Vol. 36 No. 5　　情　報　処　理　　May 1995

解　説

ソフトウェアプロセス

1. ソフトウェアプロセスに関する研究の概要†

落水　浩一郎††

1. はじめに

ソフトウェアプロセスに関する研究は，ソフトウェアプロジェクトの設計・管理に理論的，技術的基盤を与えることを目標とする．すなわち，ソフトウェアプロセスモデルは，これまで個別に研究されてきた，開発パラダイムや開発方法論，CASEツールとソフトウェア開発環境，資源管理や品質管理等のソフトウェア工学の種々の成果を前提として，ソフトウェア開発のスタイルに応じた適切な要素技術を選択・統合するための枠組を与える．

1980年代の半ばから始まったソフトウェアプロセスに関する研究は，個人のスキルや組織やプロジェクトの管理運営法等の人間系に関する話題も研究対象に含める必要があり，この10年間にわたり研究成果の確実な積み上げはあったものの，現状では解決すべき課題はまだ多い．一方，最近，標準化の動きがあり，ソフトウェアプロセスに関する研究の現状を把握しておくことは重要である．

本稿は，ソフトウェアプロセスに関する研究の起源，発展の経過，主な成果，今後の課題等をまとめつつ，ソフトウェアプロセスに関する研究の全体像を把握することを目的とする．

上記のような立場から，**2.**では，ソフトウェアプロセスに関する内外の研究コミュニティによる研究活動の概要を紹介する．**3.**では二つの主要な研究の流れと成果を説明する．一つは**プロセス記述言語とその実行環境**の開発であり，もう一つは**組織の評価手段としてのプロセス成熟度モデル**に関する研究である．特に，ソフトウェアプロセス国際ワークショップの活動経過を紹介しつつ，研究の起源と発展の経過，主な成果と今後の展望を明らかにする．**4.**では，上記活動の主な成果である，「統合環境による柔軟な標準化手段の提供」と「ツール統合による作業の連続性の保持」をとりあげ現状を紹介する．最後に，**5.**において，本分野における研究課題を列挙するとともに，今後の課題をまとめる．

2. ソフトウェアプロセスに関する内外の研究活動

ソフトウェアプロセスの研究に関する内外の研究コミュニティの活動を紹介する．

（1） ソフトウェアプロセス国際ワークショップ（ISPW*）

ソフトウェアプロセスの研究に関して牽引車の役割を果たしてきた会議である．1984年から現在まで，8回の会議が開催され[1)~8)]，**3.1.**に述べるように，ソフトウェアプロセスに関する諸概念，諸技術を提案・整備してきた．第9回は1994年10月に米国で開催される予定である．

（2） ソフトウェア工学国際会議（ICSE**）

第3回ISPWの直後に行われた第9回ICSE（1987年）からソフトウェアプロセスはICSEの主要テーマの一つとなった．あとに述べる「プロセス記述言語とその実行環境」，「プロセス成熟度モデル」，「プロセスモデルに基づく統合環境」等が主要な成果である．

（3） ソフトウェアプロセス国際会議（ICSP*3）

ISPWは少人数の参加者に限定されたディベートの場であるが，研究者の増加につれて，研究

† Survey of Research Activities on Software Process by Koichiro OCHIMIZU (School of Information Science, Japan Advanced Institute of Science and Technologies).
†† 北陸先端科学技術大学院大学情報科学研究科

* International Software Process Workshop
** International Conference on Software Engineering
*3 International Conference on the Software Process

成果の公式な発表の場として，1991年に国際会議が開始された．特に，1993年に実施された第2回ICSP[9]ではCMMとISOに関する標準化のパネルがあり賛否両論の反応があった．第2回ICSPに関しては井上による報告がある[10]．第3回は1994年10月に米国で開催される予定である．

（4） ソフトウェアプロセス技術に関するヨーロピアンワークショップ（EWSPT*）

第1回は1991年5月にイタリアで，第2回は1992年9月にノルウェイ，第3回は1994年2月にフランスで開催された．たとえば第2回の会議では，人間や人間集団の行動を記述し規定することの困難さ，プロセスダイナミズム，様々なプロセス記述方式の提案と比較等が主な話題となっている[11]．ESF（Eureka Software Factoryプロジェクト）やEspritプロジェクトを母体として活発な研究活動が展開されている．

（5） ソフトウェア技術者協会（SEA）の活動

日本では，SEAによる啓蒙，調査活動が活発であった．たとえば，以下に紹介する機関誌SEAMAILの記事内容は日本におけるソフトウェアプロセスに関する関心の推移を表しており興味深い．

1987年 プロセスモデル発生の経緯とプロセスプログラミングの紹介[12),13)]

1989年 第1回ソフトウェアプロセスワークショップ（JSPW）報告[14)]

1990年 第5回ISPW報告[15)]，第6回ISPW報告[16)]

1991年 ソフトウェアプロセスに関する調査（第1次集計結果）報告…500通の回答をもとにまとめられた，企業におけるソフトウェアプロセスに関する方針やガイドラインおよび問題点に関する調査結果の報告，および成熟度モデル（後述）との関連に関する考察[17)]

1993年 ISO 9000-3の内容の紹介…ISO 9000-3がハードウェア指向であることの問題点の指摘[18)]，ISO 9001およびISO 9000-3の欠陥の要約[19)]

（6） 日本ソフトウェアプロセスワークショップ（JSPW）

日本ソフトウェア科学会ソフトウェアプロセス研究会，ソフトウェア技術者協会，情報処理学会の共催/協賛で都合4回開催され，日本におけるソフトウェアプロセスの現状等が検討された．特に，第4回JSPWにおいて望月によって提案された「組織の規模に応じたソフトウェアプロセスの役割」をスキー競技の旗門設計に例えた解釈は，落水によって第2回ICSPのパネルセッションで報告され参加者の賛同を得た．すなわち，大規模組織におけるプロセスモデルの役割を滑降競技における旗門配置，小規模組織におけるソフトウェアプロセスの役割を回転競技における旗門配置に対応付ける解釈である．

（7） 日本ソフトウェア科学会ソフトウェアプロセス研究会

第4回ISPWを契機として片山によって設立され，JSPWを主催してきた．1993年にソフトウェアプロセスの基礎研究に関する研究者の層を厚くすることを目標にして第1回ソフトウェアプロセス研究会が開催された．1994年10月には日本ソフトウェア科学会第11回大会において，ミニワークショップ「プロジェクト管理におけるソフトウェアプロセスの役割」を開催予定である．

（8） 情報処理学会ソフトウェア工学研究会

定例の研究会で継続的に研究発表がなされてきた．また，1994年5月には標準化の問題を主要な話題としてソフトウェアプロセスシンポジウムが開催された[20)]．標準化の動きが顕著になるにつれ，情報処理学会の果たす役割は重要なものとなるであろう．

（9） SDA国際共同研究プロジェクト

ソフトウェアプロセス研究の黎明期に，プロセスモデルに基づく統合環境のプロトタイプ開発を目標として活動したSDA国際共同研究プロジェクトが果たした先駆的な役割も注目すべきである[21)]．

その他，各企業において，主に「成熟度のレベル」に関する調査研究が実施されてきた．

3. 二つの主要な研究の流れ

前章で紹介したISPW，ICSE，ICSPにおける討論，成果報告を概観すると，ソフトウェアプロセスに関する研究は次の二つの大きな流れを持つ．

プロセス記述言語と実行環境 ソフトウェアプ

* European Workshop on Software Process Technology

ロセスをツール統合やプロジェクト管理のツールとしてとらえる立場である．

開発体制の評価手段の整備 ソフトウェアプロセスを組織における開発体制の評価手段としてとらえる立場である．

3.1 プロセス記述言語とその実行環境

ISPWの活動成果を年代順に追うことで，ソフトウェアプロセスに関する研究の起源と成果の概要を明らかにする．

（1） 第1回，1984年

残念ながら第1回の議事録は筆者の手許にない．ISPWの活動目的は，岸田[12]によると「ウオータフォールモデルにかわる**新しい開発パラダイム**の模索，**ソフトウェア開発環境のアーキテクチャデザインの土台**としてのプロセスモデルの必要性，という二つの源流をもとにしてソフトウェアプロセスワークショップが開催され，…中略…スパイラルモデル（第2回），契約モデル（第3回），プロセスプログラミング（第3回）等が次々に提案された」とある．

（2） 第2回，ソフトウェアプロセスとソフトウェア開発環境，1985年

このワークショップにおいては次のような問いかけとそれを受けた討論がなされた．

（a） 従来のアプローチはソフトウェアプロセス/知識ベース/操作的仕様等のアプローチに比べてどのような点が良いのか．それを比較するためのフォーマルモデル（メタモデル）はあるのか．

（b） 現在のプログラミング環境は，将来のソフトウェア開発環境になり得るのか．

（c） 大成功をおさめたり，致命的な失敗をひきおこすソフトウェア開発方法論の特徴は何か．たとえば，構造化分析/設計法の利点と欠点（検証，変換の困難さ）が検討された．

（3） 第3回，ソフトウェアプロセスにおける反復，1986年

ソフトウェアプロセス記述に固有の新しい話題である，「ソフトウェア開発における，ゆり戻しやバックトラックの問題をどのように制御すべきか，またそれをどのように形成化して，プロセス記述言語の構文要素として定義するか」が検討された．

メタモデル，活動とオブジェクト間の依存関係，制御構造等の討議セッションを通じて，

（a） 「反復」とは「以前の決定（生成されたオブジェクト）にさかのぼってそれを変更すること」であり，

（b） 「依存関係」とは「様々な決定の間に存在する相互関係（活動）を定義することで仮説（生成されたオブジェクト）に含まれる矛盾を検出しやすくする」ことであり，

（c） 「制御構造」とは「意思決定および変更の過程を形式化して表現することにより，相互関係が意思決定過程に与える影響を明示的にすること」であるという合意が得られた．

1987年の第9回ICSEでL. Osterweilによって提案された，「ソフトウェアプロセスを記述することにより，開発工程全般にわたって，規範性と自動化の度合を高めうる」という**プロセスプログラミング**[22]の概念は「制御構造」のセッションで発表されたものである．

（4） 第4回，ソフトウェアプロセスの起動法の表現，1988年

実行可能なプロセスモデルとその起動法，実行環境等が話題になった．また規模（scale）の問題やソフトウェアプロセス記述の目的（メリット）についても種々の意見交換がなされた．

（5） 第5回，ソフトウェアプロセスモデルの利用経験，1989年

（a） プロセスモデルはソフトウェア開発や保守をガイドし制御するのにどの程度有効か．

（b） 少規模プロセスから大規模プロセスまでうまくカバーできるのか．

（c） 個人やチームの活動に課せられる制約を満たしていくためのポリシーをどのように表現するのか．

（d） 汎用プロセスモデルの主要なパラメータとカスタマイズの手段は．

（e） プロセスの動的変更や進化にどのように対応するのか．

等の話題が提供された．プロセスモデルの構文的/意味論的側面，プロセスモデルのためのオブジェクトベース，プロセスモデルの実行制御の側面等にわたって意見が交換された．

（6） 第6回，ソフトウェアプロセスの支援，1990年

プロセスモデルに基づく統合環境の要素技術が検討された．すなわち，プロセスモデルの主要な

パラメータは何か，人間系との関係をどのようにとらえるか，プロセス記述能力の比較研究，プロセスモデルに基づく統合環境のアーキテクチャはいかにあるべきか，その中におけるオブジェクト管理のありかたは，ソフトウェア開発における労力のかなりの部分を占める，チーム構成員の協調やコミュニケーションをどのようにモデル化できるのか等である．また，百花繚乱のプロセス記述言語と実行環境について，共通例題をもとに比較研究をする努力が開始された．この会議の主な成果である**共通例題**については付録に全文を掲載する．

（7） 第7回，ソフトウェアプロセスにおける協調とコミュニケーション，1991年

このワークショップの主題はグループウェアではない．各種のプロセス記述法は，規模，成熟度のレベル，対象分野に応じて，それぞれ長短を持つ．複数個のプロセス記述が必要であるという認識を前提として，部分的プロセス記述を統合したり，プロセス記述間の情報フローを定めたりする問題が検討された．主要な論点は以下のとおりであった．（1）適切なプロセスの選択法（2）共通性の括り出し法（3）規模に対応したプロセス記述のレベルとその間の結合法（4）分散開発環境への写像法，各種プロセス駆動法（状態駆動，ルールベース，トリガ駆動，手続き的）の長短の比較を，IBISモデルに基づいて議論を進め，また記録した，最初の討議セッションの報告は興味深い．プロセス記述の動的変更，ソフトウェアプロセスとそれを定義し利用する人間系との関係等も話題になった．

（8） 第8回，プロセス技術実践の現状，1933年

リサーチプロトタイプのレベルではあるが，プロセスモデルに基づく統合環境（たとえば，AP5, PROCESSWEAVER, Marvel, Merlin, Life* FLOW, MELMAC, Matisse/SMART等）のデモが行われた点に注目すべきである[23]．

一連のワークショップによって，プロセス記述の方針と形式化（第3回），プロセス記述の実行環境（第4回），プロセスモデルやプロセス記述を実世界に適用する際の問題点の洗い出し（第5回），各種プロセス記述の比較手段の提供（第6回），複数のプロセス記述（同一のプロセスモデルの複数のインスタンス，または，複数のプロセスモデルのインスタンス）間の通信の支援（第7回），プロセスモデルに基づく統合環境の実例の紹介と比較（第8回）のように，研究は着実に進展しつつある．しかし，実用レベルに到達するにはクリアされるべき課題はまだ多い．たとえば，以下の話題はワークショップで繰り返し討議されているが，現在のところ有効な解がなく今後の発展が望まれる問題である．

1. **規模の問題**をクリアする必要がある．
2. 単一のプロセス記述ではなく，**部分的記述とその起動，統合の手段**が問題である．
3. **事例研究**が必要である．
4. **人間系との関係**についてもう少し切り込みが必要である．

3.2 組織の評価手段としてのプロセス成熟度モデル

品質の高いソフトウェアを経済的に作成するためには，組織のソフトウェアプロセスを日常的に改善することが必要であるという立場から，W. S. Humphreyは**プロセス成熟度のレベル**を形式化した[24]．現在の組織のレベルをアセスメントにより認識した上で，段階的に上位の状態に移行することを推奨している（**図-1**）．

図-1において，特に重要なのは，「定義されている」状態である．「反復可能状態」から「定義された状態」に成熟度のレベルをあげるためには，

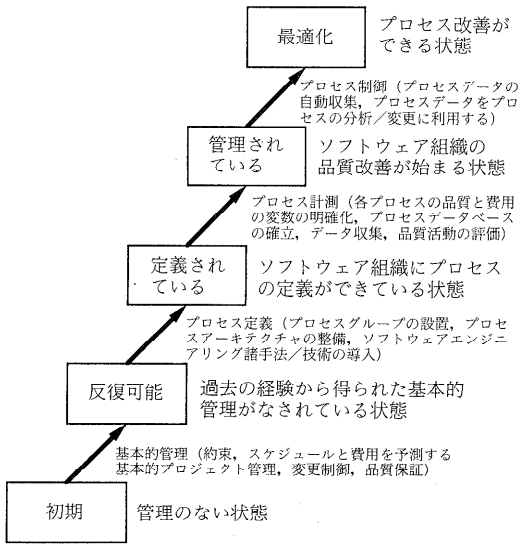

図-1 プロセス成熟度のレベル

プロセスアーキテクチャを整備する必要がある．ここでプロセスアーキテクチャは以下のように定義される[24]．(1)**ソフトウェアプロセス**とは，ソフトウェアの生産および進化の過程で利用されている活動，手法，および慣習の集合である．(2)**ソフトウェアプロセスアーキテクチャ**とは，プロジェクト固有のソフトウェアプロセスを定義するための枠組みである．(3)ソフトウェアプロセスアーキテクチャをある特定の目的をもって具体化したものを**ソフトウェアプロセスモデル**とよぶ．

「定義されている」状態は，プロセスを吟味し改善策を決定するための基礎固めができた状態である．「プロセス記述言語とその実行環境」と「組織の評価手段としてのプロセス成熟度モデル」は，このレベルにおいて接点を持つ．すなわち，プロセスアーキテクチャを形式的に記述・実行する手段を与えるのが「プロセス記述言語とその実行環境」に関する研究の目的である．

最近，ソフトウェアプロセスに関する標準化の動きが顕著であり，ISO 9000-3 と SPICE (Software Preocess Improvement Capability dEtermination) が注目を集めている．ISO 9000-3 は，ISO/TC 176（ハードウェアの品質システムの専門家）が作成した ISO 9001 のソフトウェア版であり，いろいろな問題点が指摘されている[18]．SPICE は ISO/IEC JTC 1/SC 7 の WG 10 によって 1992 年から標準化の努力が開始されたものであり，「ソフトウェア供給者候補の能力の判定を助ける」，「組織自体のソフトウェア開発プロセスの改善を助ける」，「新しいプロジェクトを請け負う際の自身の能力の判定を助ける」等の目的を持つ．SPICE に関しては日本では日本規格協会のプロセスアセスメント委員会を一つの母体として検討が進められている[25)~27)]．

4. ソフトウェアプロセスの効用

4.1 統合環境による柔軟な標準化手段の提供

ソフトウェア開発環境とは，プロセスを確立する方法ではなく，効果的なプロセスをより一層効率的にするものである[24]．プロセスモデルの確立が自動化のきっかけを作り，環境の充実がそれを実現する．L. Osterweil によって提案された，プロセスプログラミングの考え方に刺激されて，ソフトウェアエンジニアリング諸活動を体系的に記述するための様々な計算モデルとプロセス記述言語が提案された．研究室レベルのプロトタイプには多くの開発事例がある（たとえば参考文献 28)）．

プロセスモデルに基づく統合環境の開発は我々に**柔軟な標準化の手段を与える**ものと予想される[29]．すなわち，以下のような手段で，方法論や技術の進歩に応じて，対応する部分を抽出・変更することが可能になる．(1)組織やプロジェクトチームの特性に適合するプロセスモデルをまず定義し，(2)それに関係するツール，プロセス記述，データベーススキーマ，通信プリミティブ機構等をとりだし，(3)カスタマイズした後に，(4)それらを統合する．

4.2 ツール統合による作業の連続性の保持

一連の作業の支援は CASE ツール単独では可能でない．作業ごとのツールや，作業グループごとに選択されたツールキットを利用するのが普通である．このとき，ツール統合の機構が必要となる．「ツール統合」の概念は 1970 年代中期に出現した．すなわち，ツール機能をツールフラグメント群の合成機能としてとらえ，ツールフラグメントの再利用を促進しようとする技術である．下流 CASE ツール統合機構の成功をステップとして，現在，**開発工程全体にわたって作業の連続性を保証するツール統合機構**の実現が技術開発の目標となっており，プロセス中心のツール統合はデータ中心のツール統合や制御中心のツール統合とならび有力な接近である[30]．

5. 今後の課題

片山[31]によれば，ソフトウェアプロセスに関する今後の研究課題は以下のとおりである．

1. プロセスの再利用や構成/進化を促進するプロセスモデルの研究
2. プロセス記述言語の開発
3. 実プロセスの分析・解析を目的としたプロセスプログラミングの実施
4. 実際の組織やプロジェクトにおけるプロセスの上演，制定 (process cnactment)
5. プロセススクリプトの実行における計算機支援機構（柔軟な実行制御機構，成果物の格納と長時間トランザクションへの対応，コミュニケーション支援機構，ナビゲーション支援機構，スケ

ジューリング機構）の開発

6. プロセス実行の担い手が人間であることを考慮に入れた柔軟な誤り回復機構の検討

7. 協調，ポリシー，調整の明示的記述

8. 動的プロセス修正，プロセス進化のようなダイナミズムの取扱い

上記のような基礎研究の発展をふまえつつ，実用的な技術としての位置を獲得するためには以下の課題も検討・推進される必要がある．

1. **事例研究の必要性**　プロセスモデルはソフトウェア開発や保守をガイドし制御するのにどの程度有効かという問いに答え，研究の方向性を定めるために必要である．

2. **規模への連続的対応**　小規模プロセスから大規模プロセスまでうまくカバーするためには，部分的記述とその起動，統合の手段を開発するとともに，組織構成員間のコミュニケーション支援と融合させる機構を開発する必要がある．

3. **汎用プロセスモデルのカスタマイズ**　汎用プロセスモデルの主要なパラメータとカスタマイズの手段を検討することは，柔軟な標準化を実現するための一つの前提条件である．このためには，Adaptation 機能を開発・充実させる必要がある[32]．ここで，Adaptation 機能とは，種々のデータベース機能やツールの集合（メタ SDE）から，ユーザ固有の作業形態に適合した開発環境（specific SDE）を，プロジェクトの規模[33]，対象分野の特性，チーム構成員の分散度，データの安全性や機密度，利用する計算機設備，コスト等のパラメータに応じて生成または選択抽出し合成する機能である[34]．

4. **ソフトウェアプロセスと CSCW 研究の融合**　ソフトウェアプロセスにおける人間要因に技術的に対応する一つの手段として，ソフトウェアプロセスに関する研究と CSCW* に関する研究の接点を追求することは重要である．一般に，ソフトウェア開発は，必ずしもスキルレベルが揃っていない専門家集団の協調作業である．割り当てられた仕事を，各自が首尾よく達成しつつ，全体としては，共通のゴールを共有・理解してプロジェクトを成功に導くためには，チーム構成員間のコミュニケーション支援は重要な課題であり，プロジェクトの規模が大きくなるにつれてその比重は増す[35),36)]．

複数の人間がアイディアや中間成果物をネットワークを介して共有し，討論や変換活動によって，それらを変化させていくようなソフトウェア開発形態に対して，どのようなモデルが適切であり，また，どのような計算機支援が可能であるかという問題は今後検討される必要がある[37),38)]．1994 年 10 月に開催される CSCW'94 において「CSCW とソフトウェアプロセスの関係」を検討するワークショップが開催される予定である．

5. 分散化への対応

21 世紀に向けて，ソフトウェア開発の形態は，各拠点に分散して存在する人的資源，知的財産を論理的に統合して実施する方向に動きつつある．このため，ソフトウェアプロセスに関する研究コミュニティは分散開発時代への移行を考慮に入れた研究対象の拡大が必要である．分散データベース，分散オペレーティングシステム，インタネットワーキング等の要素技術を前提として設計された環境アーキテクチャの上での仕事の形態を考慮に入れる必要があろう[39]．

参考文献

1) Proc. of the 1st ISPW (1985).
2) Software Engineering Note (1985 年に開催された 2nd ISPW の特集), Vol. 11, No. 4 (1986).
3) Proc. of the 3rd ISPW (1986).
4) Proc. of the 4th ISPW (1988).
5) Proc. of the 5th ISPW (1989).
6) Proc. of the 6th ISPW (1990).
7) Proc. of the 7th ISPW (1991).
8) Proc. of the 8th ISPW (1993).
9) Proc. of the 2nd ICSP (1993).
10) 井上：第 2 回ソフトウェアプロセス国際会議報告，日本ソフトウェア科学会第 1 回ソフトウェアプロセス研究会資料，SP93-1-10，pp. 89-92 (1993).
11) Derniame, J. C.: EWSPT'92 Report, Proc. of the 2nd ICSP, pp. 160-164 (1993).
12) 岸田：プロセスプログラミング, Seamail, Vol. 2, No. 4 (1987).
13) Osterweil, L.: プロセスプログラミング, Seamail, Vol. 2, No. 8 (1987).
14) 片山他：第 1 回ソフトウェアプロセスワークショップ（JSPW）報告, Seamail, Vol. 4, No. 10-11 (1989).
15) 中島：第 5 回 ISPW 報告，Seamail，Vol. 5, No. 1-2 (1990).
16) 玉井：ISPW 初体験の記（6th ISPW に参加して), Seamail, Vol. 5, No. 8-9 (1990).

* Computer Supported Cooperative Work

17) 岸田, 野村：ソフトウェアプロセスに関する調査（第1次集計結果), Seamail, Vol. 6, No. 12 (1991).
18) 松原：ISO 9000-3は本当にソフトウェア品質の改善に役立つか？, Seamail, Vol. 8, No. 4 (1993).
19) Haraut, J.：ソフトウェア工学の観点から見たISO 9001およびISO 9000-3の適用の困難性についての分析, Seamail, Vol. 8, No. 4 (1993).
20) ソフトウェアプロセスシンポジウム論文集, 情報処理学会 (1994).
21) Kishida, K., Katayama, T., Matsuo, M., Miyamoto, I., Ochimizu, K., Saito, N., Sayler, J. H., Torii, K. and Williams, L. G.: SDA: A Novel Approach to Software Environment Design and Construction, Proc. of the 10th ICSE, pp. 69-79 (1988).
22) Osterweil, L.: Software Processes are Software Too, Proc. of the 9th ICSE, pp. 2-13 (1987).
23) 元治：ソフトウェアプロセスに関する第8回国際ワークショップの報告, 日本ソフトウェア科学会ソフトウェアプロセス研究会第1回報告集, pp. 89-92 (1993).
24) 藤野監訳：ソフトウェアプロセス成熟度の改善, 口科技連 (1991).
25) 松原：SPICEの背景と概要, SEA Forum「ソフトウェア・プロセス・アセスメントの国際規格SPICEの動向」配布資料 (1994).
26) 掘田：Baseline Practices Guide, 同上.
27) 青山：Process Assessment Guide, 同上.
28) Osterweil, L. and Taylor, R.: The Architecture of the Arcadia-1 Process Centered Software Environment, Proc. of the 6th ISPW, pp. 155-158 (1990).
29) 落水：CASEとは, 日本ソフトウェア科学会サマーチュトリアル資料 (1991).
30) Schefstrom, D. and van den Brock, G.: Tool Integration, Wiely (1993).
31) 片山：ソフトウェアプロセスとその研究課題, 日本ソフトウエア科学会第11回大会, ソフトウェアプロセスミニワークショップ基調講演 (1994).
32) Riddle, W. E.: Software Designer's Associates: A Preliminary Description, Proc. of the 20th HICSS, pp. 371-381 (1987).
33) Perry, D. E. and Kaiser, G. E.: Models of Software Development Environments, Proc. of the 10th ICSE, pp. 60-68 (1988).
34) Tully, C.: SDE Architecture Issues, Proc. of the 6th ISPW, pp. 31-36 (1990).
35) Curtis, B., et al.: On Building Software Process Models Under the Lamppoast, Proc. of the 9th ISPW, pp. 96-103 (1987).
36) 落水, ソフトウェア開発におけるグループウェアの役割, ソフトウェア・ツール・シンポジウム '92, pp. 83-92 (1992).
37) 落水, ネットワークを介した協調作業に関するモデルとツールの課題, 電子情報通信学会, 信学技報, CST94-12, pp. 17-24 (1994).
38) 落水, 門脇, 藤枝, 堀：ソフトウェア分散開発支援環境「自在」のアーキテクチャ設計, 電子情報通信学会, 信学技報, SS94-18, pp. 1-8 (1994).
39) Rodden, T., Mariani, J. A. and Blair, G.: Supporting Cooperative Applications, Computer Supported Cooperative Work, Vol. 1, No. 1, pp. 41-67 (1992).

(平成6年9月13日受付)

落水浩一郎（正会員）

1946年生. 1969年大阪大学基礎工学部卒業. 1971年同大学院修士課程, 1974年同博士課程修了. 工学博士. 静岡大学工学部講師, 助教授, 教授を経て, 1992年北陸先端科学技術大学院大学情報学研究科教授. ソフトウェア工学, 特に, オオブジェクト指向ソフトウェア分析・設計方法論, ソフトウェア開発環境, ソフトウェアプロセスモデル, Computer Supported Coperative Software Developmentに興味をもつ. 著書「ソフトウェア工学実践の基礎-分析・設計・プログラミング」(日科技連). ソフトウェア科学会会員.

[付録]共通例題☆
ソフトウェアプロセスモデリングのための例題

M. I. Kellner, P. H. Feiler, A. Finkelstein,
T. Katayama, L. J. Osterweil, M. H. Penedo
and H. D. Rombach
訳：篠田陽一, 鈴木正人
北陸先端科学技術大学院大学 情報科学研究科

1. はじめに

本例題はソフトウェアプロセスモデリングにおける様々なアプローチの理解, 比較を助けることを目的として設計されている. このような比較を行うとき, 対象となるアプローチの相対的な強さや弱さの評価を指摘する人がいる. しかしながら, このような強さや弱さの決定は, 与えられたアプローチが達成しようとしているゴールと目的の集合に依存する. たとえば, その主たる目的がプロセスの実働の自動的なサポートである場合, 直接実行可能性は何よりも重要である. 一方で, 人間の理解を助けることや, コミュニケーションに関することに重点をおくアプローチにおいては, 直接実行はさほど重要ではない.

理解と比較を容易にするため, 本例題は実世界のソフトウェアプロセスに見られる問題を数多く含むように注意して設計されている. これにより, 多くの異なったカテゴリに属する問題をモデル化する機会を提供している.

☆ 本共通例題はIEEEの許可を得て, M. I. Kellner et al., "Software Process Modeling Example Problem", Proc. of 6th ISPW pp. 19-29を翻訳したものである. Copyright by IEEE.

本例題は一つの核問題（core problem）といくつかの拡張問題（optional extension）から構成されている．異なったモデリングのアプローチを理解するための最初の共通の立場を与えるために，核問題を解くことが必要とされる．異なったアプローチの可能性を示すために拡張問題があり，これらはより自由なモデル化の機会を与える．

本例題を作成するにあたり，小さくて単純な問題と，盛り沢山で複雑な問題の間の困難なバランスを決めるために多くの努力が費やされた．単純すぎる問題は非現実的であり，複雑すぎる問題は手に負えなくなるためである．我々はまた例題と解答の適切性を保証するために，例題を他のドメインからでなく，ソフトウェアプロセスのドメインから採用した．さらに核問題の仕様に関しては，それを我々の共通の理解に任せたり，未指定のまま放置することがないように詳細な記述を行った．このため問題の作成に多くの時間を費やし，記述自体も長いものになったが，これにより解答を比較する上での安定かつ一貫した基盤を与えることが期待される．拡張問題はアプローチの能力を示すための広範囲の場を提供するものであり，上で述べた核問題の自由度の低さを補うものである．

2. 核問題

2.1 概　要

核問題は，ソフトウェア変更プロセスの比較的限定された部分に注目している．ソフトウェアシステムの設計，コーディング，単体テスト，比較的局所的な変更の管理に焦点を当てている．これら一連の動作は変更の要求によって始まり，ライフサイクルにおいては開発フェーズの後期か，あるいはサポートフェーズ（メンテナンスおよび改良）の期間中に発生するものと考えられる．問題を限定するために，与えられた要求の変更はすでに解析され，コンフィギュレーション管理委員会のような監督機関によって承認されているものと仮定する．変更は（モジュールのような）単一のコードユニットのみに影響を与えることが分かっている．本例題は，プロジェクトマネージャが変更をスケジューリングし，適当なスタッフに仕事を割り当てることから始まる．新バージョンのコードが新しい単体テストをパスした時点で，例題は終了する．

本例題を記述するために，いくつかの方法が用いられている．文章による物語風の解説は，広く理解されているという利点を持つので，ここでもこの方法を選択することにするが，例題の文章を理解しやすくするためにはある程度の構造化が必要である．ここでは物語の筋を主要なタスクによって大まかに構成し，各々のタスクに対しては構造化を最小におさえるようにした．話の構成が解答にバイアスを与えることも懸念されるが，この方法によってそのようなバイアスを最小におさえたままで，問題に対するより良い理解を得ることが可能であると思われる．

各々のタスクの説明は，そのステップの総括的な解説で始まる．ここにはモデル化のアプローチに基づいて，タスクが何をどのように扱うか（少なくとも記録するか）が物語風の記述によって示されている．この記述に続いて，入力と出力のリストが示される．ソースは入力ごとに示され，デスティネーションが出力ごとに示される．物理的な通信メカニズム（電子メール，あるいは手渡しなど）も指定される．すべての可能な入出力がこの部分にリストアップされてはいるが，クラスのインスタンスによっては入出力が決定しないこともあり得ることに注意する必要がある．そのような特殊な場合は，そのタスクの説明の節において解説されている．たとえばデザインレビューからのフィードバックは，デザインが承認されなかったときに限って Review Design の出力となる．そのタスクを実行する責任を持つ人が次に示される．その後，そのステップの実行に関する様々な制限が記述される．扱われるオブジェクトに関する詳細のような付加的な情報を含むタスクもある．

最初の説明はこの例題プロセス（核問題のみ）全体のカプセル化された抽象に関するもので Deveolop Change and Test Unit というタイトルがついている．続いて下に示すようなプロセスの部分ステップに関する説明がある．

- Schedule and Assign Tasks（タスクのスケジューリングと割当て）
- Modify Design（デザインの修正）
- Review Design（デザインレビュー）
- Modify Code（コードの修正）
- Modify Test Plan（テスト計画の修正）
- Modify Unit Test Package（単体試験パッケージの修正）
- Test Unit（単体試験）
- Monitor Progress（進行状況の監視）

これらの部分ステップの説明の前に，次節でグローバルな問題を議論することにする．

2.2 全体的な条件

核問題においては，資源の制限や競合は起こらないことを仮定する．したがって，変更に関する作業は，エンジニアがタスクの割当てを受け取った時点でただちに始まる．資源が利用できないことによる遅れは生じないものとする．簡潔さのために，共通の入力と出力はいちいちあげられていない．プロジェクトマネージャによって実行されないタスクをテクニカルステップと定義する．このとき，以下の情報は，この例題プロセスのすべてのテクニカルステップにおいて利用することができる．したがって各ステップで特に記述されない．

- 要求変更（Schedule and Assign Tasks より手渡し）
- タスク割当てとスケジュール日程の通知（Schedule and Assign Tasks より電子メール）
- 修正されたタスク割当てとスケジュール日程の通知（Monitor Progress より電子メール）

上記に加え，以下の情報は各テクニカルステップによって必ず作成される．それゆえ各ステップでは記述されない．

- 各ステップが終了したことの通知（Monitor Progress へ電子メール）

以前のステップへのフィードバックは Review Design ステップから Modify Design ステップへのように，明示的に示されたときのみに限られるという

モデル化がなされるべきである．現実にはそのようなフィードバックの多くの例（Modify Code から Modify Design，あるいは要求を明確にするための戻り，および Test Unit から Modify Design 等）が認められるのであるが，この例題ではモデル化する必要はない．我々の興味は与えられたアプローチが，そのようなフィードバックをどう扱うかを決定することであり，実際のプロセスの豊富な挙動を完全にモデル化することではない．

問題にバリエーションを与えるため，オブジェクトは紙の上の手書きのもの（設計書等）と，計算機上にあるもの（ソースコード等）を仮定する．物理的コミュニケーションの機構あるいはメディアには，適当に決められたものもある．手作業のインスタンス（口頭，または手渡し）は，代わりの手段でおきかえることが許される．たとえば，デザインは文書ではなく自動化された形式で示されるとしてもよい．しかし，自動化されたもの（電子メールや計算機入出力）は指定されたままでなければならない．さらに，この例題ではいくつかのオブジェクトは，自動化されたコンフィギュレーション管理の下にあるものとして記述されている．これらはソースコード，オブジェクトコード，単体テストパッケージであり，これらに対して意図される解答は，修正のためにオブジェクトの最新バージョンを取り出し，修正された新バージョンを挿入するようなものである．

核問題のそれぞれの解答にはその詳細が隠蔽された抽象的なステップ Develop Change and Test Unit および，以前説明した 8 つの部品ステップへの分解が最終的には示されなければならない．しかしながら，中間的なグループ化や抽象化は好きなように採用して構わない．たとえば，前に定義したテクニカルステップというグループ化を使ってもよい．

次に，この例題プロセスに関係する組織的構造について解説する．プロジェクトチームは対象となるソフトウェアシステムに関する仕事に責任を持つ．チームは一人のプロジェクトマネージャと，ソフトウェアエンジニアのグループで構成される．エンジニアは設計とコーディングに責任を持つ“design engineer”（設計エンジニア，以後 DE）というグループと，テストの作成と実行に責任を持つ“quality assurance engineer”（品質保証エンジニア，以後 QAE）というグループから構成される．各タスクにおけるこれらの人々あるいはチームの責任は，各々のタスクにおいて示される．また，“configuration control board”（コンフィギュレーション管理委員会，以後 CCB）という組織がプロジェクトチームとは別に存在し，総括的な方向付けの機能と必要な権限を持っている．

2.3 変更の実行およびテスト—Develop Change and Test Unit

2.3.1 説　明

このステップはこの核問題でのプロセス記述における最上位の抽象である．詳細は核問題の残りの部分に記述される．解答ではこのステップの記述と共に，より詳細な記述への分解が示されなければならない．

2.3.2 入　力

1. 要求変更（CCB より手渡し）
2. 権限（CCB より口頭）

2.3.3 出　力

特定の出力はない．最終結果は様々なファイルやデータベースの中，あるいは修正されたソフトウェアユニットの様々な表現（デザイン，ソースコード，オブジェクトコード）および対応する修正された単体テストに含まれる．これらはソフトウェアライフサイクル中において，この例題で扱う変更に続くすべてのステップにおいて利用できる．

2.3.4 責　任

このステップはプロジェクトマネージャによって実行される．

2.3.5 制　限

1. このステップは CCB によって変更作業の許可が与えられたらただちに開始する．
2. このステップは単体テストが成功のうちに終了したときか，CCB が変更を取り消したときに終了する．

2.4 タスクのスケジューリングおよび割当て—Schedule and Assign Tasks

2.4.1 説　明

このステップはプロジェクトを管理する機能であり，この例題プロセスで最初に実行されるものである．このステップはこのソフトウェアの変更に関する作業のスケジュールを作成し，個々のタスクを指定されたスタッフのメンバ，具体的には DE と QAE に割り当てる作業を含む．デザインのレビューもスケジュールに加えられ，適当な作業者（DE 1 人，QAE 1 人，他に 2 人のソフトウェアエンジニア）がこれを担当する．

この例題の後に続く基本プロセスは組織にとって標準的であり，作業者はそれを熟知しているものと仮定する．それゆえ，管理ステップはタスクに対するリソースの割当ておよびタスクを実行するために必要な資源の要求の見積りを行うことに重点をおく．

2.4.2 入　力

1. 要求変更（CCB より手渡し）
2. 変更許可（CCB より口頭）
3. 作業計画（ファイルより計算機入力）

2.4.3 出　力

1. 更新された作業計画（ファイルへ計算機出力）
2. タスク割当てとスケジュール日程の通知（関連する全員へ電子メール）
3. 要求変更（作業を割当てられた全員へ手渡し）

2.4.4 責　任

このステップはプロジェクトマネージャによって実行される．

2.4.5 制　限

1. このステップは，CCB による変更許可が与えられたらただちに開始する．
2. このステップは出力が作成されたときに終了する．すべての出力は同時に作成されると仮定する．

2.5 デザイン変更—Modify Design

2.5.1 説　明

このステップは，要求変更により影響を受ける単体コードユニットのデザインの修正を行う．このタ

スクは非常に創造的なタスクである．修正されたデザインはレビューされ，最後にコードという形で実現される．デザインレビューのステップからのフィードバックに基づいたデザインの修正も行う．

2.5.2 入　力

1．現在のデザイン（ソフトウェアデザインドキュメントファイルより手渡し）

2．デザインレビューからのフィードバック（Review Design より手渡し）

2.5.3 出　力

1．修正されたデザイン（Review Design, Modify Code, Modify Unit Test Package へ手渡し）

2.5.4 責　任

このステップは指定された DE によって実行される．

2.5.5 制　限

1．このステップは，プロジェクトマネージャによるタスク割当てがなされたらただちに開始する．

2．（デザインが認可されていないときに）デザインのレビューが完了すると，すぐに次の繰返しが開始される．

3．このステップは，出力が決定したときに終了する．

2.6 デザインレビュー—Review Design

2.6.1 説　明

このステップは修正されたデザインの正式なレビューを行うものであり，デザインの修正を行った DE を含むチームによって行われる．レビューの結果は以下の三つのうちのどれかである．

1．無条件合格—デザインはすべて認可される；修正後の認可されたデザインは，ソフトウェア設計書に組み込まれる．

2．小変更要請—デザインの小さな変更が要求され，デザイナにはフィードバックが与えられる．再レビューはおそらく形式的なものとなる．

3．大変更要請—デザインの大規模な変更が要求され，デザイナにはフィードバックが与えられる．

レビューが終了すると，プロジェクトマネージャに結果が通知される．この組織のプロセスにおいては正式なデザインレビューが比較的新しいステップであるということから，その評価を行うために必要な測定を行っている．ここでは発見された誤りの数と，レビューを準備し，実行しているレビューチームの全体の作業がプロジェクトマネージャに報告される．

2.6.2 入　力

1．修正されたデザイン（Modify Design より手渡し）

2.6.3 出　力

1．デザインレビューからのフィードバック（Modify Design へ手渡し）

2．認可された修正後のデザイン（ソフトウェア設計書ファイルへ手渡し）

3．レビュー結果の通知，発見された誤りの数，作業の報告（Monitor Progress へ電子メール）

2.6.4 責　任

このステップは，プロジェクトマネージャに指定されたデザインレビューチームによって実行される．このチームは DE 一人，QAE 一人，それに二人のソフトウェアエンジニアを含む．

2.6.5 制　限

1．このステップは，スケジュールされた時点で入力が存在すれば実行される．（入力が遅れた場合，Monitor Progress ステップがレビューを後に回すように再スケジュールする）

2．このステップは出力が決定したときに終了する．すべての出力は同時に決定されるものと仮定する．

2.7 コード変更—Modify Code

2.7.1 説　明

このステップはデザインの変更をコードに実現し，修正されたソースコードをオブジェクトコードにコンパイルする．実現は，既存のソースコードを修正することによって行われる．このステップは，ソースコードの追加修正の要求を示すテスト結果のフィードバックに基づいて行われる場合もある．

2.7.2 入　力

1．修正されたデザイン（Modify Design より手渡し）

2．現在のソースコード（ソフトウェア開発ファイルより計算機入力）

3．コードのフィードバック（Test Unit より口頭）

2.7.3 出　力

1．修正されたソースコード（ソフトウェア開発ファイルへ計算機出力）

2．オブジェクトコード（ソフトウェア開発ファイルへ計算機出力）

2.7.4 責　任

このステップは DE によって実行される．

2.7.5 制　限

1．このステップは，プロジェクトマネージャによってタスクが割り当てられたらただちに開始する．（したがって，あまり勧められないことではあるが，デザインが始まる前にコーディングを開始することも可能とする．）デザインが認可されていない場合は，エンジニアの判断でいつこのタスクを開始するかを決める．もしこのステップが先に開始していないときは，デザインが完成してから開始される．

2．テストのためにリリースされた修正後のコードは，修正されたデザインを反映している．このステップは修正されたデザインが認可される前に完了することはない．

3．コンパイルが完了し，出力が決定したときに，このステップは終了する．

4．次の繰返しは，Test Unit ステップが完成したときに開始する．

2.7.6 追　記

このステップは，その属性と相互関係の記録が必要とされるようなオブジェクトの集合を取り扱う．ソースコード，オブジェクトコードともに，自動化されたコンフィギュレーション管理の下におかれている．ソフトウェアユニットのソースコードの最新版は，このステップで行われる修正のための出発点となる．コンパイルが成功すると，ユニットのソースコードと新しいバージョンは対応するオブジェクトコードといっしょに記録される．さらにこれらは，

それらの元となっている修正されたデザインと対応づけられている必要がある．

ソースコードの情報は，本体，タイムスタンプ，バージョン情報，そしてこの変更の責任者であるエンジニアを含む．オブジェクトコードの情報は，本体，目標となるシステム，タイムスタンプを含む．あるソースコードは複数のオブジェクトコードと対応することがあり，同様に，あるデザインのバージョンは複数のソースコードのインスタンスに対応することがある．与えられたソースコードのインスタンスのコンパイルにおいて，タイムスタンプ，使用したコンパイラのバージョンとオプション（オプティマイザやデバッグの有無など），エラーメッセージや警告，メモリマップのような，付随するドキュメントなどを記録することは重要である．

2.8 試験計画の変更—Modify Test Plans

2.8.1 説　明

このステップは，ソフトウェアの修正のきっかけとなる要求変更に関連した機能の試験を含む試験計画および目的の修正を行う．これらの試験計画はソフトウェアデザインに似ている．試験されるべき機能と能力，試験の方法を規定する．一方で，実際の試験データや手続きなどは，次のステップ（Modify Unit Test Package）で扱われる．

2.8.2 入　力

1. 現在の試験計画（試験計画ファイルより手渡し）

2.8.3 出　力

1. 修正された試験計画（試験計画ファイルへ手渡し）

2.8.4 責　任

このステップは指定されたQAEによって実行される．

2.8.5 制　限

1. このステップはプロジェクトマネージャによるタスクの割当て後ただちに開始する．
2. このステップは出力が決定したときに終了する．

2.9 単体試験パッケージの変更—Modify Unit Test Package

2.9.1 説　明

このステップは，試験計画と目的のための修正に従って，影響を受けたコードユニットのための実際の試験パッケージの修正を行う．試験パッケージは，付随するコンピュータファイルに記録された文章による手続きの他にも，試験の下でユニットを動かし評価するためのコンピュータソフトウェアを含むと仮定される．これらの試験パッケージは自動化された構成管理の下におかれ，単体試験パッケージの新しいバージョンはこのステップが終了したときに作成される．修正されたデザインおよびこのユニットのためのソースコードはこのステップへの入力としても使われることもある．ただし，いつもというわけではない．

このステップの次の繰返しは，単体試験パッケージの追加修正が要求されたことを示す試験からのフィードバックに基づいて行われる．

2.9.2 入　力

1. 修正された試験計画（試験計画ファイルより手渡し）
2. 現在の単体試験パッケージ（試験パッケージファイルより計算機入力）
3. 修正されたデザイン（Modify Designより手渡し）
4. ソースコード（ソフトウェア開発ファイルより計算機入力）
5. 試験パッケージのフィードバック（Test Unitより口頭）

2.9.3 出　力

1. 修正された単体試験パッケージ（試験パッケージファイルへ計算機出力）

2.9.4 責　任

このステップは指定されたQAEによって実行される．

2.9.5 制　限

1. このステップはModify Test Planステップの完了後ただちに開始する．
2. このステップは出力が得られたときに終了する．
3. （要求があれば）次の繰返しはTest Unitステップが完了したときに開始する．

2.9.6 追　記

単体試験パッケージのインスタンスは，ソフトウェア，手続き，タイムスタンプ，バージョン情報，そしてこのパッケージインスタンスの責任者であるエンジニアを含む．

2.10 単体試験—Test Unit

2.10.1 説　明

このステップは，修正されたコード上の単体試験パッケージの実行と，その結果の解析を行う．DE，QAEともにこのステップに関与している．試験パッケージ全体の実行が，解析やその先の活動に先立って行われる．単体試験は有用であるが，単体コードに対する妥当なカバレージの達成を試すために自動化されたカバレージアナライザが用いられており，カバレージが90%に達すると受け入れられる．すべての試験が完了し，カバレージが90%に達すると単体試験は終了する．さもなければ，DEとQAEが共同で試験の結果を解析し，適切な活動を決定する．問題を制限するために，ソースコードの修正，単体試験の修正のどちらか一方あるいは両方が必要であるということを解析の結果とする．指定された修正が行われると，ユニットは再試験される．

もしユニットが試験にパスし，許容し得る達成率を満たせば，この例題プロセスは完了する．インテグレーション試験のような単体試験以上のステップは核問題の範囲を越えるものである．プロジェクトマネージャには，ユニットコードの最新バージョンがインテグレーション試験に利用できることが通知される．

2.10.2 入　力

1. オブジェクトコード（ソフトウェア開発ファイルより，計算機入力）
2. 単体試験パッケージ（試験パッケージファイルより，計算機入力）

2.10.3 出 力

1. 試験の結果（試験履歴ファイルへ計算機出力）
2. コードのフィードバック（Modify Code へ口頭）
3. 試験パッケージのフィードバック（Modify Unit Test Package へ口頭）
4. 成功した試験の通知（Monitor Progress へ電子メール）

2.10.4 責 任

このステップはDEとQAEの共同で実行される．

2.10.5 制 限

1. このステップは，オブジェクトコードと単体試験パッケージがともに利用可能になった時点で開始する．
2. このステップは出力が決定したときに終了する．すべての出力は同時に得られると仮定する．

2.10.6 追 記

コードのインスタンス上に対する単体試験の実行のインスタンスの集合は，単体試験記録ファイルに記録される．この記録は，試験実行のタイムスタンプ，試験に失敗したという表示，到達したカバレージに加え，使用される単体試験のバージョンとオブジェクトコードのバージョンも含んでいる．複数の単体試験パッケージのバージョンが，与えられたオブジェクトコードのバージョンに適用される可能性もあれば，その逆もあり得ることに注意せよ．

2.11 進捗状況管理—Monitor Progress

2.11.1 説 明

このステップはプロジェクトマネージャによるタスクの進行状態のモニタリングを行う．これは各ステップの終了通知や，その他の非公式な情報に基づいて行われる．タスクがプランに沿って実行されている間は，このステップでは何も起こらず何も出力されない．しかしプランからの逸脱が生じると，タスクの再スケジューリングが行われる．逸脱が深刻な場合には，プロジェクトマネージャはすべての修正作業を中断し，CCBにこの変更作業を中止するように勧告する．CCBは中止に同意するか，中断した箇所からの再開を指示する．このような場合，関係者は彼らの作業について，その再開または中止をプロジェクトマネージャより通知される．

2.11.2 入 力

1. 終了通知（各テクニカルタスクより電子メール）
2. 現在の作業プラン（ファイルより計算機入力）
3. レビュー結果，発見された誤りの数，成果を集めたもの（Review Design より電子メール）
4. 試験成功の通知（Test Unit より電子メール）
5. キャンセルに関する決定（CCB より口頭）

2.11.3 出 力

1. 更新された作業プラン（ファイルへ計算機出力）
2. 修正されたタスク割当て，および日程スケジュールの通知（関係者全員へ電子メール）
3. 中止勧告（CCB へ口頭）

2.11.4 責 任

このステップはプロジェクトマネージャにより実行される．

2.11.5 制 限

1. このステップは最初のタスク，すなわち Schedule and Assign Task が終了したらただちに開始する．
2. このステップは例題プロセスが動いている間中ずっと存在する．この例ではこのタスクは次のどちらかが成立したときに終了する．

（1） 単体試験が成功のうちに終了したとき
（2） CCBによるキャンセルが関係者に通知された後

3. 拡張問題

3.1 概 要

この節ではこれまでに述べた核問題のいくつかの拡張を示す．これらの目的は与えられたモデリングアプローチの核問題によって示すことのできない能力を示すさまざまな機会を与えることである．各々の拡張は核問題の上に構成され，それぞれ独立したものと解釈される．これらは開かれており，自由に解釈できるような余地を残してある．

3.2 コード変更の詳細

この拡張は Modify Code ステップの詳細をより完全にするためのものである．ソースコードの変更はエンジニアによって直接行われるが，オブジェクトコードの変更はコンパイラが行う．現在のソースコードはこのステップの始まりにおいて，構成管理のライブラリから取り出される．エンジニアはソースコードを変更するためにエディタを起動する．ソースコードがコンパイルできるようになったらソースコードファイルを保存してコンパイルを起動する．コンパイルエラーがあった場合は，ソースコードを変更して再コンパイルを行う．デザインが承認される前にコンパイルが成功した場合，デザインが承認された後，単体試験のためにリリースされる前に，コードの最終的なチェックがエンジニアの手作業で行われる．これによりソースコードの変更が必要になることもある．

コードの中間バージョンとコンパイル情報，生成物がコンフィギュレーション（バージョン）管理機構のもとで個々のライブラリ（データベース）に保持されることを仮定すると都合がよいだろう（要求はしない）．最終バージョンだけがソフトウェア開発ファイルに格納された形で必要である．

3.2.1 基本的な詳細の具体化

一つの選択肢は，低レベルのプロセスステップや詳細の扱いを，適当な詳細を加えて上の記述を表現することで示すことである．

3.2.2 ツールの自動的起動

さらに野心的な拡張は，これをツールの自動起動と組み合わせることである．ツールには少なくともエディタとコンパイルが含まれなければならない．ここで仮定されているコンフィギュレーション管理ツールや最終バージョンがソフトウェア開発ファイルにおかれ，単体試験で利用できるようになったときの電子メールのメッセージを自動的に作成するツールなども自動化の枠組に含むことが望まれる．

3.3 デザインレビューの詳細

この拡張は Reviw Design ステップの詳細をより完

全にするためのものである．このステップはエンジニアのグループによって実行され，作業者間のコミュニケーション，共同作業，協調を含んでいる．このステップをさらに詳細にするには，このようなグループによるレビュープロセスがどのように発生するかに関するダイナミクスを扱う．この拡張を含むために，どのような適当なレビュープロセスをも仮定してよい．

3.4 スケジュールとリソースの制限

この拡張はスケジュールとリソースの制限に関するプロジェクト管理の問題を扱う．

3.4.1 基本的スケジューリング

プロセス内の各ステップの妥当な所要時間を仮定する（Modify Design に 3 日，Modify Code に 4 日，など）．タスクを達成するためのリソースは必要なだけ遅れなしで利用できると仮定する．この仮定はプロセス中の繰返しループ（デザインが承認されるまでの Modify Design-Review Design の数回の繰返しのような）を扱わなければならない．（作業が始まる前の）初期スケジュールがプロセスモデルに基づいてどのように決まるかを明示せよ．

この作業は Schdule and Assign Tasks ステップの内部で実行される．

3.4.2 スケジュールの修正

上から導かれる初期スケジュールがプロセスの実行の間に得られた実際の結果を反映してどのように修正されるかを明示せよ．たとえば，コーディングが予想より 2 日余計にかかったとする．単体試験を完了するために残りのスケジュールにどのような影響を与えるか，またそれをどうやって発見するか．

この作業は Monitor Progress ステップの内部で実行される．

3.4.3 リソース制限下におけるスケジューリング

先に述べた拡張の一つあるいは両方を，リソースの制限を扱うように修正せよ．これにはタスクが実行可能となったときに作業者がいない，または部分的にしかそのタスクに従事できない（全体の 50%の時間しか作業できないといったような）場合が含まれる．

3.5 複数の変更の複合

この拡張は複数の変更を扱うことを要求する．ソフトウェアの単一の変更を考えず，多くのものが独立に，しかもおそらく並列に開発されていると仮定する．個々の変更は核問題で記述されている．しかしながら，変更の集合がそれぞれ独立した単体試験をパスしたとき，それらは完全なシステムにまとめられ，新しい構成に対するインテグレーション試験が行われることになる．もちろんインテグレーション試験では，単体試験またはシステムの他の部分に存在する問題が明らかになり，それを修正するための変更がさらに必要になるであろう．

3.6 プロセスの変更

この拡張はプロセスに対する変更の同定とその実現を含む．簡単なものとしては，改良する可能性を特定するためにプロセスを解析し，一つかそれ以上を選択しておき，将来使用できるようにしておくことが考えられる．この例として，単体試験の間に見つかった誤りを解析し，この種の誤りを検知できるように将来のデザインレビューを改良することを試みることがあげられる．これを行うためには，試験の結果として記録されるデータに，（エラーのクラス分けなどの）付加的な情報を与え，プロセス改良のための変更完了後のステップが付け加えられるべきである．

さらに挑戦的なものは，実行時のダイナミックなプロセスの変更を行うことである．たとえば，割当てられた作業者が作業できなくなってしまった場合，2 回目以降のデザインレビューを行わない決定がなされる場合を考えることができる．この決定は，実行されないレビューがスケジュールされたときにダイナミックに行われ，プロセスの残りの部分はこのインスタンスにより継続される．プロセスモデルはこのようなダイナミックな変更をどのように扱うか？

3.7 モデル作成者の選択

モデル化のアプローチの特徴を示すためのどのような拡張も可能とする．それぞれの拡張問題の性質が述べられ，モデルの特殊な能力が強調されるべきである．