パターンを採用したプログラムと Flyweight パターンを採用しなかったプログラムでは差があった。採用したプログラムでは同じ Gene オブジェクトのアドレスに何度もアクセスしている。これに関しては検証する方法がなかったために推定の域を出ていないが、JVM の JIT コンパイラが最適化処理を行っている可能性がある。

上記の結果より、実装の戦略に基づいて開発したシステムがその戦略を反映していることが確認できた。それにより、処理時間のパフォーマンスが向上することも確認した。また、GA の計算モデルである生成検査法に Flyweight Pattern を適用することが有効であることがわかった。われわれは,GoF のデザインパターンの本を読み，我々の解釈で具体的なインプリメンテーションを与えた。GoF のデザインパターンの本で紹介されている各パターンの特徴をその一部ではあるが，今回の実験によりそれが事実であると確認した。

表 2：Flyweight パターンを採用したプログラムの実行時間

	100					1000					10000				
population	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50
time	30.1	43.1	30.0	35.1	41.0	140.2	177.2	170.4	207.1	243.2	1499.1	2027.0	1769.2	2151.0	2433.3

表 3：Flyweight パターンを採用しなかったプログラムの実行時間

	100					1000					10000				
population	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50
time	35.0	33.0	43.0	43.0	41.0	145.1	200.5	193.4	241.5	275.4	1655.3	2234.2	2019.9	2380.3	2751.0

表 4：Flyweight パターンを採用しなかったプログラムの GC の処理時間と全体の処理時間に対する割合

	100					1000					10000				
	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50
time	3.3	3.2	3.9	5.1	4.0	12.4	21.8	30.3	37.6	37.9	96.7	148.0	182.9	214.2	263.2
%	5.4	5.4	5.8	7.0	5.8	6.5	8.2	11.4	11.8	10.8	5.0	7.9	7.9	7.8	8.3

表 5：Flyweight パターンを採用したプログラムの GC の処理時間と全体の処理時間に対する割合

	100					1000					10000				
	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50
time	3.3	3.3	2.8	3.6	4.1	6.3	10.5	7.6	11.7	12.2	48.9	63.6	71.6	84.8	84.1
%	5.5	4.6	4.7	5.5	5.8	3.4	4.5	3.4	4.4	4.1	2.9	2.8	3.5	3.5	3.1

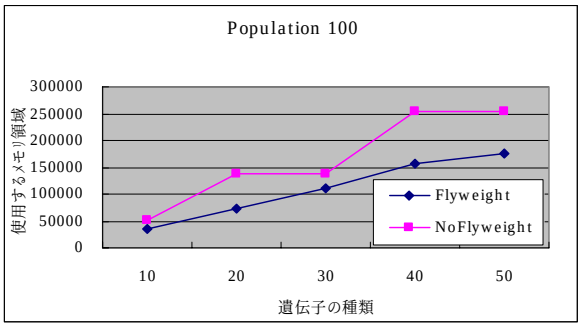

図 4：個体数 100 のときのメモリの使用量

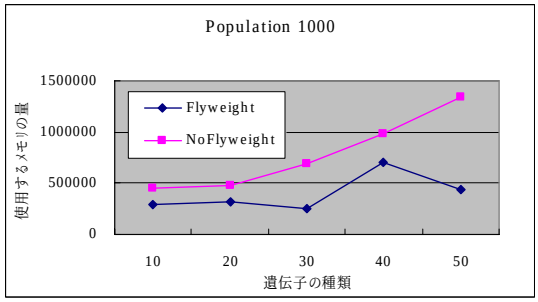

図 5：個体数 1000 のときのメモリの使用量

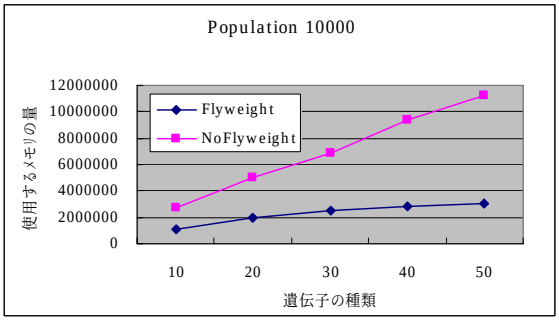

図 6：個体数 10000 のときのメモリの使用量

6. 終わりに

本稿では、ソフトウェア開発計画立案問題に含まれる制約について述べ、ソフトウェア開発計画立案問題に遺伝的アルゴリズムが適していることを示した。そして、遺伝的アルゴリズムを使用するソフトウェア開発計画自動立案システムの特徴から現れる実装上の問題を解決するための実装のための戦略を提案した。そして、採用した実装のための戦略の一つを取り上げ、戦略に基づいて開発したシステムと、そうではないシステムとの比較実験を行って評価した。その結果、戦略に基づいて開発したシステムが戦略を反映していることが確認できた。

今後の課題として、実験を通じて、ソフトウェア開発計画立案問題に適したパラメータを導く予定である。

謝辞　本研究を行うに当たり、ご協力いただいた芝浦工業大学情報工学科知能ソフトウェア工学研究室の関係者各位に感謝をいたします。

参 考 文 献

[1]S. Komiya N. Sawabe and A. Hazeyama、”Constraints-Based Schedule Planning for Software Development、” IEICE Trans. Inf. & Syst.、 vol. J79-D-I、 no9、 pp544-557、 Sept 1996.

[2]S. Komiya A. Hazeyama、”A Meta-Model of Work Structure of Software Project and a Framework for Software Project Management System 、 ”IEICE Trans INF & SYST 、 vol.E81-D.No12、pp1415-1428、Dec 1998.

[3]Erich Gamma、 Richard Helm、 Ralph Johnson、 John Vlissides、 “Design Pattern Elements of Reusable Object-Oriented Software”

[4]Liu,L and Horowitz, E., "Object Database Support for a software Project Management Environment," ACM SIGSOFT Software Engineering Notes Vol.13, No5, pp.85-96(Nov.1988)A. Gupta, R. Jain: Visual Information Retrieval, Communications of the ACM, Vol.40, No.5, ｐｐ.71-79 (1997).

[5]Liu,L and Horowitz, E., "A formal model for software project management," IEEE Transaction on Software Engineering, Vol. 15, No. 10, pp.1280-1293, Oct. 1989.

[6]Matsumoto Y and Ajisaka T, A Data Model in the Software Project Database KyotoDB, JSSST Advances in Software Engineering Environments, (the International Conference on Software Engineering pp.150-157 (1985)

[7]Penedo M.H.and Stuckle, E.D., PMDB - A Project Master Database For Software Engineering Environments, 8th International Conference on Software Engineering, pp.150-157 (1985).

[8]Sato, H. "project management expert system," Proc. ACM CSC'87, Feb. 1987.

[9]Biggerstaff, T., J. Perlis, A. J.: "Foreword to IEEE Transaction on Software Engineering," Vol. SE-10, No.5, pp.474-477, Sept. 1984.

[10]結城浩　著 “Java 言語で学ぶデザインパターン”, Softbank Publishing June 2001.