情報処理学会論文誌 Vol. 50 No. 12 3084–3094 (Dec. 2009)

テクニカルノート

摂動を用いた影響波及解析による ボトルネック工程の自動検出

木下大輔[†1] 内川裕貴[†1]
小坂祐也[†1] 古宮誠一[†1]

企業などでは，複数のプロジェクトが同時期に並行して進められている．このため，同一の人的リソース（作業者）または非人的リソース（開発環境など）が，同時期に複数のプロジェクトまたは作業で必要となり，使用されることがある．同一のリソースが同時期に複数のプロジェクトまたは作業で使用されているとき，そのようなリソースが使用されている工程に対して日程調整や使用しているリソースの変更などを行う場合，同時期に同一のリソースが使用されている別プロジェクトまたは別の作業との関係で，使用しているリソースの変更も日程調整もできないような状況が生じる．我々は，このような状況にある工程をボトルネック工程と呼んでいる．ボトルネック工程にまで工程遅延が波及した場合，日程調整やリソース変更などの対策が行えないために，プロジェクトを納期までに完了できなくなってしまうことがある．ボトルネック工程にまで工程遅延が波及しないようにするためには，ボトルネック工程よりも前の工程で遅延を解消するなどの対策を講ずる必要がある．そこで本稿では，実プロジェクトで起こりうる種々の工程遅延が実際に起こったと仮定して，仮の工程遅延を与えること（これを摂動と呼ぶ）により，工程遅延が後続工程にどのような影響を及ぼすかをシミュレート（影響波及解析）することにより，ボトルネック工程を自動検出する方法を提案するとともに，その方法の有効性を示している．

Automatic Detection of Bottleneck Processes through Perturbation-based Repercussion Analysis

Daisuke Kinoshita,[†1] Hiroki Uchikawa,[†1]
Yuya Kosaka[†1] and Seiichi Komiya[†1]

In the organizations such as companies, plural projects are carried out in parallel. In addition, plural projects share human and non-human resources (i.e. worker, development environment) to carry out each work. Therefore it is not rare that the demand that a project want to use shared resource between plural projects around the same time. In such a situation, there is the process that cannot change resource and adjust schedule by competing resource's schedule. We call such a process bottleneck process. If a delay spread to bottleneck process, project cannot complete at due date because it is not possible to adjust schedule or to change resource. It is necessary to take measures to prevent process delay from spreading to bottleneck process. Therefore this paper proposes a method to detect bottleneck process automatically by Perturbation-based Repercussion Analysis to simulate what repercussion effects on successive processes are affected by given delay (perturbation).

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．

企業などでは，複数のプロジェクトが同時期に並行して行われている．このため，同一の人的リソース（作業者）または非人的リソース（開発環境など）が，同時期に複数のプロジェクトまたは作業で必要となり，使用されることがある．同一のリソースが同時期に複数のプロジェクトまたは作業で使用されているとき，そのようなリソースが使用されている工程に対して日程調整や使用しているリソースの変更などを行う場合，同時期に同一のリソースが使用されている別プロジェクトまたは別の作業との関係で，使用しているリソースの変更も日程調整もできないような状況が生じる．このような状況にある工程を我々はボトルネック工程と呼んでいる．遅延がこのような工程にまで波及した場合，日程調整やリソース変更などの対策が行えないために，プロジェクトが納期までに完了できなくなってしまうことがある．ボトルネック工程にまで工程遅延が波及しないようにするためには，ボトルネック工程よりも前の工程で遅延を解消するなどの対策を講ずる必要がある．しかし，ボトル

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

ネック工程を事前に検出することは容易ではない．なぜなら，自プロジェクトだけでなく，他プロジェクトの開発計画をも把握しなければならないからであり，日々変化していくスケジュールをも把握しなければならないからである．

そこで我々は，実プロジェクトで起こりうる種々の工程遅延が実際に起こったと仮定して，仮の工程遅延を与えること（これを摂動という）により，工程遅延が後続工程にどのような影響を及ぼすかをシミュレート（影響波及解析）することにより，ボトルネック工程を自動検出する方法を提案するとともに，提案した方法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立案問題が持つ制約とその内容について述べる．3 章では，関連研究について述べる．4 章では，ボトルネック工程と潜在的ボトルネック工程の定義を述べるとともに，その事例を示す．5 章では，工程遅延が後続工程に及ぼす影響波及の解析方法を明らかにする．6 章では，4 章で示した事例を基に提案した解析方法の適用例を示す．7 章では，6 章で示した適用例（工程遅延がもたらす影響波及の解析事例）を基に，提案した解析方法の有効性を議論する．8 章では本稿のまとめを述べる．

2. ソフトウェア開発計画問題が持つ制約

本研究では，ソフトウェア開発計画案が満たさなければならない条件を制約としてとらえる[1),2)]．以下に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．**図 1** 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 a による成果物（中間成果物）α が工程 b に着手する前に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 b の成果物（中間成果物）β が工程 c に着手する前に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．

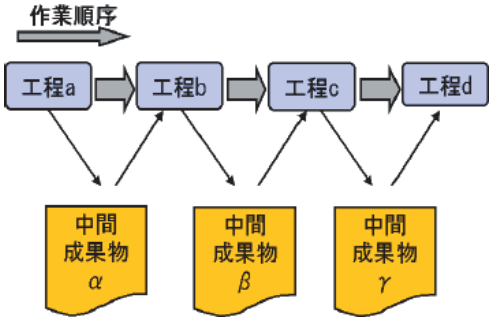


図 1　中間成果物に基づく作業順序に関する制約
Fig. 1　The constraints on working sequence based on intermediate products.

(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．たとえば，ある作業者 P1 に対して，1 週間（5 日）に作業 A（所要日数 2 日）と作業 B（所要日数 2 日）の 2 つが割り当てられていたとすると，作業者 P1 のその週の稼働率は 80%となる．このとき，P1 の稼働率の上限が 80%以上に設定されていれば，これらの作業は P1 に割当て可能であるが，80%未満に設定されていれば，これらの作業は P1 には割当て不可能となる．このようにすれば，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．

3. 関 連 研 究

PMDB [7]，Design-Net [8),9)]，KyotoDB [10]，PROMX [11)] 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）[6),13),16),17)] 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）[13)–15)] 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 4)，5) 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計画案の再立案が動的に行える．

4. ボトルネック工程と潜在的ボトルネック工程の定義とその事例

(1) ボトルネック工程の定義とその事例

複数のプロジェクトで使用しているリソースがあったとする．このリソースを使用している工程に遅延が波及したときに，その工程の日程を調整することも，その工程で使用するリソースを取り替えることもできない場合がある．このような状況にある工程をボトルネック工程と呼ぶ．**図 2** にその具体例を示す．

図 2 の上部は，プロジェクトのスケジュールを表し，矢印は「作業順序に関する制約」を示している．具体的には，図 2 の上部は「工程 A の作業が完了してから工程 B と D の作業が開始される」という作業順序に関する制約を示している．図 2 の下部はリソースの使用スケジュールを表している．理解しやすくするために，ここでは工程 C に割当て可能なリソースは α のみであると仮定する．図 2 はリソース α がプロジェクト X の工程 C と，プロジェクト Y の工程 Q と工程 R に割り当てられている状況を示している．この事例では，リソース α が割り当てられている，連続する 3 つの工程 C，Q，R の間には余裕日がまったくないことに読者は注意してほしい．

このような状況において，プロジェクト X の工程 B に遅延が発生した場合，「作業順序に関する制約」により，遅れた日数分だけ工程 C の開始日と終了日が予定より遅くなる．プ

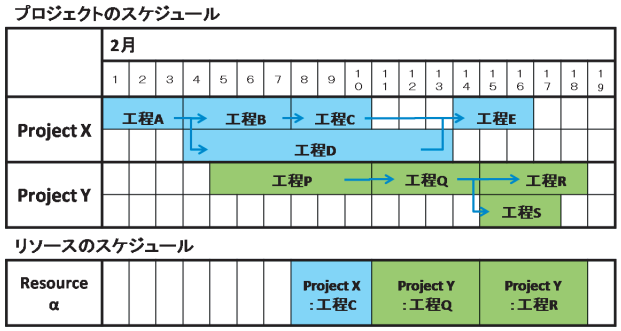


図 2　ボトルネック工程の例
Fig. 2　An example of bottleneck processes.

ロジェクト X のスケジュールだけを見ると，工程 C には 3 日間の余裕日があるため，3 日間の遅れまでなら，遅れた日数分だけ工程 C の開始日と終了日を遅らせることで調整できると読み取れてしまう．しかし，そのようにすると「リソースの割当て可能期間に関する制約」により，工程 C，Q，R に割り当てられているリソース α は，遅れた日数分だけ工程 Q の作業に着手するのが遅くなり，工程 Q のために 3 日間の作業日数を確保できなくなるので，工程 Q にはリソース α を割り当てることができない．そのため，リソース α を工程 C に割り当てるには，リソース α の空きスケジュールが 3 日以上連続する 19 日まで待たなければならない．このため，要員の追加，プロジェクト Y 工程 Q のリソース変更などを行わない限り，プロジェクトの終了日が大幅に延期され，プロジェクトは失敗してしまう．このため，工程 C はその作業日程を 1 日も動かすことができない．また，工程 C に割当て可能なリソースは α だけなので，工程 C では使用するリソースを取り替えることもできない．したがって，図 2 の例では，工程 C がボトルネック工程である．

プロジェクトを成功へ導くためには，たとえ 1 日の工程遅延でも，その影響がボトルネック工程に波及しないようにプロジェクトを進めていかなければならない．そのためには，ボトルネック工程（図 2 の例では工程 C）の直前の工程（図 2 の例では工程 B）の終了時までに，1 日の遅延も生じていない状態にしなければならない．

(2) 潜在的ボトルネック工程の定義とその事例

元来はボトルネック工程ではなかった工程が，工程遅延の影響を受け，その実施日が遅くなることによって，ボトルネック工程に変質する場合がある．そのような状況にある工程を潜在的ボトルネック工程と呼ぶ．

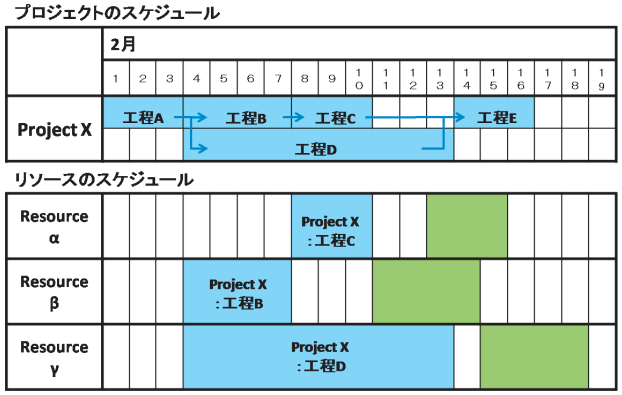


図 3　ボトルネック工程の例 2
Fig. 3　Another example of bottleneck processes.

図 3 を例にとって説明する．工程 B，C，D にリソース β，α，γ がそれぞれ割り当てられていて β，α，γ 以外に割当て可能なリソースは存在しないとする．また，工程 A，E に関しては，割り当てるリソースに何の制約もないので何日でも遅らせることが可能であるとする．

図 3 において，工程 C に割当て可能なリソースは α しか存在しない．しかも，リソース α の制約により，工程 C は最大で 2 日間までしか遅らせることができない．このため，2 日間の工程遅延が工程 C に及ぶと，工程 C はその作業日程を動かすことも，使用するリソースを取り替えることもできなくなる．つまり，2 日間の工程遅延で工程 C はボトルネック工程に変質する．

工程 B に割当て可能なリソースは β しか存在しない．しかも，リソース β の制約により，工程 B は最大 3 日間まで遅らせることが可能であるように見える．しかし，工程 B の後続工程 C が 2 日間の工程遅延でボトルネック工程に変質するので，2 日間の工程遅延で工程 B はその作業日程を変更することも，使用するリソースを取り替えることもできなくなる．つまり，2 日間の工程遅延で工程 B はボトルネック工程に変質する．

工程 A の後続工程は工程 B と工程 D の 2 つである．工程 A の後続工程 B が 2 日間の工程遅延でボトルネック工程に変質するので，工程 A–工程 B–工程 C–工程 E のパスでは，工程 A は最大で 2 日間の工程遅延しか許されないことが分かる．一方，工程 A–工程 D を調べてみると，工程 A は割り当てるリソースに何の制約もないのでボトルネック工程には

なりえない．これに対して，工程 D に割当て可能なリソースは γ しか存在しない．しかも，リソース γ の制約により，工程 D は 1 日しか遅らせることはできない．このため，1 日の工程遅延で工程 D はその作業日程を動かすことも，使用するリソースを取り替えることもできなくなる．つまり，1 日の工程遅延で工程 D はボトルネック工程に変質する．このため，工程 A–工程 D のパスでは，工程 D の先行工程 A は 1 日の工程遅延しか許されないことが分かる．両者を比較することにより，工程 A は 1 日の工程遅延しか許されないと結論される．

以上の考察により，図 3 の例では，潜在的ボトルネック工程は工程 B，工程 C，工程 D の 3 つであり，工程 B は 2 日間の工程遅延で，工程 C は 2 日間の工程遅延で，工程 D は 1 日の工程遅延でそれぞれボトルネック工程に変質する．

5. 提案する工程遅延の影響波及解析方法

本稿で提案する（工程遅延の影響波及の）解析方法は，「実プロジェクトで起こりうる種々の工程遅延が実際に起こったと仮定して，仮の工程遅延を与えることにより，仮に与えた工程遅延が後続工程にどのような影響を及ぼすかをシミュレートする」ものである．

影響波及解析の手順を，**図 4** を用いて説明する．なお，下記の番号は図 4 の番号と対応している．

① 工程遅延が発生した工程の終了日を変更する．
② 工程遅延が発生した工程の後続工程を「作業順序に関する制約」に基づいて検索する．
③ 検索の結果から後続工程の有無をチェックする．
④ すべての後続工程で，開始日の変更が必要な工程の開始日をすべて変更する．
⑤ 変更された開始日より「リソースの割当て可能期間に関する制約」に基づき，要員のスケジュールを変更する．ただし，解析対象のプロジェクト以外の予定は変更しない．もし，その要員に他のプロジェクトの予定が入っている場合，その予定が終了した後工程に，作業を割り当てる．
⑥ 要員の再割当てを行った結果，遅延した日数よりも遅れが大きいかを確認する．
⑦ 遅れが大きい工程の場合，調整が必要となる工程である可能性が高いため，その工程の情報を保存しておく．② から ⑦ を繰り返し，後続工程がなくなった（プロジェクトが終了した）時点で，影響波及解析を終了する．

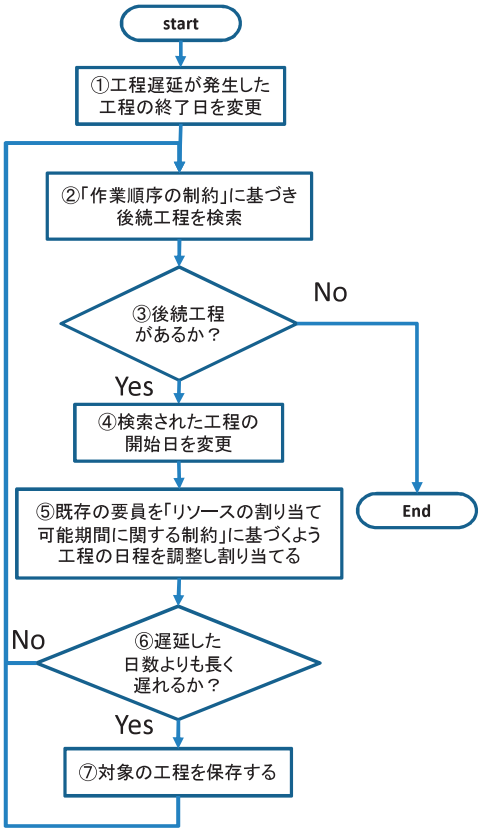


図 4　影響波及解析の流れ
Fig. 4　A flowchart of repercussion analysis.

6. 提案した解析方法の適用例

先にあげた図 3 の例を基に，解析方法（シミュレーションの方法）の適用例を以下に示す．

(1) 最初の遅延が工程 A に発生した場合

工程 A に最初の遅延が発生したと仮定して，想定した日数の遅延を工程 A に与える．1 回に与える摂動（遅延の日数）は，1 回目は 1 日で 2 回目は 2 日と，1 回ごとに 1 日刻みで

増えてゆき，最終回は 5 日（1 週間）である．摂動の日数を最大で 1 週間としたのは，ソフトウェア開発プロジェクトで行われる進捗管理が 1 週間に 1 回であることが多いからである．

(i) 工程 A に与えた遅延日数が 1 日の場合

この場合，（工程 A に割り当てられたリソースには，割当て可能期間に関する制約がないので）工程 A の終了日は 1 日だけ延びて，2 月 4 日となる．工程 A の終了日が 1 日延びたことにより，工程 A の後続工程である工程 B と D の遂行期間も．それぞれ 1 日延びる．工程 B と D の遂行期間が 1 日延びても，工程 B と D に割り当てられたリソース β と γ の割当て可能期間に関する制約には抵触しないので，工程 B の遂行期間は 2 月 5 日〜8 日，工程 D の遂行期間は 2 月 9 日〜14 日となる．

工程 B の終了日が 1 日延びたことにより，工程 B の後続工程 C の遂行期間は早くても 2 月 9 日〜11 日となる．工程 C の終了日が 1 日延びても，（工程 C に割り当てられたリソース α の割当て可能期間に関する制約には抵触しないので）工程 C の遂行期間は 2 月 9 日〜11 日となる．

工程 C と工程 D の両方の後続工程である工程 E は，工程 C と工程 D の遅延の影響を受けるが，工程 E に割り当てられたリソースには割当て可能期間に関する制約がないので，その遂行時期は 1 日延びるだけである．このため，工程 E の遂行期間は 2 月 15 日〜17 日となる．

(ii) 工程 A に与えた遅延日数が 2 日の場合

この場合，工程 A に与えた遅延日数が 1 日の場合と同様に考えれば，各工程の開始日と終了日が決定する．その結果，工程 A は終了日が 2 日だけ延びて 2 月 5 日となる．工程 A の遅延の影響を受けて，工程 B はその遂行期間が 2 月 6 日〜9 日となる．工程 B の遅延の影響を受けて，工程 C の遂行期間は 2 月 10 日〜12 日となる．

工程 A の遅延の影響を受けて，工程 D の遂行期間は早くても 2 月 6 日〜15 日となるが，工程 D に割り当てられたリソース γ には，2 月 15 日〜18 日に別プロジェクトの予定が入っているので，この時期に γ を割り当てることはできない．そのうえ，工程 D に割当て可能なリソースは γ だけなので，工程 D の遂行期間は γ の予定が空く 2 月 19 日〜28 日となる．このため，この段階になって，前の段階で（つまり，工程 A の遅延が 1 日だったときに），工程 D はすでに潜在的なボトルネック工程になっていたことが分かる．

工程 C の遅延の影響を受けて，工程 E の遂行期間は早くても 2 月 13 日〜15 日となり，工程 D の遅延の影響を受けて早くても 3 月 1 日〜3 日となる．しかし，工程 E に割り当てられたリソースには割当て可能期間に関する制約がないので，結論として工程 E の遂行期間は 3 月 1 日〜3 日となる．

(iii) 工程 A に与えた遅延日数が 3 日の場合

この場合，工程 A に与えた遅延日数が 1 日の場合と同様に考えれば，各工程の開始日と終了日が決定する．その結果，工程 A は終了日が 3 日だけ延びて 2 月 6 日となる．工程 A の遅延の影響を受けて，工程 B の遂行期間は 2 月 7 日〜10 日となる．工程 B の遅延の影響を受けて，工程 C の遂行期間は早くても 2 月 11 日〜13 日となる．ところが，工程 C に割り当てられていたリソース α には，2 月 13 日〜15 日に別プロジェクトの予定が入っているので，この時期に α を割り当てることはできない．そのうえ，工程 C に割当て可能なリソースは α だけなので，工程 C の遂行期間は α の予定が空く 2 月 16 日〜18 日となる．このため，この段階になって，前の段階で（つまり，工程 A の遅延が 2 日だったときに），工程 C はすでに潜在的なボトルネック工程になっていたことが分かる．

工程 A の遅延の影響を受けて，工程 D の遂行期間は早くても 2 月 7 日〜16 日となる．ところが，工程 D に割り当てられたリソース γ には，2 月 15 日〜18 日に別プロジェクトの予定が入っているので，この時期に γ を割り当てることができない．そのうえ，工程 D に割当て可能なリソースは γ だけなので，工程 D の遂行期間は γ の予定が空く 2 月 19 日〜28 日となる．このため，この段階になって，前の段階で（つまり，工程 A の遅延が 1 日だったときに），工程 D はすでに潜在的なボトルネック工程になっていたことが分かる．

工程 C の遅延の影響を受けて，工程 E の遂行期間は早くても 2 月 19 日〜21 日となり，工程 D の遅延の影響を受けて早くても 3 月 1 日〜3 日となる．しかし，工程 E に割り当てられたリソースには割当て可能期間に関する制約がないので，結論として工程 E の遂行期間は 3 月 1 日〜3 日となる．

(iv) 工程 A に与えた遅延日数が 4 日の場合

この場合，工程 A に与えた遅延日数が 1 日の場合と同様に考えれば，各工程の開始日と終了日が決定する．その結果，工程 A は終了日が 4 日だけ延びて 2 月 7 日となる．工程 A の遅延の影響を受けて，工程 B の遂行期間は 2 月 8 日〜11 日となる．ところが，工程 B に割り当てられたリソース β には，2 月 11 日〜14 日に別プロジェクトの予定が入っているので，この時期に β を割り当てることができない．そのうえ，工程 B に割当て可能なリソースは β だけなので，工程 B の遂行期間は β の予定が空く 2 月 15 日〜18 日となる．このため，この段階になって，前の段階で（つまり，工程 A の遅延が 3 日だったときに），工程 B はすでに潜在的なボトルネック工程になっていたことが分かる．工程 B の遅延の影響を受

けて，工程 C の遂行期間は 2 月 19 日〜21 日となる．

工程 A の遅延の影響を受けて，工程 D の遂行期間は早くても 2 月 8 日〜17 日となる．ところが，工程 D に割り当てられたリソース γ には，2 月 15 日〜18 日に別プロジェクトの予定が入っているので，この時期に γ を割り当てることができない．そのうえ，工程 D に割当て可能なリソースは γ だけなので，工程 D の遂行期間は γ の予定が空く 2 月 19 日〜28 日となる．

工程 C の遅延の影響を受けて，工程 E の遂行期間は早くても 2 月 22 日〜24 日となり，工程 D の遅延の影響を受けて早くても 3 月 1 日〜3 日となる．しかし，工程 E に割り当てられたリソースには割当て可能期間に関する制約がないので，結論として工程 E の遂行期間は 3 月 1 日〜3 日となる．

(v) 工程 A に与えた遅延日数が 5 日の場合

この場合，工程 A に与えた遅延日数が 1 日の場合と同様に考えれば，各工程の開始日と終了日が決定する．その結果，工程 A は終了日が 5 日だけ延びて 2 月 8 日となる．工程 A の遅延の影響を受けて，工程 B の遂行期間は 2 月 9 日〜12 日となる．ところが，工程 B に割り当てられたリソース β には，2 月 11 日〜14 日に別プロジェクトの予定が入っているので，この時期に β を割り当てることができない．そのうえ，工程 B に割当て可能なリソースは β だけなので，工程 B の遂行期間は β の予定が空く 2 月 15 日〜18 日となる．このため，この段階になって，前の段階で（つまり，工程 A の遅延が 3 日だったときに），工程 B はすでに潜在的なボトルネック工程になっていたことが分かる．工程 B の遅延の影響を受けて，工程 C の遂行期間は 2 月 19 日〜21 日となる．

工程 A の遅延の影響を受けて，工程 D の遂行期間は早くても 2 月 9 日〜18 日となる．ところが，工程 D に割り当てられていたリソース γ には，2 月 15 日〜18 日に別プロジェクトの予定が入っているので，この時期に γ を割り当てることができない．そのうえ，工程 D に割当て可能なリソースは γ だけなので，工程 D の遂行期間は γ の予定が空く 2 月 19 日〜28 日となる．

工程 C の遅延の影響を受けて，工程 E の遂行期間は早くても 2 月 22 日〜24 日となり，工程 D の遅延の影響を受けて早くても 3 月 1 日〜3 日となる．しかし，工程 E に割り当てられたリソースには割当て可能期間に関する制約がないので，結論として工程 E の遂行期間は 3 月 1 日〜3 日となる．

(2) 最初の遅延が工程 B に発生した場合

この場合，工程 A には遅延がないので，工程 A の後続工程である工程 D の遂行期間は当初の予定どおり 2 月 4 日〜13 日となる．

(i) 工程 B に与えた遅延日数が 1 日の場合

この場合，工程 B の終了日は 1 日延びて 2 月 8 日となる．工程 B の後続工程 C の遂行期間も 1 日延びて 2 月 9 日〜11 日となる．工程 E の遂行期間は予定どおりの 2 月 14 日〜16 日となる．

(ii) 工程 B に与えた遅延日数が 2 日の場合

この場合，工程 B の終了日は 2 日延びて 2 月 9 日となる．工程 B の後続工程 C の遂行期間も 2 日延びて 2 月 10 日〜12 日となる．工程 E の遂行期間は予定どおりの 2 月 14 日〜16 日となる．

(iii) 工程 B に与えた遅延日数が 3 日の場合

この場合，工程 B の終了日は 3 日延びて 2 月 10 日となる．このため，工程 C の遂行期間は早くても 2 月 11 日〜13 日となる．ところが，工程 C に割り当てられていたリソース α には，2 月 13 日〜15 日に別プロジェクトの予定が入っているので，工程 C の遂行期間は α の予定が空く 2 月 16 日〜18 日となる．このため，この段階になって，前の段階で（つまり，工程 A の遅延が 2 日だったときに），工程 C はすでに潜在的なボトルネック工程になっていたことが分かる．

工程 A には遅延がないので，工程 C の遅延の影響のみを受けて，工程 E の遂行期間は早くても 2 月 19 日〜21 日となる．

(iv) 工程 B に与えた遅延日数が 4 日の場合

この場合，工程 B の終了日は 4 日延びて 2 月 11 日となる．このため，工程 C の遂行期間は早くても 2 月 12 日〜14 日となる．ところが，工程 C に割り当てられていたリソース α には，2 月 13 日〜15 日に別プロジェクトの予定が入っているので，工程 C の遂行期間は α の予定が空く 2 月 16 日〜18 日となる．

工程 A には遅延がないので，工程 C の遅延の影響のみを受けて，工程 E の遂行期間は 2 月 19 日〜21 日となる．

(v) 工程 B に与えた遅延日数が 5 日の場合

この場合，工程 B の終了日は 5 日延びて 2 月 12 日となる．このため，工程 C の遂行期間は早くても 2 月 13 日〜15 日となる．ところが，工程 C に割り当てられていたリソース α には，2 月 13 日〜15 日に別プロジェクトの予定が入っているので，工程 C の遂行期間は α の予定が空く 2 月 16 日〜18 日となる．

工程 A には遅延がないので，工程 C の遅延の影響のみを受けて，工程 E の遂行期間は 2

月 19 日〜21 日となる．

(3) 最初の遅延が工程 C に発生した場合

この場合，工程 A と工程 B に遅延がないので，工程 B の後続工程である工程 C の遂行期間は当初の予定どおり 2 月 8 日〜10 日となる．工程 A に遅延がないので，工程 A の後続工程である工程 E の遂行期間は当初の予定どおり 2 月 4 日〜13 日となる．

(i) 工程 C に与えた遅延日数が 1〜3 日の場合

この場合，工程 C の終了日は，遅延が 1 日のときに 2 月 11 日，2 日のときに 2 月 12 日，3 日のときに 2 月 13 日となる．しかし，工程 C には 3 日間の余裕があるので，工程 C の後続工程である工程 E にはまったく影響がなく，工程 E の遂行期間は当初の予定どおり 2 月 4 日〜13 日となる．

(ii) 工程 C に与えた遅延日数が 4〜5 日の場合

この場合，工程 C の終了日は，遅延が 4 日のときに 2 月 14 日，5 日のときに 2 月 15 日となる．ところが，工程 C に割り当てられていたリソース α には，2 月 13 日〜15 日に別プロジェクトの予定が入っているので，工程 C の遂行期間は α の予定が空く 2 月 16 日〜18 日となる．

工程 A と B に遅延がないので，工程 C のみの遅延の影響を受けて，工程 E の遂行期間は 2 月 19 日〜21 日となる．

(4) 最初の遅延が工程 D に発生した場合

この場合，工程 A，B，C のいずれにも遅延がないので，これら工程の遂行期間はいずれも予定どおりとなる．

(i) 工程 D に与えた遅延日数が 1 日の場合

工程 D の終了日は 1 日延びて 2 月 14 日となる．工程 D の後続工程 E は工程 D の遅延の影響を受けて，その遂行期間は 2 月 15 日〜17 日となる．

(ii) 工程 D に与えた遅延日数が 2〜5 日の場合

この場合，工程 D の終了日は，遅延が 1 日のときに 2 月 14 日，2 日のときに 2 月 15 日，3 日のときに 2 月 16 日，4 日のときに 2 月 17 日，5 日のときに 2 月 18 日となる．ところが，工程 D に割り当てられていたリソース γ には，2 月 14 日〜18 日に別プロジェクトの予定が入っているので，そのプロジェクトにおいて γ が割り当てられていた工程の開始日を遅らせなければならなくなるので，そのプロジェクトに迷惑をかけることになる．このため，この段階になって，前（つまり，工程 D の遅延が 1 日のみ）の段階で，工程 D はすでにボトルネック工程になっていたがことが分かる．

工程 D の後続工程である工程 E には割当て可能期間にまったく制約がないので，工程 D が遅延した日数分だけ工程 E の遂行期間が遅くなる．

(5) 最初の遅延が工程 E に発生した場合

この場合，工程 A，B，C，D のいずれにも遅延がないので，これら工程の遂行期間はいずれも予定どおりとなる．

工程 E に割り当てられたリソースには割当て可能期間に関する制約がないので，工程 E に遅延が発生しても，遅延日数分の遅延にしかならない．つまり，割当て可能期間に関する制約に基づく遅延日数の増幅はない．

7. 適用事例に基づく提案手法の有効性の議論

6 章では，摂動を用いた影響波及解析の方法（工程遅延が後続工程に及ぼす影響を解析する方法）を図 3 の例に適用した事例を示した．その解析結果を**表 1** に示す．

我々は，摂動を用いた影響波及解析により，与えた摂動（工程遅延の日数）に対して，各工程の遅延がそれぞれ何日になるかを求めた．その結果は表 1 のとおりである．

(1) 与えた摂動の日数よりも遅延が大きくなっている箇所は次の 2 つであることが判明した．

・工程 A の遅延が 3〜5 日の場合
　工程 C が 8 日の遅延，工程 E が 5 日の遅延
・工程 B の遅延が 3〜5 日の場合
　工程 C が 8 日の遅延，工程 E が 12 日の遅延

(2) 与えた摂動の日数よりも遅延が大きくなっている箇所は，与えた摂動の日数よりも少ない遅延のときにすでにボトルネックになっていることが判明した．具体的には次の 2 カ所である．

・工程 A の遅延が 2 日の時点で工程 C がボトルネック状態になっている．
・工程 B の遅延が 2 日の時点で工程 C がボトルネック状態になっている．

(3) 本稿では，摂動を 1 日刻みで与えたが，1 日刻みでなくても，ボトルネック状態にある工程を検出できることが多い．たとえば，摂動を 2 日刻みで与えても，(1) と (2) の結果が得られる．

(4) 本稿で提案した解析方法を使えば，どの工程が何日の遅延でボトルネック状態になるかということ（潜在的ボトルネック工程の詳細）を調べることが可能となる．

(5) 工程遅延が発生する前に，潜在的ボトルネック工程の詳細を知ることができれば，潜

表 1　摂動を用いた影響波及解析の結果
Table 1　The results of perturbation-based repercussion analysis.

与えた摂動		工程 A	工程 B	工程 C	工程 D	工程 E
工程 A 3 日間	1 日	2/1－2/4,遅延 1 日	2/5－2/8，遅延 1 日 問題なし	2/9－2/11，遅延 1 日 問題なし	2/5－2/14，遅延 1 日 ボトルネック	2/15－2/17，遅延 1 日
	2 日	2/1－2/5,遅延 2 日	2/6－2/9，遅延 2 日 問題なし	2/10－2/12，遅延 2 日 ボトルネック	2/19－2/28，遅延 15 日	3/1－3/3，遅延 15 日
	3 日	2/1－2/6,遅延 3 日	2/7－2/10，遅延 3 日 ボトルネック	2/16－2/18，遅延 8 日	2/19－2/28，遅延 15 日	3/1－3/3，遅延 15 日
	4 日	2/1－2/7,遅延 4 日	2/15－2/18，遅延 11 日	2/19－2/21，遅延 11 日	2/19－2/28，遅延 15 日	3/1－3/3，遅延 15 日
	5 日	2/1－2/8,遅延 5 日	2/15－2/18，遅延 11 日	2/19－2/21，遅延 11 日	2/19－2/28，遅延 15 日	3/1－3/3，遅延 15 日
工程 B 4 日間	1 日	2/2－2/3,遅延なし	2/4－2/8，遅延 1 日 問題なし	2/9－2/11，遅延 1 日，問題なし	2/4－2/13，遅延なし	2/14－2/16，遅延なし
	2 日	2/2－2/3,遅延なし	2/4－2/9，遅延 2 日 問題なし	2/10－2/12，遅延 2 日 ボトルネック	2/4－2/13，遅延なし	2/14－2/16，遅延なし
	3 日	2/2－2/3,遅延なし	2/4－2/10，遅延 3 日 ボトルネック	2/16－2/18，遅延 8 日	2/4－2/13，遅延なし	2/19－2/21，遅延 5 日
	4 日	2/2－2/3,遅延なし	2/15－2/18，遅延 11 日	2/19－2/21，遅延 11 日	2/4－2/13，遅延なし	2/22－2/24，遅延 8 日
	5 日	2/2－2/3,遅延なし	2/15－2/18，遅延 11 日	2/19－2/21，遅延 11 日	2/4－2/13，遅延なし	2/22－2/24，遅延 8 日
工程 C 3 日間	1 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/11，遅延 1 日 問題なし	2/4－2/13，遅延なし	2/14－2/16，遅延なし
	2 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/12，遅延 2 日 ボトルネック	2/4－2/13，遅延なし	2/14－2/16，遅延なし
	3 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/16－2/18，遅延 8 日	2/4－2/13，遅延なし	2/19－2/21，遅延 5 日
	4 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/16－2/18，遅延 8 日	2/4－2/13，遅延なし	2/19－2/21，遅延 5 日
	5 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/16－2/18，遅延 8 日	2/4－2/13，遅延なし	2/19－2/21，遅延 5 日
工程 D 10 日間	1 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/4－2/14，遅延 1 日 ボトルネック	2/15－2/17，遅延 1 日
	2 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/19－2/28，遅延 15 日	3/1－3/3，遅延 15 日
	3 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/19－2/28，遅延 15 日	3/1－3/3，遅延 15 日
	4 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/19－2/28，遅延 15 日	3/1－3/3，遅延 15 日
	5 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/19－2/28，遅延 15 日	3/1－3/3，遅延 15 日
工程 E 3 日間	1 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/4－2/13，遅延なし	2/14－2/17，遅延 1 日
	2 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/4－2/13，遅延なし	2/14－2/18，遅延 2 日
	3 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/4－2/13，遅延なし	2/14－2/19，遅延 3 日
	4 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/4－2/13，遅延なし	2/14－2/20，遅延 4 日
	5 日	2/2－2/3,遅延なし	2/4－2/7，遅延なし	2/8－2/10，遅延なし	2/4－2/13，遅延なし	2/14－2/21，遅延 5 日

在的ボトルネック工程に遅延が及ばないようにするための事前対策を立てることが可能となる．

(6) 実際に遅延が発生し，その遅延がボトルネック状態にある工程へ波及する可能性があると判断できた場合には，ボトルネック状態にある工程へ遅延が波及する前に対策を講じ，工程遅延を解消しなければならない．そのための対策としては以下の 3 案がある．

(i) ボトルネック状態にある工程の前工程に対してクラッシングを施す．
(ii) ボトルネック状態にある工程そのものにクラッシングする．
(iii) ボトルネック状態にある工程が使用するリソースを，別プロジェクトの開発計画では使用しないように変更する．

ここで，クラッシングとは，割当て可能なリソースを数多く割り当てることである．我々は，工程遅延が発生したとき，(i) および (ii) により，工程遅延を回復できるような計画案

を自動生成する機能をすでに実装するとともに，論文発表をしている[2)]．

しかし，(iii) は競合する複数のプロジェクトにおいて，どのプロジェクトを優先してクラッシングするかという問題を含んでいるので，人間の判断が必要となるので，完全な自動対策は不可能である．

なお，本稿の例では，説明の都合上簡略化した事例を用いている部分がある．たとえば，本稿では土曜日・日曜日・祝日といった休日を考慮していないため，実際にそれらを考慮に入れた場合，5 日の遅延が 1 週間の遅延となり，より大きな遅延に感じられることになる．

また，本稿で採り上げた事例は，1 つのリソースが 1 度に複数のプロジェクトや作業を兼務している状況が実際にはありえないほど極端であるが，本稿で提案した手法の有効性を示すために用いた．問題の質を変更していないので，そのような事例でも本稿で提案した手法の有効性を示すうえでは有効であると考えている，

8. おわりに

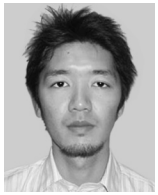
本稿の提案では，プロジェクトに種々の摂動を仮に与えることにより，その影響がどのように波及していくのかシミュレートする影響波及解析を用いることで，ボトルネック工程を検出することが可能になった．また，従来工程遅延発生後に用いられた影響波及解析を，発生前の段階で用いることで，pro-active な対策が行えることを可能にした．

本稿では，プロジェクトに存在するボトルネック工程とその問題点を述べた．そして，影響波及解析を行うことにより，ボトルネック工程を自動検出する手法と，その有効性について述べた．さらに，検出されたボトルネック工程よりも前の工程にクラッシングを適用することにより，ボトルネック工程に遅延を波及しないようにできることを述べた．

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) Hazeyama, A. and Komiya, S.: Workload Management Facilities for Software Project Management, *IEICE Trans. Inf. Syst.*, Vol.E81-D, No.12, pp.1404–1414 (1998).
4) 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 (2004).
5) 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).
6) Leach, L.P.: *Critical Chain Project Management*, Artech House Professional Development Library, London 313 (2000).
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) Liu, L. and Horowitz, E.: A formal model for software project management, *IEEE Trans. Softw. Eng.*, Vol.15, No.10, pp.1280–1293 (1989).
9) 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).
10) 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).
11) Sato, H.: Project management expert system, *Proc. ACM CSC'87* (Feb. 1987).
12) 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 (2004).
13) 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).
14) エリヤフ・ゴールドラット：ザ・ゴール，ダイヤモンド社 (2001).
15) エリヤフ・ゴールドラット：ザ・ゴール 2，ダイヤモンド社 (2002).
16) 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).
17) 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).
18) 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).
19) Kinoshita, D., Yaegashi, R., Uenosono, K., Hashiura, H., Uchikawa, H. and Komiya, S.: Automatic Creation of a Crashing-Based Schedule Plan as Countermeasures against Process Delay, *International Journal of Systems Applications, Engineering & Development*, Vol.2, Issue 4, pp.170–177 (2008).

(平成 21 年 4 月 3 日受付)
(平成 21 年 7 月 2 日採録)

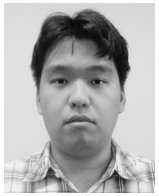


木下　大輔

平成 14 年芝浦工業大学工学部工業経営学科卒業．平成 16 年同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，現職．現在，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．

内川　裕貴

平成 19 年芝浦工業大学工学部情報工学科卒業．平成 21 年同大学大学院工学研究科修士課程修了．平成 21 年株式会社日立製作所入社，現職．



小坂　祐也

平成 21 年芝浦工業大学工学部情報工学科卒業．現在，同大学大学院工学研究科修士課程在学中．

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〜平成 12 年 IPA 技術センター特別研究員．平成 3〜9 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 6〜9 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．