余裕日をも考慮に入れたソフトウェア開発計画の自動立案

高須賀 公紀[†] 嶋村彰吾[††] 内川裕貴[††]
木下大輔[†] 林 雄一郎[†] 古宮誠一[†]

著者らは，ソフトウェア開発プロジェクトのスケジュール立案と開発を担当する要員の割当てを自動的に行うシステムを研究開発してきた．これまでは，開発期間が最短となるような戦略で計画案を立案するシステムを主に研究してきた．このため，先行工程と後続工程の間の余裕日を考慮する必要はなかった．しかし，開発期間最短で開発コストが最小となるような戦略で計画案を立案する場合には，余裕日を考慮することで，よりコストの安い要員の割当てが可能になることがある．このため，本稿ではPDM（Precedence Diagramming Method）法を導入し，各工程の余裕日を計算することにより，余裕日をも考慮した計画案を自動的に立案する仕組みを提案するとともに，提案した仕組みの有効性を検証している．

Automatically Creating Software Development Plans to Have Taken Slack Time into Consideration

Kiminori Takasuka,[†] Shogo Shimamura,[††] Hiroki Uchikawa,[††]
Daisuke Kinoshita,[†] Yuichirou Hayashi[†] and Seiichi Komiya[†]

The authors have been developing an automatic software development schedule planning system which creates schedules and assigns workers for software development. The systems which they developed before now mainly were created software development schedule plans with the shortest development period strategy. For this reason, slack time between a predecessor process and a successor process did not need to be taken into consideration when the system was developed. However, when a plan which is the shortest development period and the minimum cost is desired, it could be possible to assign cheaper cost workers by taking slack time into consideration. Therefore this paper proposes a mechanism for creating software development schedule plans to have taken slack time into consideration through calculating slack time of each process by using PDM (Precedence Diagramming Method). And the paper proves that the proposed mechanism is effective in creating software development schedule plans stated above.

1. はじめに

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てなどに関する計画）というものが必ず存在する．したがって，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，計画どおりに作業が進んでいるかどうか確認することは，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，マネジメント作業の中で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ことによって検討の深さを確認するための時間を確保することが必要である．また，作業量の見積りやリスクの予測が行われた後，それらの結果を基にソフトウェア開発計画を立案することは，経験豊かなプロジェクト管理者であっても容易な作業ではない．なぜなら，日々変化する個々の要員のスケジュールやスキルなど，考慮しなければならない事柄が非常に多いからである．また，要員の急病，作業見積りの不備，突然の仕様変更などにより，たびたび計画を練り直すことになると，プロジェクト管理者には大きな負担となる．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

† 芝浦工業大学大学院
Graduate School of Shibaura Institute of Technology
†† 芝浦工業大学
Shibaura Institute of Technology

とともに，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．

これまでに開発したシステム[1)~5),13)]は，利用可能な自組織の要員を，遺伝的アルゴリズム（GA: Genetic Algorithm）を用いて自動的にプロジェクトの各工程に割り当てることによって開発計画の立案を行ってきた．そこでは，立案に際しては開発期間が最短となるような計画案を優先的に採用する戦略をとってきた．しかし，最近のソフトウェア開発では，短納期でかつ低コストでの開発が求められている．低コスト化の手段として，開発期間を延ばしてでも低コストのリソースを割り当てる方法が考えられる．しかし，開発期間を延ばすと，短納期でかつ低コストであるというニーズに応えることはできない．すなわち，全体の開発期間を変えずに，より低コストのリソースを割り当てるという戦略で開発計画を立案することが必要となる．

ところで，ソフトウェア開発工程にはクリティカルパスが必ず存在する．このクリティカルパスとは，先行工程と後続工程の間に日程的な余裕がまったくない工程どうしのつながりを意味し，これに対して，着手を遅らせても全体の開発期間が変わらない工程（日程にゆとりのある工程）もまた存在する．本稿では，この意味で工程上のゆとりを余裕日と定義し，余裕日を持つ工程に要員を割り当てることにより低コスト化を図る．

これまでに開発したシステム[1)~5),13)]では，開発期間が最短となるような戦略をとってきた．このため，先行工程と後続工程の間の余裕日を考慮する必要がなかった．つまり，**図 1** の上段の例では工程 C に対しては，要員 (高橋) のみが割当て対象となっていた．しかし，余裕日を考慮に入れると，図 1 の下段のように要員 (鈴木) も割当て可能になる．その結果，よりコストの安い要員の割当ても可能になることがある．たとえば，要員 (高橋) と要員 (鈴木) の時間単価がそれぞれ 2,500 円と 1,800 円だったとすると，要員 (鈴木) を割り当てることでより安いコストを実現できる．このように，余裕日を考慮に入れることでよりコストの安い計画案を生成できる可能性がある．

そこで，本稿では，これまでに開発したシステム[1)~5),13)]で用いていた GA による要員割当ての仕組みに加えて，新たに PDM（Precedence Diagramming Method）法を導入することにより，余裕日をも考慮した計画案を自動的に立案・生成する仕組みを提案するとともに，提案した仕組みの有効性を検証する．

本稿の構成を以下に示す．まず，2 章でソフトウェア開発計画を立案するうえで前提となるソフトウェア開発計画立案問題が持つ制約とその内容について述べる．そして，ソフトウェア開発計画立案問題とは，制約を満足するリソースの組合せ最適化問題を解くことであることを示し，最適化問題を解くのに GA を採用していることを述べる．3 章では，各工程における要員割当て人数をどのように与えるのかを述べる．4 章では，これまで開発したシステム[1)~5),13)]における要員の割当ての仕組みと本稿で提案する割当ての仕組みの差異を簡単に述べる．5 章では，本稿で提案する要員割当ての仕組みを述べるうえで必要となる PDM 法について述べる．その後，6 章で本稿の提案となる余裕日を考慮に入れたソフトウェア開発計画の自動立案の仕組みを具体的に述べる．7 章では，6 章で提案する仕組みの有効性を，単純な例題を使って検証する．8 章では，関連研究との比較について述べる．9 章では，本稿のまとめを述べる．

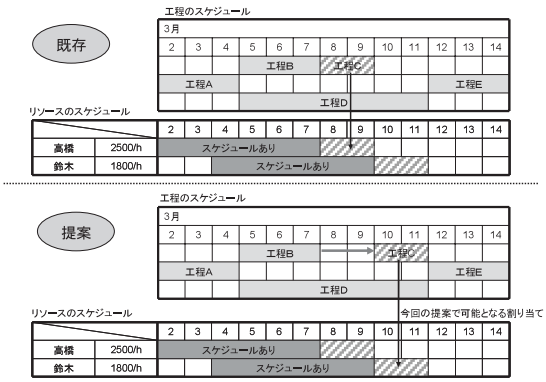

図 1 余裕日を考慮に入れた割当て
Fig. 1 Worker assignments to have taken slack time into consideration.

2. ソフトウェア開発計画立案問題が持つ制約

本研究では，ソフトウェア開発計画が満たさなくてはならない条件を制約としてとらえる．以下にソフトウェア開発計画立案問題の持つ制約の内容を具体的に記す．

(1) 作業順序に関する制約

ソフトウェア開発の各工程は，中間成果物を媒介として，それらの実施順序が決定する．たとえば，**図 2** の工程 b を例にあげて説明する．

工程 b を実施するには，工程 b に着手する前に工程 a による成果物（中間成果物）α が生成されていなければならない．これを工程 b の事前条件と呼ぶ．また，工程 c に着手する前に工程 b の成果物（中間成果物）β が生成されていなければならない．これを工程 b の事後条件と呼ぶ．中間成果物 α，β によって工程 a，b，c の実施順序が決まる．このような制約を**作**

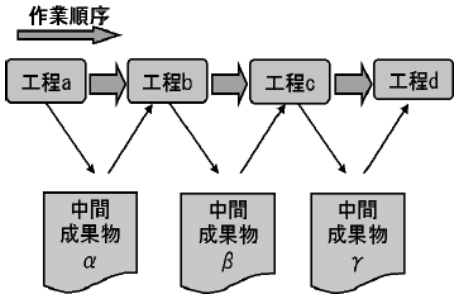

図 2 中間成果物に基づく作業の順序に関する制約
Fig. 2 Constraints on an operation sequence based on intermediate products.

業順序に関する制約と呼ぶ.

(2) **リソースの割当て条件に関する制約**

ソフトウェア開発の各作業には，その作業を実施するうえで必要となるスキル，資格，機能などを持つ人的リソース（要員），または非人的リソース（マシン環境など）しか割り当てることができない．これを，**リソースの割当て条件に関する制約**と呼ぶ．たとえば，プログラム開発言語や，システムのテスト，デバッグ作業などの工程はそれぞれの作業を遂行する能力を持つ者でなければ担当できない．このため，ソフトウェア開発の作業スケジュールは，ソフトウェア開発のための各作業が持つ人的リソースと非人的リソースの割当て条件に関する制約に依存する．

(3) **リソースの割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，割当て条件を満足するリソースでも，そのリソースにとって割当て可能な期間（= 空きスケジュール）にしか，そのリソースを割り当てることができないという制約がある．このような制約を**リソースの割当て可能期間に関する制約**と呼ぶ.

(4) **リソースの能力的限界に関する制約**

各リソースの能力的限界を表現するために容量（“capacity”）という概念を導入し，リソースの属性として表現する．具体的には，容量をそのリソースの稼働率（単位は%）の上限値で表現する．すなわち，そのリソースに割り当てられた 1 日の作業時間の合計（並列に進められる複数の作業に同一のリソースが割り当てられた場合にはそれらの合計）を，そのリソースが 1 日稼動可能な時間で割り，その値に 100 を掛けることによって得られる値を稼働率とする．そして，あらかじめ設定された各リソースの稼働率の上限をもって，その**リソースの能力的限界に関する制約**と呼ぶ．この制約を導入すれば，稼働率を作業者の負荷状況を示す尺度として利用することができ，作業者に対する過度の作業の割当てをチェックできる．非人的リソースに対しても同様の概念を導入する．なお，容量（上限稼働率）は一般にリソースのランクによって異なると考えられる．また，稼動率が 100%つまり 1 日に割り当てられる上限は現在 8 時間をデフォルトとしている．

プロジェクトの各工程に割り当てられたリソースの組合せが，制約をすべて満たしていれば，その組合せはソフトウェア開発計画の候補と考えることができる．つまり，ソフトウェア開発計画を立案することは，制約の多い組合せ最適化問題を解くことであるといえる．なお，この最適化問題を解く仕組みとして，我々は遺伝的アルゴリズム（GA: Genetic Algorithm）を採用している．GA の計算モデルの根本的な仕組みは ‘Generate & Test’ である．解の候補を次々に生成し，制約を満たすもののみを解の候補として残す．しかる後に，それらの中で評価値の最も高いものを最終的に解として選択する．すでに文献 1)，2)，13) において，この問題への GA の導入理由を述べるとともに，導入した GA が，ソフトウェア開発計画の立案問題に有効であることを示した．

なお，以降において説明を簡略化するために，本稿では，非人的リソースについては考慮しないものとする．

3. 各工程における要員割当て人数の与え方

本研究では作業者の能力を｛よくできる，できる，不得手，遂行不可能｝の 4 段階で評価することにした．ここでの “よくできる” 能力とは 1 人で素早く作業することができ，かつ不得手な人を指導しながら作業することもできる人を，“できる” は他人を指導しながら作業をすることはできないが，1 人で作業することはできる人を，“不得手” は他人の指導があれば作業することができる人を，“遂行不可能” はプロジェクトの遂行中に他人の指導があっても，作業の遂行が不可能な人をそれぞれ意味している．通常の場合，各工程に割り当てられる要員の数は複数である．ある工程の要員割当てパターンが**表 1** のとおりであったとする．表 1 の例では，この工程の要員割当て人数のパターンが 3 種類であることを示している．Pattern1 は，（この工程に求められる技術が）“よくできる” の水準にある要員がこの工程を 1 人で担当すると，作業に 3 日かかることを示している．Pattern2 は，“よくできる” の水準にある要員 1 人と，“不得手” の水準にある要員 1 人の計 2 人で担当すると，作業に 2 日かかることを示している．Pattern3 は，“できる” の水準にあ

表 1 割当てパターン例
Table 1 An example of worker assignment patterns.

	Pattern1	Pattern2	Pattern3
A(よくできる)	1	1	
B(できる)			2
C(不得手)		1	
D(遂行不可能)			
所要日数	3	2	2

る要員が 2 人で担当すれば作業に 2 日かかることを示している．このような要員割当てパターンを基に，各工程での作業遂行に必要な要員の数を指定する．本システムでは，指定されたすべての要員割当てパターンを試行し評価することにより要員割当てを行う．

なお，割当てパターンはあらかじめプロジェクト管理者が工程ごとに見積もった作業量を基に，表 1 の形式で工程ごとにシステムに入力することを想定している．本稿では，このような割当てパターンがあらかじめ入力されていると仮定したうえで今後の話を展開していく．

4. これまでと本稿で提案する割当ての仕組みとの差異

本システムは，ソフトウェア開発計画立案問題を表現するために**図 3** で示すような，GA の染色体を上位層と下位層の 2 階層に分ける階層型コーディングを採用している．上位層では，各遺伝子が作業工程を表し，遺伝子の並びである染色体で作業順序を表現する．下位層では，各遺伝子が割当てパターンを表し，各々は上位層と下位層の遺伝子座（染色体上で各遺伝子の置かれている位置）が同じものがそれぞれ対応している．たとえば，図 3 の例では設計レビュー（DR）に Pattern1 が選択されていることを表している．

これまで採用していた要員割当ての仕組み[5] を簡単に述べると，以下の手順となる．

- 工程の並びの候補を生成し，制約条件を満足しているかどうか評価する（制約を満足しているものだけを残す）．
- 割当てパターンの並びの候補を生成し，制約条件を満足しているかどうか評価する（制約を満足しているものだけを残す）．
- 割当てパターンに基づいて要員を割り当て，制約条件を満足しているかどうか評価する（制約を満足しているものだけを残す）．

しかし，この仕組みだと図 1 の上段のように，先行工程が完了した直後に（つまり，余裕日のない状態で）後続工程が開始され，その工程のスケジュールに基づいた要員割当てしか行うことができない．

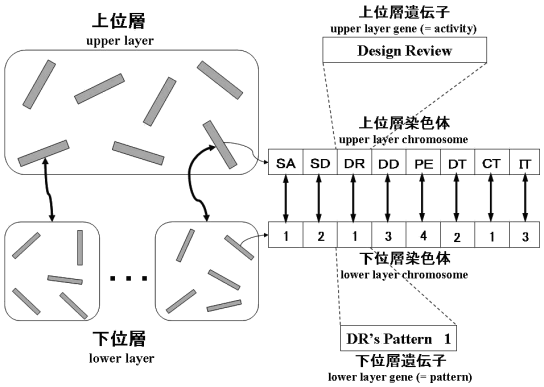

図 3 2 階層 GA モデルの構造
Fig. 3 The structure of two-layered GA.

そこで，図 1 の下段のような割当てを可能とするために，本稿の提案では，以下の手順で割当てを行う．

- 工程の並びの候補を生成し，制約条件を満足しているかどうか評価する（制約を満足しているものだけを残す）．
- 割当てパターンの並びの候補を生成し，制約条件を満足しているかどうか評価する（制約を満足しているものだけを残す）．
- PDM 法を用いて余裕日とクリティカルパスを自動算出する．
- 各工程の（余裕日の異なる）スケジュールパターンの生成を行う．
- スケジュールパターンの並びの候補を生成し，制約条件を満足しているかどうか評価する（制約を満足しているものだけを残す）．
- クリティカルパス上の工程に対して，割当てパターンに基づいて要員を割り当て，制約条件を満足しているかどうか評価する（制約を満足しているものだけを残す）．
- 非クリティカルパス上の工程に対して，割当てパターンに基づいて要員を割り当て，制約条件を満足しているかどうか評価する（制約を満足しているものだけを残す）．

割当ての仕組みがこれまでと同一なのは，上記における 2 つ目の項目までである．6 章で具体的な仕組みを述べるが，そのうえで必要となる PDM 法に関して 5 章で述べる．

5. PDM 法の導入による余裕日の算出

これまで開発してきたシステムでは，先行工程が完了した直後に後続工程が開始される．**図 4** を例にす

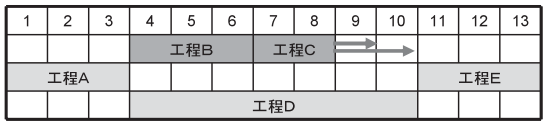

図 4 工程の実施順序の例
Fig. 4 An example of use.

ると，工程 A が完了した直後に工程 B と工程 D が開始され，工程 B が完了した直後に工程 C，工程 D が完了した直後に工程 E が開始されることを示す．ここで，実施順序と各工程の所要日数が決まると，クリティカルパスを算出することが可能となる．クリティカルパスとは，余裕日のまったくない工程どうしのつながりを意味する．図 4 の場合は工程 A・工程 D・工程 E というつながりがクリティカルパスに該当する．ここで，クリティカルパス上ではない工程 B・工程 C に着目すると，完了後 2 日間にわたって空白期間が存在していることが分かる．この期間を，本稿では工程 B と工程 C の**余裕日**と定義する．

余裕日を考慮すると工程 B と工程 C の工程を 1 日，または 2 日ずらすことが可能となる．すると，工程 B と工程 C に割り当てられるリソースの候補が増加する可能性が発生し，限られたリソース内での計画案の候補が増加する可能性が高くなる．そのため，より安い計画案の候補も生成される可能性があると考えられる．

大規模なソフトウェア開発になればなるほど余裕日を持った工程は増大する可能性があり，余裕日をも考慮に入れた計画案の立案を人手で行うことは，経験豊かなプロジェクト管理者でも非常に困難となる．

本研究では，余裕日を求めるのに **PDM（Precedence Diagramming Method）法**と呼ばれるネットワーク・スケジューリング手法を採用している[6),15)]．

大別すると，ネットワーク・スケジューリング手法には AOA（Activity On Arrow）法と AON（Activity On Node）法の 2 つが存在する．PERT（Program Evaluation and Review Technique）や CPM（critical path method）は，AOA 法に該当するが，WBS（Work Breakdown Structure）で詳細化した作業（ワークパッケージ）をネットワーク図に展開するのに不便な面がある．AON 法はこのような人間による取扱いの不便さを改良する手法の一環として提唱されたものである．PDM 法とは，基本的な作図法は AON 法と同じで作業（アクティビティ）をノードとして，その順序・依存関係をアロー（矢線）でつないでいき，依存関係にある 2 つの工程間の順序を論理的に 4 つの関係で定義することで表記する方法のことである．PERT などは先行作業が終了すればすぐに後続作業がスタートする “イベントドリブン型” なので，手作業でダミー線を引かないと計算できないケースがある．これに対し，PDM 法は連続する 2 つの作業の間にラグ（待ち時間）とリード（準備期間）の概念を持ち込み，より精緻な日程計画を表現できるように工夫されている．このため，手作業でダミー線を引くことなく，いつでも余裕日を計算できる．これが，採用理由である．

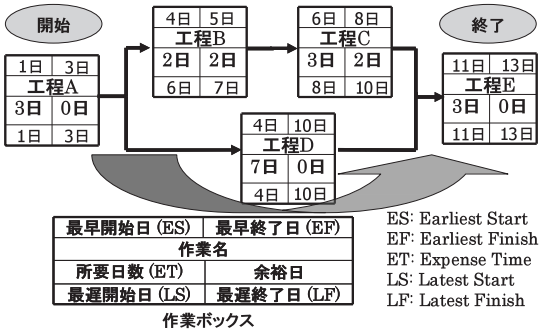

図 5 PDM 法の適用例
Fig. 5 An example of use of precedence diagramming method.

PDM 法では，並行作業はノードを上下に並べるが，順番に行われるアクティビティは次の 4 つの論理的順序関係で整理される．

- FS（finish-to-start）
- SS（start-to-start）
- SF（start-to-finish）
- FF（finish-to-finish）

以上の 4 つのうち，先行作業が終わると後続作業が始まる FS を採用している．

これをもとに，図 4 の工程のスケジュールから余裕日とクリティカルパスを求めている例を**図 5** に示す．

以下の手順で図 5 の各値が決まっていく．工程 A の ES を ESa，ET を ETa と標記する．そのほかの場合も同様に標記するものとする．

このとき，「開始」から「終了」の方向へ

① $\mathrm{ESa} = 0$

② ETa，ETb，ETc，ETd，ETe の値（与えられた値）を記入する

③ $\mathrm{EFa} = \mathrm{ESa} + \mathrm{ETa}$

④ $\mathrm{ESb} = \mathrm{EFa}$，$\mathrm{ESd} = \mathrm{EFa}$

⑤ $\mathrm{EFb} = \mathrm{ESb} + \mathrm{ETb}$，$\mathrm{EFd} = \mathrm{ESd} + \mathrm{ETd}$

⑥ $\mathrm{ESc} = \mathrm{EFb}$，$\mathrm{EFc} = \mathrm{ESc} + \mathrm{ETc}$

⑦ $\mathrm{ESe} = \max\{\mathrm{EFc}, \mathrm{EFd}\}$

⑧ $\mathrm{EFe} = \mathrm{ESe} + \mathrm{ETe}$

「終了」から「開始」の方向へ

⑨ $\mathrm{LFe} = \mathrm{EFe}$，$\mathrm{LSe} = \mathrm{LFe} - \mathrm{ETe}$

⑩ LFc = LSe，LFd = LSe
⑪ LSc = LFc − ETc，LSd = LFd − ETd
⑫ LFb = LSc，LSb = LFb − ETb
⑬ LFa = min{ LSb, LSd }，LSa = LFa − ETa
⑭ 各工程の ES と LS の差から余裕日を計算

の手順で余裕日などを計算する．

この例では，工程 B と工程 C が余裕日 2 日となる．また，余裕日の値が 0 日どうしのつながりがクリティカルパスであり，工程 A・工程 D・工程 E のつながりがこれに該当する．

6. 余裕日をも考慮した割当ての仕組み

本稿で提案する要員の割当ての仕組みを本章で具体的に述べる．

リソースの割当て処理を行うには，まず上位層において工程の並びを生成する．図 4 の実施順序は，**図 6** の形式に変換され，システムに記録される．図 6 をもとに工程の並びを生成する．ただし，以下の 3 つのいずれの解釈でも成立するような工程の並びとして生成される（このうちのどれを採用するかは，システムの実行の直前に与えられる制約条件のデータによって決定する）．

- A→B→C の順に作業を実施する．
- A が B，C よりも後に実施されてはならない．
- A，B，C が同時に実施されてもよい．

生成された工程の並び（染色体）は，作業順序に関する制約を満たしていなければ，破棄される．破棄されずに残った工程の並びを**図 7** に示す．

次に，制約を満足する上位層の染色体に対応する割当てパターンの並び（下位層の染色体）を生成する．生成された割当てパターンの並びを**図 8** に示す．

このとき，生成された割当てパターンの並びのそれぞれに対して，休日などを考慮して全体の所要日数を計算する．この時点で，納期を超えてしまった染色体は破棄する．破棄されずに残った染色体については，各工程の所要日数が決まるので，5 章で述べた PDM 法を用いてクリティカルパスと各工程の余裕日を求めることが可能となる．次にカレンダ情報を参照して，余裕日と土日祝日などを考慮した各工程の具体的な日付を付したスケジュールパターンの生成を行う．図 4 の工程 A と工程 B を例にスケジュールパターンを生成すると**図 9** のようになる．工程 A は余裕日が 0 日なのでパターンを 1 つ生成する．工程 B は余裕日が 2 日なので，0 日・1 日・2 日ずらした場合に対するパターンをそれぞれ生成する．このようにして，すべての工程に対して余裕日と土日祝日などを考慮したスケジュールパターンの生成を行う．

次に，**図 10** のようなスケジュールパターンの並びの生成を行い，生成されたスケジュールパターンの並

	必要な中間成果物	生成される中間成果物
工程A		α、β
工程B	α	γ
工程C	γ	δ
工程D	β	ε
工程E	δ、ε	

図 6 事前条件・事後条件
Fig. 6 Pre-conditions and post-conditions.

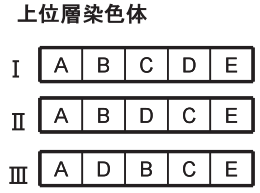

図 7 工程の並び
Fig. 7 Process sequence.

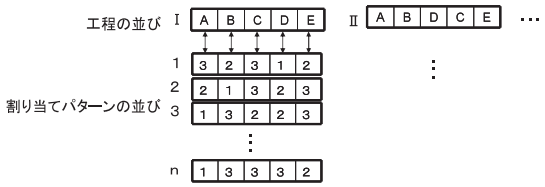

図 8 割当てパターンの並び
Fig. 8 The sequence of worker assignment patterns.

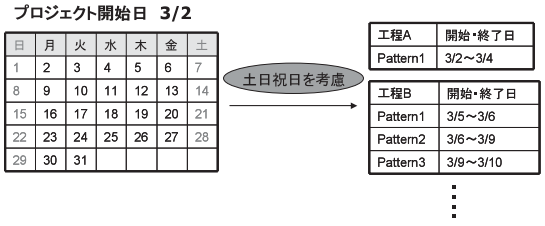

図 9 スケジュールパターン
Fig. 9 The patterns of scheduling.

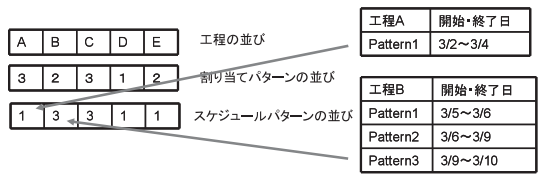

図 10 スケジュールパターンの並び
Fig. 10 The sequence of scheduling patterns.

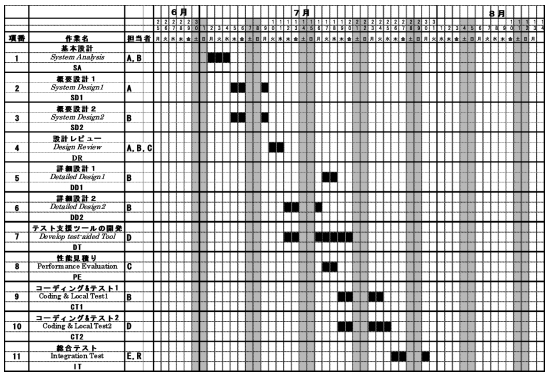

図 11 開発計画案の例
Fig. 11 An example of software development schedule plan.

びが作業順序の制約を満たしているかどうかの評価を再度行う．なぜなら，ここで余裕日を考慮したために，工程のずらし方により，工程 B と工程 C の日程が重なる場合があるからである．このとき，作業順序の制約を満足しているスケジュールパターンの並びに対して，工程ごとに割当てパターンに基づいて具体的なリソースの割当てを行っていく．各工程へのリソースの割当ては，クリティカルパス上の工程，非クリティカルパス上の工程の順に行っていく．このとき，リソースの割当て条件に関する制約とリソースの割当て可能期間に関する制約を考慮しながら割当てを行う．具体的には，まず工程を担当するために求められるスキルとスキルレベルを持った要員をデータベースからリストアップする．そして，工程の開始/終了日を算出してスケジュール的に割当てが可能な要員を検索して割当てを行う．このときに，制約を満足するリソースを検索することができないために割当て不可能となった場合には，その染色体は破棄される．そして，すべての工程に対して割当てパターンに基づいて要員の割当てを行うことができたものだけを開発計画案の候補として採用する．ここで，リソースの割当て情報と工程の開始/終了日などのデータをデータベースに格納する．このとき，開発計画案の候補が複数生成される場合があるので，候補の中から 1 つの案を選び出すための評価を行わなくてはならない．本システムでは，プロジェクト管理者が選択する「作業期間最短」「コスト最小」など戦略に基づいて計画案の候補の順位付けを行っている．また，同時に**図 11** のようなガントチャートなどの詳細情報を提示することにより，プロジェクト管理者は上記の戦略以外での計画案を選択することも可能である．

ここで，クリティカルパス，非クリティカルパスの順でリソースの割当てを行っているのは，工程管理上，より重要なクリティカルパスに有用な人材（要員）を優先的に割り当てるためである．このことは，システムの計算量の削減効果から見ても重要な効果をもたらしている．我々のシステムでは，GA を使って，より現実的な時間で，より良い計画案を生成するという考えの下で構築されている．このため，クリティカルパス上の工程にリソースの割当てができないのであれば，非クリティカルパス上の工程にリソースを割り当てることは時間の無駄となるからである．**図 12** を用いて割当て処理の効率化が実現できていることを説明する．図 12 は，1 つの割当てパターンの並び（図中では［1, 2, 1, 1, 1］となっている）に対して複数（図中では 3 種類）のスケジュールパターンの並びが存在している．このスケジュールパターンの並びそれぞれに対して，要員の割当てを行う．このとき，1 番左（1 番目に割当てを行う）のスケジュールパターンに対する割当てに着目すると，クリティカルパス上の工程で割当てが不可能であることを示している．すると，この割当てパターンの並びに対するスケジュールパターンの並びすべてに対して，以降の要員の割当ては無意味となる．この時点（最初に検討したスケジュールパターンの並び）でこの案を破棄すべきことであることが判明する．

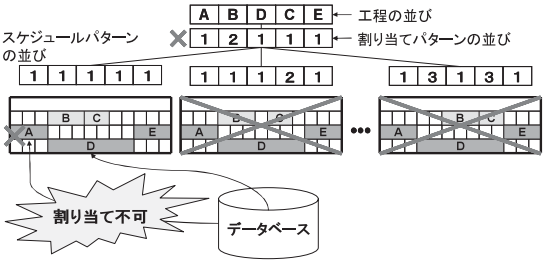

図 12 要員割当ての効率化
Fig. 12 Promotion of worker assignment efficiency.

これまで開発してきたシステムの要員割当ての仕組みでは，クリティカルパスの概念を要員の割当てに用いることはできなかった．そのため，1 つの割当てパターンの並びに対するスケジュールパターンの並びすべてに対して割当てを行い，その結果はじめて割当てパターンの並びの破棄の判断が可能となった．スケジュールパターンの並びが 3 つで，工程 A に割当て可能な要員がいなかったとすると，提案する仕組みの場合，1 つ目の工程で，従来の仕組みだと 15（$= 3 \times 5$）個すべての工程に対して割当てを行った時点ではじめて破棄の判断ができた．これらのことから，提案する仕組みによる処理時間の削減効果はあきらかである．

7. 開発計画案の自動立案例とその評価

本章では，理解を助けるために単純な開発計画案の自動立案例をあげ，本稿で提案する手法の有効性を示す．

［作業概要］

例題プロジェクトとして，2006 年 8 月 5 日から開催されるある展示会用のデモシステムを開発するプロジェクトを考える．このプロジェクトは 2006 年 7 月 3 日に組織され，2006 年 8 月 4 日までにシステムを完成させなくてはならない．システム開発のために行わなくてはならない作業は，基本設計，概要設計，概要設計レビュー，詳細設計，コーディング，単体テスト，統合テストである．開発計画案立案の際に考慮すべき制約を以下に列挙する．デモシステムは高性能を要求されたシステムであるため，詳細設計の一部分に対して性能見積りを行いコーディング作業に反映させる．性能見積りは性能解析技術を必要とする．展示会で使用するハードウェアは今回出展のための開発マシンであり，7 月 31 日にならないと利用できない．しかもこのマシンは，8 月 3 日には展示会場に搬出されなくてはならない．展示会で使用するマシンの使用期間に関する制約によりテストを効率良く行うためにテスト支援ツールの開発を行うことになった．このような状況下の中で計画の立案を行い，プロジェクトを開始した．

例題プロジェクトの人的リソースはそれぞれ以下の制約を持つ．

- プロジェクト管理者（Project Manager）はプロジェクトの管理，進捗の管理，承認活動を行う．
- 設計技術者（Design Engineer）の基本的な責務は設計とコーディング作業である．
- 品質保証技術者（Quality Assurance Engineer）の基本的な責務はテスト計画とその実施である．

各工程に必要なスキルを**表 2** に示す．各役割を担う要員としてプロジェクト管理者には X，各設計技術者は A，B，C，D，E，F の 6 名，品質保証技術者は G，H，I の 3 名がおり，この中から要員を割り当てる．ここでの人的リソースの属性値を**表 3** に示す．また，プロジェクトでは開発マシン，展示会で稼動するマシン，性能見積りを行う際に使用するツールを使用する．ここでの非人的リソースの属性値を**表 4** に示す．この例題プロジェクトの Activity Diagram は**図 13** となる．また，プロジェクト管理者により見積もられた割当てパターンを**表 5** に示す（簡略化のため各工程の割当てパターンは 1 つとする）．なお，1 日の作業時間は 8 時間とし，土日祝日は休みである．

以上の属性値（制約を含む）を考慮し，これまでの手法で立案した開発計画案を**図 14** の上段に示し，本稿で提案する手法で立案した開発計画案を図 14 の下

表 2 各工程に必要なスキル一覧
Table 2 A list of the skills required for assigning to each process.

	必要なスキル
SA (System Analysis)	OOSE, OMT, C
SD1 (System Design1)	OMT, TDD, C
SD2 (System Design2)	OMT, TDD, C
DR (Design Review)	OMT, C
DD1 (Detailed Design1)	OMT, C
DD2 (Detailed Design2)	OMT, C
DT (Develop test-aided Tool)	C, AOP
CT1 (Coding & Unit Test)	C
PE (Performance Evaluation)	PA (Performance Analysis)
CT2 (Coding & Unit Test)	C
IT (Integration Test)	C, JUNIT

表 3 人的リソースの属性値
Table 3 The attributes of each human resource.

Name	Skill	Skill Level	Available time	Cost (/h)
A	OOSE, OMT, PA	よくできる	Any	4000
B	OMT, TDD, C	できる	Any	3200
C	OMT, TDD, C	できる	Any	3000
D	OMT, TDD, C	できる	～7/7,7/18～	2800
E	C	不得手	Any	2500
F	OMT, TDD, C	できる	7/7～7/14,7/25～	2900
G	C, JUNIT	よくできる	Any	4200
H	C, JUNIT	できる	Any	3300
I	C, JUNIT	できる	Any	3500

表 4 非人的リソースの属性値
Table 4 The attributes of each non-human resource.

Name	Available time
Performance Analysis Tool	Any
Machine for Development	Any
Machine for Exhibition	7/31～8/2

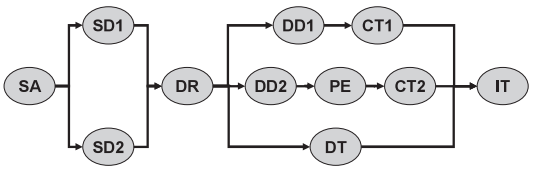

図 13 例題プロジェクトの Activity Diagram
Fig. 13 The Activity Diagram of an example project.

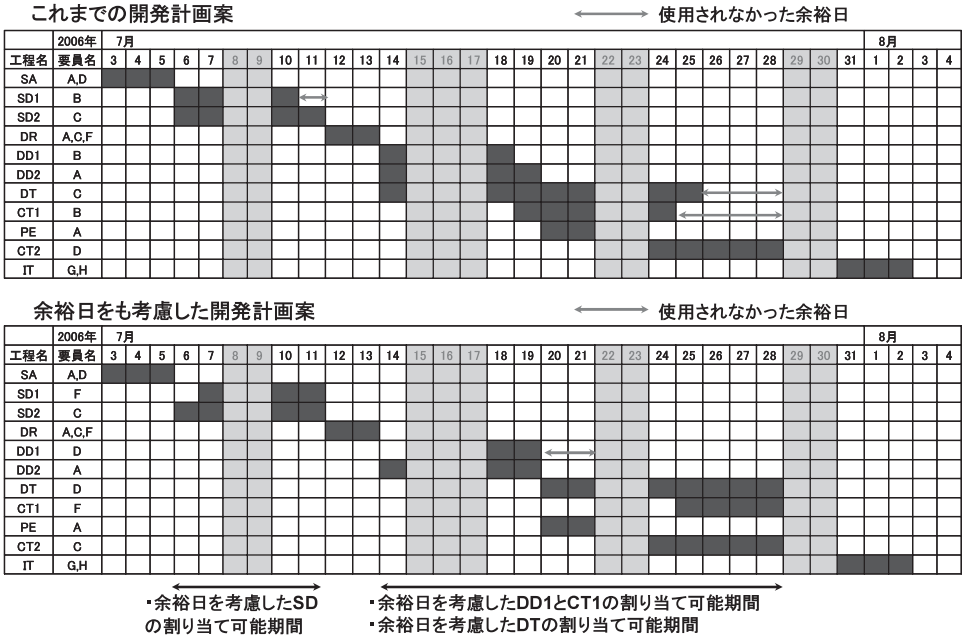

図 14 これまでと余裕日をも考慮した場合の開発計画案
Fig. 14 Current worker assignments vs. the worker assignments to have taken slack time into consideration.

表 5 各工程の割当てパターン
Table 5 The worker assignment patterns of each process.

	よくできる	できる	不得手	遂行不可能	所要日数
SA (System Analysis)	1	1			3
SD1 (System Design1)		1			3
SD2 (System Design2)		1			4
DR (Design Review)	1	2			2
DD1 (Detailed Design1)		1			2
DD2 (Detailed Design2)	1				3
DT (Develop test-aided Tool)		1			7
CT1 (Coding & Unit Test)		1			4
PE (Performance Evaluation)	1				2
CT2 (Coding & Unit Test)		1			5
IT (Integration Test)	1	1			3

段に示す．

図 14 の上段と下段で異なる箇所は，SD1 が B→F，DD1 が B→D，DT が C→D，CT1 が B→F，CT2 が D→C の 5 カ所である．

この計画案が生成された理由は，GA の計算モデルの根本的な仕組みが ‘Generate & Test’ であるため，次々と計画案の候補を生成していき，すべての制約を満たす計画案が最終的に廃棄されずに残ったためである．この案は以下の制約を満足している．読者もこの点に留意してほしい．

- 工程の作業遂行に必要なスキルを要員が保持している．
- 各工程のスケジュールに要員が割当て可能である．
- 各工程が余裕日を考慮したスケジュールに収まっている．

2 つの計画案を比較すると，これまでの開発計画案は変更部分のみのコストは 510,400 円となりトータルコストは 1,268,000 円となる．余裕日をも考慮した開発計画案は変更部分のみのコストは 484,000 円となりトータルコストは 1,241,600 円となる．この 2 つの計画案の差額は 26,400 円となり，コストダウンに成功していることが分かる．

以上のような単純な事例でも本手法がこれまでの手法よりも有効であることが理解できる．まして，プロジェクトの規模が大きくなり，割当てパターンが増えれば増えるほど立案は煩雑になり，人の手で余裕日をも考慮した立案を行うことは困難を極めるので，本手法の有効性はさらに大きくなる．

ここでは，それぞれ 1 つずつしか開発計画案を示していないが，実際は，「作業期間最短」「コスト最小」などの戦略に基づいてそれぞれ複数の計画案が順位付けされて立案される．その中から，プロジェクト管理者によって 1 つに絞られる．たとえば，**図 15** の中で工程 C に対して要員 A と要員 B のどちらが割り当てられている方を良いとするかは，プロジェクト管理者の考え方しだいである．要員 A だと，コストは安いが

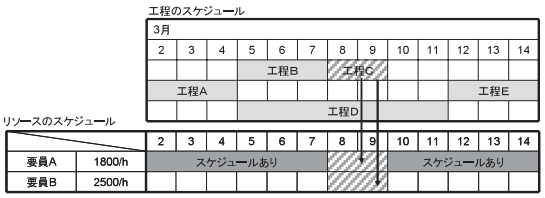

図 15 要員割当ての際の注意
Fig. 15 Cautions in case of worker assignment.

要員 A は自身のスケジュールに余裕がないため，遅れが生じたら対応できない．要員 B だと，コストは高いが要員 B は自身のスケジュールに余裕があるため，遅れが生じても問題はない．読者もこの点に留意してほしい．

8. 関連研究との比較

PMDB [8], Design-Net [9),10)], KyotoDB [11)], PROMX [12)] 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，リソースの割当て条件や割当て可能期間に関する制約を明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，その割当て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．

最後に，最近話題になっている CCPM（Critical Chain Project Management）[7),14),18),19)] と，我々のアプローチとの比較を行う．CCPM について議論するには，まず TOC（Theory of Constraints：制約条件の理論）[14),16),17)] から語らなければならない．TOC とは，企業活動の中で最も弱い部分（これを TOC では制約条件と呼んでいる）に着目し，そこを集中的に強化・改善することにより，最小の努力で最大の成果をあげようとするマネジメント手法である．このような TOC の考えに基づき，全体最適化の視点から行うプロジェクト管理手法が CCPM である．従来の Critical Path の代わりに Critical Chain を用い，工程見積りに含まれる安全性確保のための工数の割増し部分を取り除いて（具体的には，成功確率 50%の工数見積りを採用して）各工程の期間を短くする代わりに，Critical Path 上の工程の最後にプロジェクトバッファ（余裕日）を設けて集約して管理する．また，合流パスの遅れから Critical Chain を守るために，Critical Chain に合流する作業と Critical Chain 上の作業の間に合流バッファを挿入する．そして，納期やコスト，さらにはリソースの制約を考えて，プロジェクトのスケジュール案を立案する．このようにしてから，プロジェクトマネージャは，各工程の進捗を管理するのではなく，バッファの消費具合によってプロジェクト全体の進捗を把握する．上記が CCPM の概要である．

CCPM と我々のアプローチとの相違は次のとおりである．

- 工数見積りの考え方が異なる．CCPM では，各工程の工数見積りに成功確率 50%の工数見積りを採用し，合流バッファとプロジェクトバッファを用いて，見積り誤りによる工程遅延の危険性を低減させている．我々のアプローチでは，合流バッファとプロジェクトバッファによる工数の割増し部分を平均化して，各工程に振り分けている．
- 進捗管理の考え方が異なる．CCPM では，各工程の進捗を管理するのではなく，バッファの消費具合によってプロジェクト全体の進捗を管理する．このため，プロジェクト全体での工程遅延は把握できるが，Critical Chain 上にない工程の進捗把握には適していない．我々のアプローチでは工程ごとに進捗を管理する．このため，Critical Path 上にあるなしにかかわらず，すべての工程で進捗把握が可能である．
- 工程遅延が発覚したときに，対策のためのスケジュール案の再立案は CCPM では容易ではないが，我々のアプローチでは，すでに文献 3)，4)，13) で示したように，我々が開発したツールによって，工程遅延回復可能なスケジュール案の再立案が動的に行える．

9. おわりに

本稿では，短納期でかつ低コストが求められている現状に対しての対策として，余裕日を考慮に入れたソフトウェア開発計画を立案する仕組みを提案し，これまで開発してきたシステムの仕組みに比べ，開発期間を変えずに，よりコストの安い要員の割当てが可能であることを示した．それにより提案した仕組みの有効性を示した．また，要員の割当て処理において，提案した仕組みが処理時間の削減効果があることも示した．なお，参考までに示すと，著者の 1 人の経験では，50 人で 1 年半のプロジェクトの計画案の立案を行うのに，人手 1 人で 1 週間ほどの時間を要した．これを，本システムを使えば，4 時間ほどで計画案の立案

をすることができる．

謝辞 この研究に日頃協力していただいている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いたします．

参 考 文 献

1) 古宮誠一，澤部直太，櫨山淳雄：制約に基づくソフトウェア開発計画の立案，電子情報通信学会論文誌 D-I，Vol.J79-D-I, No.9, pp.544–557 (1996).
2) Komiya, S. and Hazeyama, A.: A Meta-Model of Work Structure of Software Project and a Framework for Software Project Management System, *IEICE Trans. Inf. Syst.*, Vol.E81-D, No.12, pp.1415–1428 (1998).
3) Yaegashi, R., Kinoshita, D., Hashiura, H., Uenosono, K. and Komiya, S.: Automatically Creating a Schedule Plan as Countermeasure by Means of "Crashing" against Process Delay, *JCKBSE'04*, Protvino, Russia, pp.24–36 (Aug. 2004).
4) 八重樫理人，木下大輔，橋浦弘明，上之薗和宏，林雄一郎，古宮誠一：工程遅延発生時におけるファーストトラッキングによる対策案の自動立案，電子情報通信学会論文誌 D-I，Vol.J88-D-I, No.2, pp.215–227 (2005).
5) 林雄一郎，八重樫理人，木下大輔，上之薗和宏，橋浦弘明，古宮誠一：ダミー割当て可能なソフトウェア開発計画自動立案システム，信学技報，KBSE2004-38, pp.19–24 (Jan. 2005).
6) 宮沢修二，宮沢正文：モダンプロジェクトマネジメントリテラシ II, *ITEC* (2001).
7) Leach, L.P.: *Critical Chain Project Management*, Artech House Professional Development Library, London 313 (2000).
8) 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).
9) Liu, L. and Horowitz, E.: A formal model for software project management, *IEEE Trans. Softw. Eng.*, Vol.15, No.10, pp.1280–1293 (1989).
10) Liu, L. and Horowitz, E.: Object database support for a software project management environment, *ACM SIGSOFT Software Engineering Notes*, Vol.13, No.5, pp.85–96 (1988).
11) Matsumoto, Y. and Ajisaka, T.: A data model in the software project database KyotoDB, *JSSST Advances in Software Science and Technology 2*, pp.103–121 (1990).
12) Sato, H.: Project management expert system, *Proc. ACM CSC'87* (Feb. 1987).
13) Kinoshita, D., Yaegashi, R., Hashiura, H., Uenosono, K. and Komiya, S.: An Automatic Schedule Planning System: Strategies and Evaluation for Implementing the System, *Joint Conference on Knowledge-Based Software Engineering 2004* (*JCKBSE'04*), Protvino, Russia, pp.37–48 (Aug. 2004).
14) 稲垣公夫：TOC クリティカル・チェーン革命，日本能率協会マネジメントセンター (1998).
15) Project Management Institute, Inc.：プロジェクトマネジメント知識体系ガイド第三版 (PMBOK ガイド)．
16) エリヤフ・ゴールドラット：ザ・ゴール，ダイヤモンド社 (2001).
17) エリヤフ・ゴールドラット：ザ・ゴール 2，ダイヤモンド社 (2002).
18) エリヤフ・ゴールドラット：クリティカルチェーン，ダイヤモンド社 (2003).
19) 中嶋秀隆，津曲公二：PM プロジェクト・マネジメントクリティカルチェーン，日本能率協会マネジメントセンター (2003).

(平成 18 年 11 月 30 日受付)
(平成 19 年 5 月 9 日採録)

高須賀公紀

平成 17 年芝浦工業大学工学部情報工学科卒業．平成 19 年芝浦工業大学大学院工学研究科修士課程修了．平成 19 年 KDDI 株式会社入社．プロジェクトマネジメントに興味を持つ

嶋村 彰吾

平成 15 年芝浦工業大学工学部情報工学科卒業．平成 19 年日本電子計算株式会社入社．

内川 裕貴

平成 15 年芝浦工業大学工学部情報工学科卒業．現在，芝浦工業大学大学院工学研究科修士課程在学中．

木下　大輔

平成 14 年芝浦工業大学工学部工業経営学科卒業．平成 16 年芝浦工業大学大学院工学研究科修士課程修了．平成 16 年株式会社日立製作所入社，現在に至る．芝浦工業大学大学院工学研究科博士後期課程在学中．ソフトウェア開発の管理を支援するツールの研究に従事．ソフトウェアプロジェクトマネジメントに関心を持つ．

林　雄一郎

平成 16 年芝浦工業大学工学部工業経営学科卒業．平成 18 年芝浦工業大学大学院工学研究科修士課程修了．平成 18 年株式会社リコー入社．P2P によるグリッド・コンピューティングに関心を持つ

古宮　誠一（正会員）

昭和 44 年埼玉大学理工学部数学科卒業．昭和 45 年（株）日立製作所入社．昭和 59 年特別認可法人情報処理技術者センター（略称 IPA）に出向し，自動プログラミングシステムをはじめとする各種 CASE ツールの構築技術，ソフトウェア設計方法論とそのメタ理論，CAI および知的 CAI 等の研究に従事．昭和 63 年～平成 12 年 IPA 技術センター特別研究員．平成 3 年～平成 9 年 IPA 新ソフトウェア構造化モデル研究本部長付を兼務．平成 5 年徳島大学客員教授．平成 7 年より千葉大学情報工学科非常勤講師．平成 9 年より芝浦工業大学客員教授兼同大学大学院非常勤講師．平成 12 年 3 月信州大学博士（工学）．平成 13 年より芝浦工業大学教授．平成 15 年より同大学専門職大学院（MOT）教授を兼務．平成 4・5 年/平成 6・7 年/平成 8・9 年知能ソフトウェア工学研究会幹事/副委員長/委員長．平成 8・9 年電子情報通信学会情報・システムソサエティ運営委員．平成 6 年～平成 9 年電子情報通信学会論文誌編集委員．平成 10・11 年電子情報通信学会論文誌編集委員．平成 10・11 年電子情報通信学会論文誌編集委員会幹事，現在に至る．