トリプティックプロセスモデル [1]
プロセスの評価と改善

Dines Bjørner

Computer Science and Engineering
Informatics and Mathematical Modelling
Technical University of Denmark
DK-28000 Kgs.Lyngby
Denmark

Graduate School of Information Science
Japan Adv. Inst. of Science & Technology
1-1, Asahidai, Tatsunokuchi
Nomi, Ishikawa 923-1292
Japan

bjorner@gmail.com

June 14, 2006. Compiled October 2, 2006

[1] Invited keynote talk for the JASPIC (Japan Software Process Improvement Consortium) Conference, 12–13 October 2006, Tsukuba, Japan

- ソフトウェア工学へのトリプティックアプローチは注意深く監視され制御された反復可能な工程を基盤として以下の項目を通じて進行される.
  - ⋆ ドメイン工学,
  - ⋆ 要求工学,
  - ⋆ ソフトウェアデザイン
- 今回の講演では,
  - ⋆ 上記の3つの段階の概要を述べ,
  - ⋆ それぞれの段階の開発の中での多くのステージを見せ,
  - ⋆ それぞれのステージの中での多くのステップを簡単に述べる.

トリプティックドグマ
背景

- 過去において, ソフトウェア工学に関する主な教科書では
  - ⋆ 要求工学とソフトウェアデザインに焦点をしぼったソフトウェア工学について述べられている.
- トリプティックドグマは
  - ⋆ 要求工学とソフトウェアデザインの2つで説明されていたソフトウェア工学を拡張し,
  - ⋆ ドメイン工学を加えた3つの段階から説明されるソフトウェア工学のためのものである.

ドグマ

- 要求規定の正当化
  - ⋆ ソフトウェアがデザインされる前に
  - ⋆ 我々は要求を理解しなければならない.
- ドメイン記述の正当化
  - ⋆ 要求が規定される前に
  - ⋆ 我々はドメインを理解しなければならない.
- トリプティックの正当化
  - ⋆ 最初に(アプリケーション)ドメインの解析と記述を行い,
  - ⋆ 要求の解析と規定を行い,
  - ⋆ 最後にソフトウェアデザインとコードの解析と明記を行う.

新しい側面

- ソフトウェア開発において新しい側面は’ドメイン工学’である.
- この新しい側面はドメイン, 要求工学の新しいいくつかの方法論的側面に “解釈” される.
- 次の主な節ではこれらの側面について見ていく.
- 上記に関して, 今年Springerから出版された本 “Software Engineering 3” に詳細が書かれている.
- 買いに行って!今すぐ!

トリプティックプロセスモデルとドキュメント
共通した側面
プロセスモデル

- トリプティックプロセスモデルは3つのプロセスモデルから構成されている. 各々,
  - ⋆ ドメイン工学,
  - ⋆ 要求工学,
  - ⋆ ソフトウェアデザイン.

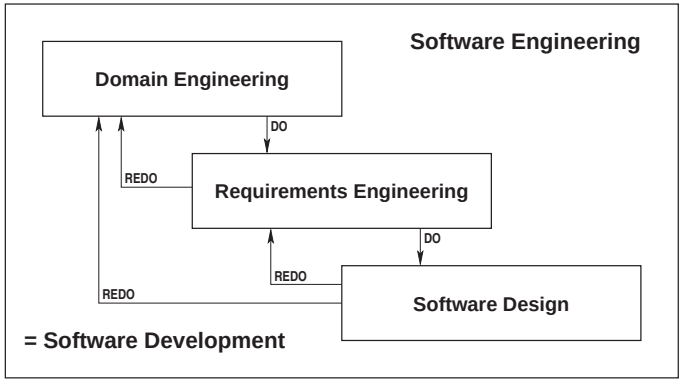

Figure 1: A simplified view of the triptych process model

ドキュメント

- ソフトウェア開発の3つの段階に共通した部分はそれらが一義的なドキュメントの開発であることである.
- 後ほど説明する図は結果として生成される上記のようなドキュメントの幅, 深さ, そしてかなり実質的な数を明示したものである.
- そして, このようなドキュメントの各集合に共通した部分は事実上, インフォメーションドキュメントの管理を行っている部分である.

1. Information
   (a) Name, Place and Date
   (b) Partners
   (c) Current Situation
   (d) Needs and Ideas
   (e) Concepts and Facilities
   (f) Scope and Span
   (g) Assumptions and Dependencies
   (h) Implicit/Derivative Goals
   (i) Synopsis
   (j) Standards Compliance
   (k) Contracts
   (l) The Teams
      i. Management
      ii. Developers
      iii. Client Staff
      iv. Consultants
   (m) Plans
      i. Project Graph
      ii. Budget
      iii. Funding
      iv. Accounts
   (n) Management
      i. Assessement
      ii. Improvement
         A. Plans
         B. Actions

Figure 2: Informative documents

ドメイン工学プロセスモデル
ドメインモデル

- ドメイン工学による開発を特定のアプリケーションドメインに適用することによって得られる主な結果はドメインモデル.
- ドメインモデルは記述の形で表現される.
- ドメイン記述は「何があるのか」, そしてそれの「ありのままの姿」を記述する.
- ドメイン記述が要求を含んでいることは期待されていない.
- ドメイン記述は要求規定ではない.
- 類推としては,
  - ⋆ 物理学者[ドメイン工学者]は自然の世界[アプリケーションドメイン]を記述し
  - ⋆ 工学者[要求工学者, ソフトウェアデザイナー]は要求の規定と実装を行う.

ドメイン工学, 物語形式

- ドメイン工学トリプティックドグマは以下のことを唱道する.
- (item 2.) 記述開発のステージ(インフォメーションドキュメントへの取り組み[items 1.a–l]が十分に行われた後)
  - ⋆ (2.a) ドメイン利害関係者の範囲の識別,
  - ⋆ (2.b) ドメイン理解の獲得,
  - ⋆ (2.c) ドメインで使われている用語(存在する言葉使い)の確立,
  - ⋆ (2.d) 全ての関係のあるビジネスプロセスの理解とラフスケッチ,
  - ⋆ (2.e) ドメインモデリング(ドメインの全ての面), そして
  - ⋆ (2.f) ドメインモデルの完成(整理統合も含む).

- ドメイン記述の部分(item 2., subitems (a–f))に編みこまれているのは解析の部分で,
  - ⋆ (3.a) 不整合, 衝突, 不完全を見つけるための解析
  - ⋆ (3.b) ドメインの妥当性検証,
  - ⋆ (3.c) ドメインの検証, そして
  - ⋆ (3.d) ドメインの法則の設立への可能な取り組み.

ドメイン工学のドキュメント

2. Descriptions
   (a) Stakeholders
   (b) The Acquisition Process
      i. Studies
      ii. Interviews
      iii. Questionnaires
      iv. Indexed Description Units
   (c) Terminology
   (d) Business Processes
   (e) Facets:
      i. Intrinsics
      ii. Support Technologies
      iii. Management and Organisation
      iv. Rules and Regulations
      v. Scripts
      vi. Human Behaviour
   (f) Consolidated Description
3. Analyses
   (a) Domain Analysis and Concept Formation
      i. Inconsistencies
      ii. Conflicts
      iii. Incompletenesses
      iv. Resolutions
   (b) Domain Validation
      i. Stakeholder Walkthroughs
      ii. Resolutions
   (c) Domain Verification
      i. Model Checkings
      ii. Theorems and Proofs
      iii. Test Cases and Tests
   (d) (Towards a) Domain Theory

Figure 3: Domain engineering document table-of-contents

ドメイン工学のステージとステップ

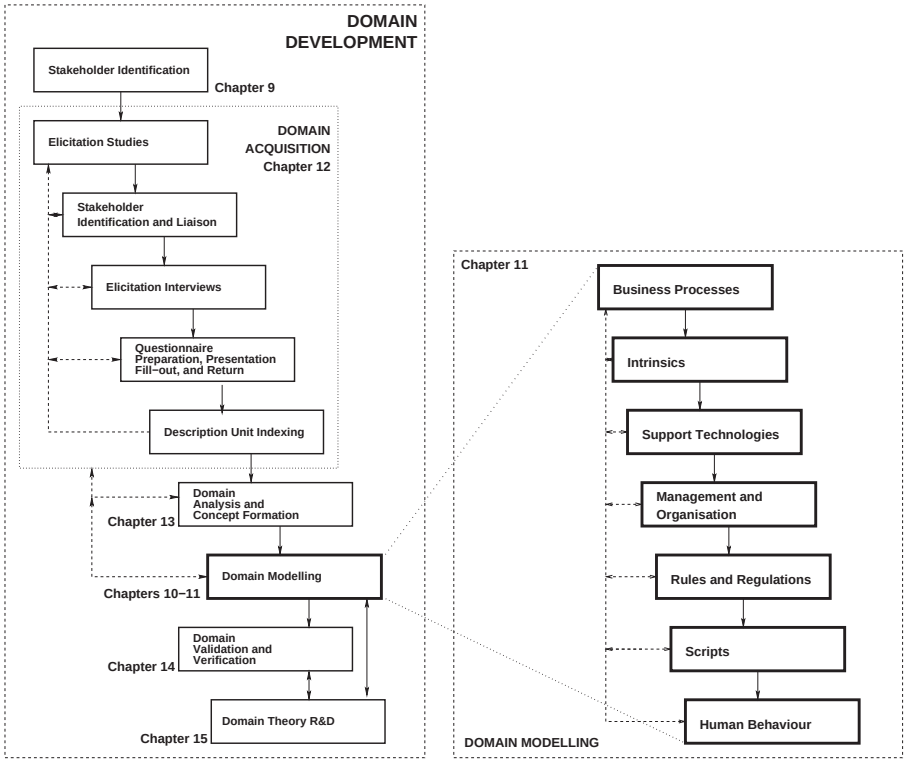

Figure 4: The domain engineering process model diagram

Figure 4はドメイン工学の開発のステージとステップ, それらの相互関係を示したものである.

要求工学プロセスモデル
機械

- 要求は機械を規定するものである.
  - ⋆ ハードウェア,
  - ⋆ ソフトウェア.

  上の2つは要求を実装するものである.
- 機械はドメインの中に存在する.
- 一度開発されると, 我々は
  - ⋆ 時々その機械が存在しているドメインをその機械の環境としてみなす —
  - ⋆ 機械 + 環境が新しいドメインとなる.

要求モデル

- 要求工学開発を
  - ⋆ ある特定のアプリケーションドメイン[2]に適用することによって得られる主な成果物は
  - ⋆ 要求モデルである.
- ドメインモデルは記述の形の中にある.
- 要求規定は何があるべきかを規定する.

[2] ドメインの例: (1) 金融サービス業, (1.1) 銀行, (1.1.1) 銀行の不動産抵当貸付ビジネス; (2) 交通業, (2.1) 鉄道システム, (2.1.1) 連動システム; 等.

要求工学, 物語形式

- 要求工学トリプティックドグマは以下のことを唱道する.
  - ⋆ (item 2.) 規定開発のステージ(インフォメーションドキュメントへの取り組み[items 1.a–l]が十分に行われた後)
    - ◇ (2.a) 要求利害関係者の範囲の識別,
    - ◇ (2.b) 要求提示の獲得,
    - ◇ (2.c) 非形式的な要求の発見のための最初の要求モデルのラフスケッチ,
    - ◇ (2.d) 要求のための用語(存在する言葉使い)の確立,

そして

- (2.e) 要求の全ての面の要求モデル,
  - (2.e.i) ビジネスモデルの再設計
    (BPR: business process reengineering),
  - (2.e.ii) ドメイン要求,
  - (2.e.iii) インターフェース要求,
  - (2.e.iv) 機械要求, そして
  - (2.e.v) 要求規定の完成.

- ⋆ 要求規定の部分(item 2., subitems (a–e))編みこまれているのは解析の部分で,
  - ◇ (3.a) 不整合, 衝突, 不完全を見つけるための解析
  - ◇ (3.b) 要求の妥当性検証,
  - ◇ (3.c) 要求の検証, そして
  - ◇ (3.d) 要求の法則の設立への可能な取り組み.

要求工学のドキュメント

2. Prescriptions
   (a) Stakeholders
   (b) The Acquisition Process
      i. Studies
      ii. Interviews
      iii. Questionnaires
      iv. Indexed Description Units
   (c) Rough Sketches (Eurekas, IV)
   (d) Terminology
   (e) Facets:
      i. Business Process Re-engineering
         - Sanctity of the Intrinsics
         - Support Technology
         - Management and Organisation
         - Rules and Regulation
         - Human Behaviour
         - Scripting
      ii. Domain Requirements
         - Projection
         - Determination
         - Instantiation
         - Extension
         - Fitting
      iii. Interface Requirements
         - Shared Phenomena and Concept Identification
         - Shared Data Initialisation
         - Shared Data Refreshment
         - Man-Machine Dialogue
         - Physiological Interface
         - Machine-Machine Dialogue

2. (Prescriptions, continued)
   (e) (Facets, continued)
      iv. Machine Requirements
         - Performance
            - ⋆ Storage
            - ⋆ Time
            - ⋆ Software Size
         - Dependability
            - ⋆ Accessability
            - ⋆ Availability
            - ⋆ Reliability
            - ⋆ Robustness
            - ⋆ Safety
            - ⋆ Security
         - Maintenance
            - ⋆ Adaptive
            - ⋆ Corrective
            - ⋆ Perfective
            - ⋆ Preventive
         - Platform
            - ⋆ Development Platform
            - ⋆ Demonstration Platform
            - ⋆ Execution Platform
            - ⋆ Maintenance Platform
         - Documentation Requirements
         - Other Requirements
      v. Full Reqs. Facets Doc.

Figure 5: Requirements engineering document table-of-contents: prescription documents

3. Analyses
   (a) Requirements Analysis and Concept Formation
      i. Inconsistencies
      ii. Conflicts
      iii. Incompletenesses
      iv. Resolutions
   (b) Requirements Validation
      i. Stakeholder Walk-through and Reports
      ii. Resolutions
   (c) Requirements Verification
      i. Model Checkings
      ii. Theorem Proofs
      iii. Test Cases and Tests
   (d) Requirements Theory
   (e) Satisfaction and Feasibility Studies
      i. Satisfaction: Correctness, unambiguity, completeness, consistency, stability, verifiability, modifiability, traceability
      ii. Feasibility: Technical, economic, BPR

Figure 6: Requirements engineering document table-of-contents: analytic documents

要求工学のステージとステップ

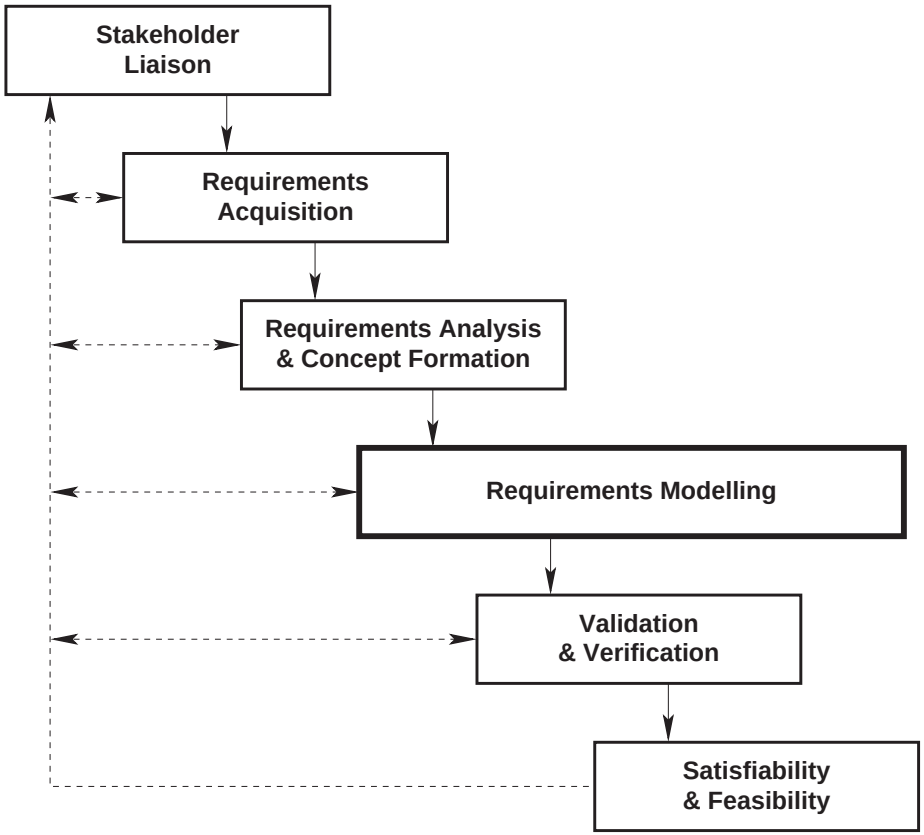

Figure 7: Diagramming a requirements process model

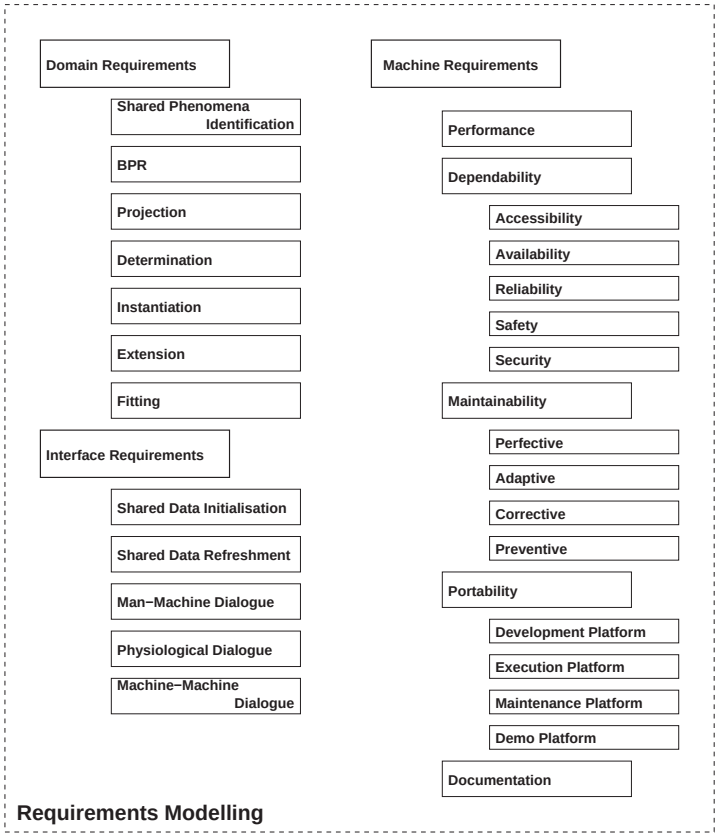

Figure 8: The requirements modelling stage

ソフトウェアデザインプロセスモデル
ソフトウェアデザイン, 物語形式

- ソフトウェアデザインプロセスは簡約すると4つのステージに分けることができる (Fig. 9 items 2.a–d):
  - ⋆ ソフトウェアアーキテクチャデザイン,
  - ⋆ コンポーネントデザイン,
  - ⋆ モジュールデザイン, そして
  - ⋆ プログラミング.
- これらは各々, 2つ以上の開発ステップから構成されていてもよい (cf. Fig. 10).
- 隣接したステップの間には正しさを保障する義務がある (V:MC:T, 検証, モデル検査, テスト).
  - ⋆ 検証で使われる証明は普通, 次のようなものである: $\mathcal{D}, \mathcal{S} \models \mathcal{R}$
  - ⋆ これは$\mathcal{S}$oftwareが$\mathcal{R}$equirementsを満たすことの証明は$\mathcal{D}$への参照を必要とすることを意味する.

ソフトウェアデザインのドキュメント

2. Software Specifications
   (a) Architecture Design ($S_{a_1} \ldots S_{a_n}$)
   (b) Component Design ($S_{c_{1_i}} \ldots S_{c_{n_j}}$)
   (c) Module Design ($S_{m_1} \ldots S_{m_m}$)
   (d) Program Coding ($S_{k_1}, \ldots, S_{k_n}$)
3. Analyses
   (a) Analysis Objectives and Strategies
   (b) Verification (S_{i_p}, $S_i \sqsupseteq_{L_i} S_{i+1}$)
      i. Theorems and Lemmas L_i
      ii. Proof Scripts $\wp_i$
      iii. Proofs Π_i
   (c) Model Checking ($S_i \sqsupseteq P_{i-1}$)
      i. Model Checkers
      ii. Propositions P_i
      iii. Model Checks $\mathcal{M}_i$
   (d) Testing ($S_i \sqsupseteq T_i$)
      i. Manual Testing
         • Manual Tests $M_{S_1} \ldots M_{S_\mu}$
      ii. Computerised Testing
         A. Unit (or Module) Tests C_u
         B. Component Tests C_c
         C. Integration Tests C_i
         D. System Tests $C_s \ldots C_{s_{i_{t_s}}}$
   (e) Evaluation of Adequacy of Analysis

Legend:

S Specification
L Theorem or Lemma
$\wp_i$ Proof Scripts
Π_i Proof Listings
P Proposition
$\mathcal{M}$ Model Check (run, report)
T Test Formulation
M Manual Check Report
C Computerised Check (run, report)
$\sqsupseteq$ "is correct with respect to (wrt.)"
$\sqsupseteq_\ell$ "is correct, modulo ℓ, wrt."

Figure 9: Software design document table-of-contents

ソフトウェアデザインステージとステップ

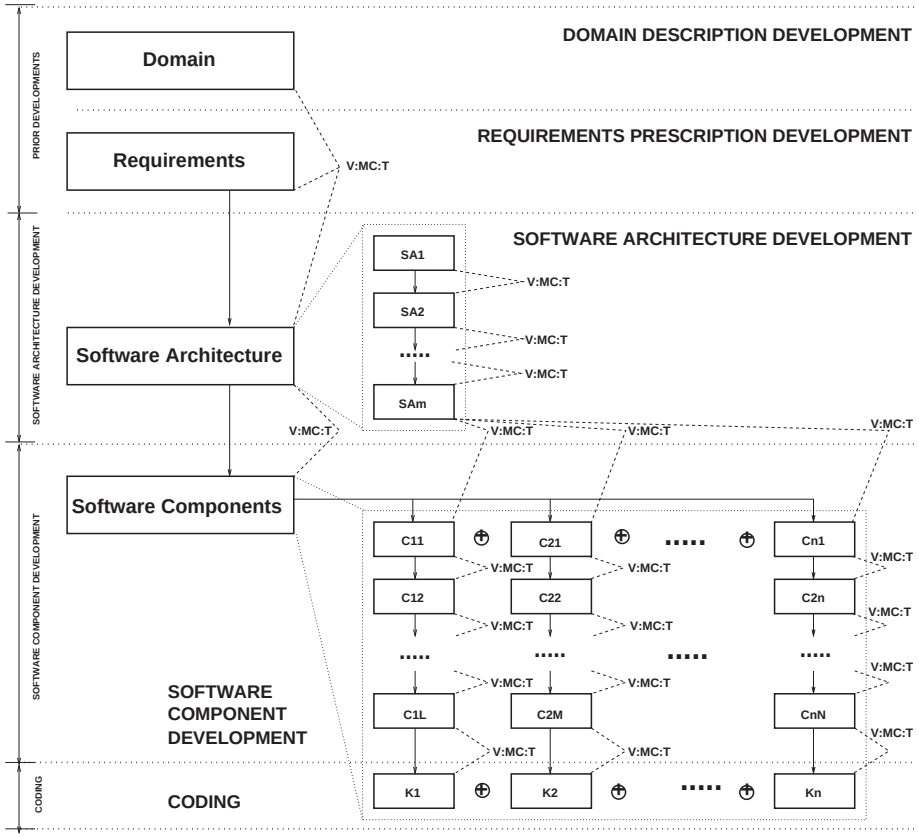

Figure 10: The software design development processes

トリプティックプロセスのまとめ
プロセスモデル: 図と内容の表

- 我々はトリプティックドグマに従って, (主に) ソフトウェア開発のプロセスを調査してきた.
- 我々はこれらのプロセスは図で表現でき, また各段階で生成されるドキュメントの集合に対する表(table-of-contents: 目次) としても表現できる, ことを見た.

プロセスモデルの意味

- Figs. 1, 4, 7–8 と 10 のような図,
  - ⋆ 実状を反映している,
  - ⋆ 構文を持っており
  - ⋆ 望むらくは意味を具体化している.

- 我々はここで意味を強調する:

  - ここで重要なのは
  - *Figs. 4, 7, 8と 10*の
  - 記述, 規定, ソフトウェアデザイン部分が囲まれている箇所と
  - 各々を結んでいる線分が
  - 方法の原理, テクニック, ツールの明確な集合を与えており,
  - また, これらの多くのテクニックが理論に基づくツールによって形式的に遂行可能あり支援される, または支援可能であることである.

以降ではプロセスモデルの図のセマンティックスにおいて上記のことは認められているものとする.

非形式的開発 対 形式的開発

- 用語'開発'は3つの段階の任意の組み合わせを行うことを意味する:
  - ⋆ ドメイン, 要求, またはソフトウェアデザインのいずれかのみ;
  - ⋆ ドメイン+要求, または要求+ソフトウェアデザイン,
  - ⋆ "大体の" 一貫性を保ち3つの段階を行うこと.
- 私の本で見られるように, 開発は
  - ⋆ **非形式的に**, または
  - ⋆ **形式的に**,
- そしてこれらの任意の "分けられた段階" の組み合わせの中で遂行される.

- **0. 非形式的開発**：以下の対象の形式化を行わない試み.
  - ⋆ ドメイン記述,
  - ⋆ 要求規定,
  - ⋆ ソフトウェアデザイン仕様
- よって, 形式的な証明, またはモデル検査を用いた検証は不可能.
- テストによる検証のみ.

おおまかに言って開発の半形式的段階から形式的段階へは3つの“点”がある.

- **1. 体系的な開発**はドメイン記述, 要求規定とソフトウェアデザインの形式化を行う. しかしこの開発では形式化しようと試みた部分を形式化するだけである.
- **2. 厳密な開発**は全てのきわめて重要[3]な性質を言明し, ときとしてはその性質の証明やモデル検査のスケッチを行ったり, 実際にそれらを行うことによって体系的な開発を拡張したものである.
- **3. 形式的開発**は全ての (正しく) 必要な性質が形式的に記述され, 定理証明やモデル検査が行われる開発である.

トリプティックパラダイムは, これらの3つ (1.–2.–3.) のいずれの開発形式に対して採用しても良い.

[3] ここでは‘重要’という語の厳密な定義は与えない.

段階, ステージ, ステップの支持

次の仮定を強調することは重要である:

- 我々にとってトリプティックパラダイムを支持することの意味は
  - ⋆ 以前概略を示した段階, ステージ, ステップが正確に遂行されることを意味する.
  - ⋆ これは以前示した, "表 (目次)" の各々の項目に対してドキュメントが生成されることを意味する.

プロセス評価と改善に対する次の我々の取り組みは上記の仮定から始める.

プロセス評価と改善の管理
‘プロセス評価’と‘改善’の概念

- ’評価’と’改善’について話すために我々は何が評価され改善されるのか識別しなければならない:
  - ⋆ ある方法を規定する原理/テクニック/ツール/管理の集合に従った結果が, 他の方法を規定する他の集合に従った結果との比較において、評価され改善される
- プロセス評価は与えられたプロセスがそのプロセスモデルを実状的, 意味論的, 構文論的に支持しているかの判断についてのものである:
  - ⋆ プロセスはプロセスモデルに “定められる” ことを (実状的, 意味論的, 構文論的に) どの程度で実現するか.
- プロセス改善は評価された開発プロセスに変更を加えることについてのものである. そのような変更が加えられたプロセス (改善されたプロセス) を使った結果が評価となる.
- 我々は最初は “評価” と “改善” を “ドキュメントを評価, 改善すること” と解釈する.
- ドキュメントはプロセスモデルの節とそれらを結ぶ線分によって表された活動から生成されるものである.

- そのような各箱や線分にはドキュメントが添付されていることがあり, そのようなドキュメントはそれぞれ構文, 意味, 実状を持っている.
- 構文や意味には普通, とても厳密な定義が与えられている.
- よって, 我々はある意味であるドキュメントがその構文と意味に “従っているか” を客観的に “はかる” (評価する) ことができる!実状を “はかる” ことに関してはより主観的となる.
- プロセスの改善を “はかる” ために各箱と線分のために計画されたドキュメントに構文, 意味, 実状に従っているかという “はかり” を添付しなければならない.
- ドキュメントは100%構文, 意味, 実状に従っているだろうか否か?
- もっと厳密に言えば, ドキュメントの質は0から1(“理想的”) のどこに位置づけられるのか?

Software Process Assessment 1

プロセスモデルの構文と意味: トリプティックのアプローチを使ってプロセス改善を上手く行うために (CMMの方法で, *from a lower to a higher level*), 管理者 (もちろん開発者も) は生成されたドキュメント, プロセスモデルの節と線分の活動から生成されると期待されているドキュメントの構文と意味を深く理解していなければならない. これは強い要件であり, どんなソフトウェア開発機構からも期待されない. そしてショートカットはない.[4] 観測されるリソースの使用の精度に関するプロセス改善はこの仮定で断言されている. 管理はトリプティック原理/テクニック/ツールの専門的な理解に基づいて管理が行われる, という仮定に基礎を置いている. 開発ドキュメントが関連する箱か線分の構文や意味を支持する "度合い"[5] で評価が決まる.

[4] 他の工学の分野ではプロジェクトの管理者 (プロジェクトの指導者) と開発者は基本的に同じレベルで標準的な教育を受けている. 従って, 彼らは自分たちの課題の構文や意味を深く理解している. 問題はソフトウェア工学の中にある.

[5] "度合い" の概念はここでは定義しない.

代表的なグループにPraxis High Integrity Systems, http://www.praxis-his.com, FORTIA: The Formal Techniques Industrial Association, www.fortia.orgなどある.

Software Process Improvement 1

プロセスモデルの構文と意味: 開発者や管理者がトリプティックプロセスモデルを使い従事できるようなプロセスの一般的な側面を改善するためには, 教育と訓練を行わなければならない. その他には方法はない.

- 我々はここでまた, Watts S. Humphrey(WSH)の *Capability Muturity Model*(CMM)について言及することによって我々の'プロセス改善'の話をつなぎ止めることを選ぶ.
- CMMは開発グループの成熟のレベルを5段階に分けている.
  - ⋆ レベル1は一番低く, "最も好ましくない" ことを表し,
  - ⋆ レベル5が最も高く, WSHが定義するのに役立つと考えたプロフェッショナリズムの "最も好ましい" レベルである.
- 開発グループによるプロセス改善は
  - ⋆ グループ(例:ソフトウェアハウス)が
  - ⋆ レベルiからレベル$i+j$まで進むような
  - ⋆ 開発プロセスの改善である.
  - ⋆ i, jは$i+j<6$となるような正の数である.
- まず, WSHのCMMの概念を復習してみよう.

The CMM: Capability Maturity Model

1. レベル 1, 初期
2. レベル 2, 繰り返し可能
3. レベル 3, 定義されている
4. Level 4, 管理されている
5. Level 5, 最適化されている

- 成熟レベル4と5の間にある決定的な違いは対象とするプロセス検証のタイプである.
- 成熟レベル4では, プロセスの違いの特別な原因と統計的な推論から得られる結果について焦点を当てている.
- プロセスは推論可能な結果を生成するかもしれないが, それらの結果は目標を設定するのには不十分かもしれない.
- 成熟レベル5では, プロセスを違える一般的な原因に注意して, 計量可能なプロセス改善目標に到達するためにプロセスを変化させ(プロセスのパフォーマンスの意味を変える), プロセスパフォーマンスの改善を(統計的な可能性を保持したまま)行うことに焦点を当てている.

プロセスモデルとプロセス

- 1つはプロセスのモデル, すなわちFig. 4, Figs. 7 and 8, Fig. 10で示した構造である.
- (これらは構文論的構造を表しているが意味論的意味も持っている.)
- もう1つはこのようなモデルの実際の使用である.
  - ⋆ すなわち, ソフトウェア開発者達
  - ⋆ (ドメイン, 要求, ソフトウェアデザイン技術者達)が
  - ⋆ ドメインのモデル, 要求のモデル, ソフトウェアデザインを開発するときに
  - ⋆ “進む” 実際のプロセスである.

グラフとグラフ縦断跡

- グラフのような表現形式で表されたプロセスモデルを仮定する. Fig. 11を見よ.

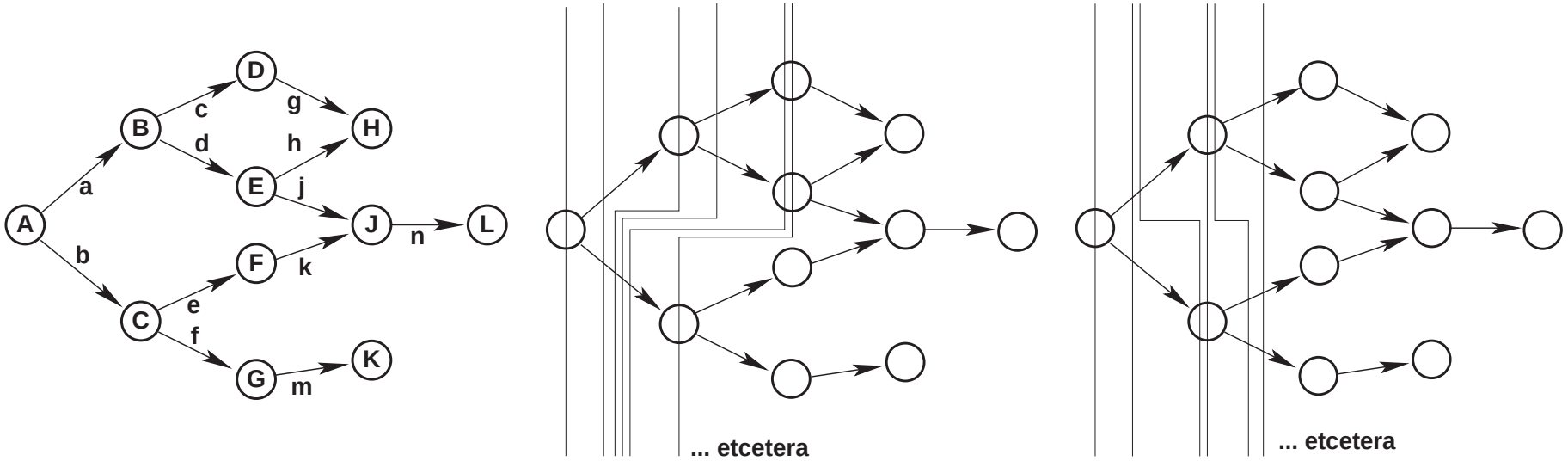

Figure 11: A graph (left) and two (incomplete) traversal traces (center and right)

- Fig. 11はあるプロセスモデルと2つのトレースを表している.
  - ⋆ REDOs, すなわち段階, ステージ, そしてステップの反復は追加のトレースを導き出す.
    - ◇ これらのトレースの全順序(集合)をOKトレースと呼ぶことにする.
  - ⋆ そして, “飛んでいる”, または単に “1つ飛ばし” の段階, ステージ, ステップはさらに追加のトレースを導き出す.
    - ◇ これらのようなトレースをNOKトレースと呼ぶことにする.
- 従って, プロセスモデルはこれらのようなトレースの無限集合として表すことができる.

プロセスモデルコンプライアンスの度合い

- 我々は今, プロセスモデルコンプライアンスの2つの概念を定義できる.
  - ⋆ 構文論と
  - ⋆ 意味論.
- プロセスモデルコンプライアンスのシンタックスの概念は実際のプロセスがプロセスモデルの反復可能なトレースと一致している “度合い” と関係している.
- プロセスモデルコンプライアンスのセマンティックスは箱と線分の意味論を支持するものと関係がある.
- 今回の講演ではこれらの概念を厳密に定義するようなことはしない. それはまた今後の講演で行うべきものである.

Software Process Assessment 2
構文論的プロセスコンプライアンス:

- 与えられた一般的なプロセスモデルは *Figs. 4, 7, 8, 10* で表されている.
- そして, プロジェクトの特定のソフトウェア開発のグラフは *Fig. 12* で例示されている.
- これらのモデルとグラフを支持しているプロセスの中で
- 実際のプロセスがこれらの図に従っているか評価することが容易にできる.

Software Process Improvement 2
構文論的プロセスコンプライアンス:

- プロセスモデルの支持は
- 少なくとも “形式的に”
- 実行された, または実行されていないと評価されたプロセスのステップやステージ (または段階) を
- 実際に確認することで改善される.

トリプティック開発のための “基礎 0”

- トリプティック開発とは, トリプティックドグマで規定された原理, テクニック, ツールを適用した開発を意味する.
- 体系的な, 厳密な, または形式的な方法のいずれかである.
- 従って, トリプティック開発は “定義によると” CMM のレベル 4 に基礎を置く.
- これはトリプティックドグマに従っているソフトウェア開発プロセスが少なくともレベル 4 のものであるという意味ではない.
- ドグマは標準を決めるものである.

Software Process Assessment 3

計画された構文論的, 意味論的コンプライアンス:

- もしあるプロセスが十分なコンプライアンスであると評価されたら,
- この場合のソフトウェア開発は *CMM* のレベル *4(*かまたはそれ以上*)*であると主張する.

Software Process Improvement 3

計画された構文論的, 意味論的コンプライアンス:

- もしあるプロセスが *CMM* のレベル *4* に到達していないと評価され,
- 少なくとも *CMM* のレベル *4* は必要であるなら,
- まず構文論的コンプライアンスを確保しなければならない. *プロセス改善# 2 (Slide 46)* を見よ.
- そして
  - ⋆ 意味論的コンプライアンスが低すぎる評価を得ている
  - ⋆ 各ステップ *(*またはステージ, 段階*)* を
  - ⋆ 意味論に従うように考え直さなければならない.

先を見越した計り

- 今までは一般的な評価と改善について述べてきた.
- これでプロセス評価と改善についてより具体的な論点について扱う準備はできた.
- しかし, まずは我々のプロセスモデルの概念をより正確にする必要がある.

プロジェクト開発グラフ

- プロセスモデル(つまりグラフ)は一般的なものである.
- モデルはどのようなソフトウェアの開発にも適応できる.
- モデルはある特定の問題フレームに適合することを例示されなければならない.
- Figure 12はthe Danish Ada compiler (1981–1984)で使われたプロジェクト開発グラフである.

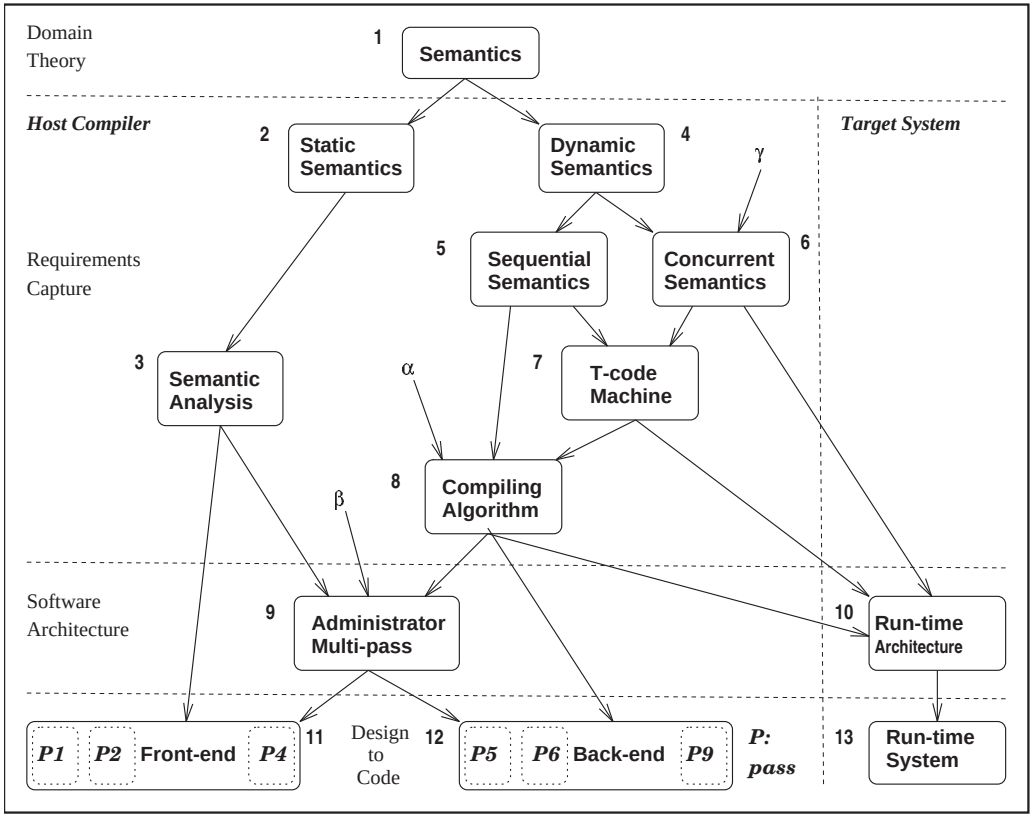

Figure 12: Project development graph: Compiler development

- このグラフは3段階のプロセスモデルを表している.

管理

- これまでこの講演では管理についてはふれてこなかった.
- 管理t[6] は計画, プロセスのリソースの使用の監視, 制御についてのものである. — リソースの使用から発生するドキュメントの質も含む.
- 計画はプロセスのスケジューリングや再スケジューリング, プロセスへの(からの)リソースの割り当て, 割り当ての取り消しについてのものである.
- ソフトウェア開発での基本的なリソースはドメイン, 要求技術者の集合とソフトウェアデザイナーの集合である.
- 他の基本的なリソースは開発者によって使われる時間, 空間, そしてツールである.

[6]我々は管理の意味を限定する. 我々は生産管理やプロジェクト資金については考慮しない.

計画 — スケジューリングと割り当て

- 計画は新しい, 試験的な, ソフトウェア開発グラフの初期設定, 選択, または開発から始まる.
- そしてプロセスモデルの概念に関してそれに詳細 (注釈) を与える.
  - ⋆ 段階 (ドメイン, 要求, ソフトウェアデザイン),
  - ⋆ ステージ (利害関係者の識別, 獲得, 解析, 記述 (規定, 仕様), 検証, モデル検査, テスト, 妥当性検証, etc),
  - ⋆ より決定的で詳細なステップ.

- 得られたソフトウェア開発グラフを基に
- 管理によってより詳しい方法で
- リソース使用 (人, 時間, 事務所, 設備, ソフトウェア開発ツール) は
- それぞれの箱や線分に関連づけられ
- 時間と空間に割り当てられることができる.

Software Process Assessment 4

リソース計画:

- どのようにしてソフトウェア開発プロジェクト計画 (これから起こるものを指示するグラフ) を評価することができるのだろうか?
- まず, "似たような" 開発問題を取り扱う傾向ことができる, 以前に作成されたソフトウェア開発グラフとその最終的な出力 (つまりそのグラフを用いた結果) と比較することができる.
- 実際のリソース使用の報告書を基にソフトウェア開発グラフを描き, リソース消費見積もり (時間, 人, 設備) をすべての節と線分に関連付けることができる.
- 従ってここで言う '評価' とはこれから取り組むプロジェクトの "憶測された評価" である.

従って, もしこれから取り組むプロジェクトの “憶測された評価” が評価する者 (管理) によって欠点がある, または疑問があると感じられたのなら, 改善を行う必要がある.

Software Process Improvement 4

リソース計画:

- まずはドメインの特定のプロジェクト開発グラフデザインの正確さを改善しなければならない.
- そしてリソース使用とグラフの各箱と線分との関連付けの正確さを改善しなければならない. 等.
- いくつかの開発プロジェクトは早い時期において “繰り返し” を伴う性質が強い. そして各 “繰り返し” においてプロジェクト開発グラフの改良が期待できる.
- 他のプロジェクトは実験的であり, 準備的なものの性質を持つ.
  - ⋆ つまりこれらは
  - ⋆ プロジェクトの終わりにおいて, ある開発者のみがプロジェクト開発グラフを知っているような研究プロジェクトに実際に適用される.
  - ⋆ そしてそこで開発されたグラフは “最も良いもの” である必要はない!

リソース使用の監視と制御

- プロジェクト(つまりプロセス)が進んで行くにつれ,
- 管理によって多くのことがチェックできる:
  - ⋆ スケジュールと割り当てに従っているか,
  - ⋆ プロセスモデルコンプライアンスの構文論的, 意味論的概念に従っているか.

- 多くのプロセスモデルではコンプライアンスの概念は構文論的なもの以外は浅薄である.
- トリプティックプロセスモデルにおいて意味論的コンプライアンスは中心的なものである:
  - ⋆ プロセスモデルの全ての箱と線分は
  - ⋆ これら (箱と線分) の期待される結果としてのドキュメントの厳密な構文論と意味論を持っている.

Software Process Assessment 5

リソース使用:

- ここでは問題は何もない.
  - ⋆ (開発プロセスの)各ステップが進んで行くにつれ
  - ⋆ そのプロセスから予測された計画への評価ができる.

- リソース使用の評価は
- 問題があることを示しており (例: ステップが終わるまで良く使われる全てのリソース),
- 何かが解決されなければならない:

Software Process Improvement 5

リソース使用:

- おそらく今の時期ではないが全ての計画したリソースを使い果たしたとき
- どんな改善もやり直すことができない. それはこの "次の" 時期かもしれない.
- 検査報告は過剰消費の原因を明らかにするかもしれない.
- 未経験な, 低すぎるリソース見積もり, または不適任なスタッフ, または単純なまたはそうでない間違い?
- ここで言う改善の意味: 繰り返しを避けるための予防措置を施す.

- リソース使用は、非常に詳細で説明可能なレベルにあるので, より良く評価され得る.
- 間違い (普通は過剰使用) は、
  - ⋆ 予測, または発見することが出来、
  - ⋆ もっと明瞭に定義された防止策を、
  - ⋆ マイルストーンが失われまたは過剰使用が起こるとしても、
  - ⋆ スキップされ得るステージやステップも含め、
  - ⋆ 規定することが出来る.

- ステージやステップが完全にスキップされるようなことがあるのはおそらく的外れなプロセスである.
- “理想の” プロセスモデルがあり,
  - ⋆ それ故, 適切なプロセス, 反復可能なプロセスの理解があるとすると,
  - ⋆ 管理によってどれが許容できる間違いかきちんと評価できる.

非形式的な開発から形式的な開発へ

- プロセス改善によって,
  - ⋆ 前に述べた我々のプロセス改善の性質をもう1度繰り返し, 拡張するために,
  - ⋆ 成果物としてのソフトウェアの質を改善する何かを理解する.
- 我々は‘成果物としてのソフトウェア’を‘成果物としてのドキュメント’と“解釈”する.
- これらのドキュメントは
  - ⋆ 非形式的(最後のプログラミング言語以外はどんな形式化も行わず)に開発される.

  または
  - ⋆ 体系的に形式的,
  - ⋆ または厳密に形式的,
  - ⋆ または形式的に形式的!

非形式的開発

- 次の性質はソフトウェア開発へのトリプティックアプローチの独立的な性質である.
  - ⋆ ドメイン工学, 要求工学, そしてソフトウェアデザインを形式化することによって
  - ⋆ これらを体系的な方法から厳密的な方法を通って形式的な方法でたどることができる.
- それ故, 開発の非形式的な側面は有益なドキュメントの開発のみに限定される.
- 有益なドキュメントは普通プロジェクトの指導者と管理者によって “開発” される.
- よって, “より上位の” レベルの管理はプロセス評価とことによると “より下位の” レベルの管理へのプロセス改善を規定することである.

Software Process Assessment 6
有益なドキュメントの非形式的開発:

- *Fig. 2 (Slide 8)*を参照する.
- この図は注意深く開発されるべきドキュメントの種類を挙げたものである. — つまり評価されるものである.
- 構文は規定されていないので,
  - ⋆ これらのドキュメントに形式的意味は無い.
  - ⋆ — これらの意図は語用に主に関してである —
  - ⋆ 評価はスタイルの問題である.
  - ⋆ 本質や中身のない有益なドキュメントを “すらすら”と無感覚的に書くのは容易である.
- それ故, 次の点について評価を行わなければならない: *Fig. 2* で挙げられた多くの有益なドキュメントの中の特定のドキュメントが、始められたプロジェクトの本質を、本当に簡潔な形で伝えているのか?

Software Process Improvement 6
有益なドキュメントの非形式的開発:

- もし有益なドキュメントが
  - ⋆ 必要な教育学的そして教育を意図した “振舞い” と共に
  - ⋆ ドキュメントの意図するメッセージが簡潔に伝えていないと評価されたら,
- そのドキュメントは改善される必要がある.
- “成熟された”, すなわち経験を積んだ管理者だけがこの改善を行える.

体系的, 厳密, そして形式的な開発

- 解析のドキュメント
  - ⋆ (テスト, 検証, モデル検査, そして妥当性検証)

  の開発と同様に
  - ⋆ ドメイン記述,
  - ⋆ 要求規定,
  - ⋆ ソフトウェアデザインの
- ドキュメントの開発は
- 体系的から厳密的を通って形式的な範囲で行われる.

Software Process Assessment 7
スタッフとツールの適性:

- プロセスモデルの中で
  - ⋆ 評価されるべきタスクの
  - ⋆ 特定のステップの構文と意味が与えられているとき,
- プロジェクト (タスク) の指導者かまたは管理者が
- コンプライアンスを評価できる.
- 評価はタスクの活動を支援するソフトウェアツール[7]の助けを大いに受ける:
- もしそれらがドキュメントを処理できるのなら,
- *OK*.
- もしできないのなら, 評価は否定的なものになる.

[7] これらのソフトウェアツールは主に主要なツール (仕様言語, 変換 (または改良), 証明システム) の使用を支援する.

- 判断基準の失敗となる2つの “両極端な” 理由がある —
- 実際の理由はこの2つの “両極端な” 理由の組み合わせになりうる.
  - ⋆ 1つはスタッフの質で, 彼らの行った仕事が期待したもの未満である場合である.
  - ⋆ もう1つは使われたツールが問題解決のタスクを支援するのに適当でなかった場合である.

スタッフの適性

- もし‘仕様の体系的な, 厳密な, そして形式的な開発と解析’の
- 評価が
- 不適当な開発決定のせいで否定的なものだと判断されたら
- 我々は次の改善策を提案する.

Software Process Improvement 7

スタッフの適性:

- 必要だと思われるとき次の3つの改善策が提案される:
  - 1つ目は
    - 開発のレベルを体系的なレベルから厳密なレベルに“移行”する,
    - または, 可能であれば厳密なレベルから形式的なレベルに移行し
    - 影響を受けるタスクをやり直す.

- ⋆ 2つ目は
  - ◇ 影響を受けたタスクを正確にやり直せるように
  - ◇ スタッフを教育, 訓練する.
  - ◇ (開発のレベルはそのままで)
- ⋆ 3つ目は
  - ◇ 影響を受けるスタッフを教育, 訓練された別のスタッフと交代し,
  - ◇ 影響を受けたタスクをやり直す.
- これらの改善決定は重大なものである.

ツール

- ツールにはいろんな種類がある.
  - ⋆ ツールは管理のために次の点について役立つ:
    - ◇ ソフトウェア開発グラフのデザイン (Fig. 12),
    - ◇ 図の “融合” によって生成される適切なプロセスモデル図 (Fig. 4, Fig. 7 and 8, Fig. 10).
    - ◇ 図に関連したプロセスの監視と制御 (すなわち評価と改善).

- ⋆ そしてツールは開発者に
  - ◇ 構文論的,
  - ◇ 意味論的

  記述, 規定を供給する.
  さらには, ソフトウェアデザインツールや
  - ◇ テスト,
  - ◇ モデル検査, そして
  - ◇ 検証 (証明補助, 定理証明器) に役に立つ,
  - ◇ 解析ツールもある.

- ⋆ これらのツールはテキスト形式, または図形式の記法を使い形式化を行うためのものである.
  - ◇ Alloy,
  - ◇ B,
  - ◇ CafeOBJ ,
  - ◇ Cas,
  - ◇ Duration Calculus,
  - ◇ LSCs,
  - ◇ MSCs,
  - ◇ Petri Nets,
  - ◇ RAISE RSL,
  - ◇ Statecharts,
  - ◇ TLA+,
  - ◇ VDM-SL, or
  - ◇ Z.
- ⋆ 上に挙げた13の言語は
  - ◇ テキスト形式化か
  - ◇ 図形式化か
  - ◇ それの結合したもののいずれかであるような

  ツールである.
- ⋆ そしてこれらは言語学的な方法, 分析的方法の使用を支援するためのソフトウェアパッケージである.

ツールの適性

- もし‘仕様の体系的な, 厳密な, そして形式的な開発と解析’の評価が
- 不適当なツールのせいで否定的なものだと判断されたら
- 我々は次の改善策を提案する.

Software Process Improvement 8

ツールの適性:

- より良いツールを選択し影響を受けた (つまり否定的な評価を受けた) タスクに適応しなければならない.
- これらのツールは
  - ⋆ 知的なもの, つまり仕様言語 (テキスト形式または図形式), そしてそれらの改良と証明システム,
  - ⋆ または知的なツールを支援する明白なソフトウェアツール.
- これらは同じく慎重な改善決定である.

プロセス評価と改善の妨げとなるもの

- 何が評価と改善の “標準な” 妨げとなりうるであろうか?
- そして何がトリプティックプロセスモデルで
  - ⋆ プロジェクトを実行するときの
  - ⋆ 似たような妨げとなりうるであろうか?

方法論の知識の不足

- 管理スタッフと開発スタッフは
  - ⋆ トリプティックプロセスモデルとそのシンタックス, セマンティックス, とプラグマティックスとの関係,
  - ⋆ 体系的から厳密的を通って形式的な開発のためのトリプティックプロセスモデルの必要性,
  - ⋆ 無数のドキュメントの作成, 使用, 保守, 相互関係のためのトリプティックプロセスモデルの必要性,
  - ⋆ 評価と可能な改善のためのトリプティックプロセスモデルの必要性に詳しくなければならない.
- 方法論の知識の不足はこれまで以上適切なソフトウェア開発プロセスの妨げとなる.

世代間のギャップ

- 古くから我々は若い志願者がソフトウェア技術者としてソフトウェアハウスで仕事を始めるのを見てきた.
  - ⋆ 彼らはトリプティックアプローチ固有の原理, テクニック, ツールなどを使いこなす能力に長けている.
- 彼らはこうした新しいツールや方法を大いに使いたがる.
- しかし彼らは大抵窒息し挫折する.
  - ⋆ なぜなら, 彼らより年配の同僚やプロジェクトの指導者は,
  - ⋆ そうしたツールや方法を使う技能を持たず,
  - ⋆ または完全にトリプティックメソッド (原理, テクニック, ツール) などに関して無知である.
- 世代に渡っての方法論の知識の不足は
- 適切なソフトウェア開発プロセスの妨げとなる.

ツールの不足

- 前に知的なツールとそれを使うための支援となるソフトウェアツールがあることを示した.
- 以下ではそれら両方のツールを考える.
- 一方で,
  - ⋆ ある特定のソフトウェア開発プロジェクトが取り掛かっている問題は
  - ⋆ 現在2006年において, プロジェクトの適切な評価可能, 改善可能な追求のための
  - ⋆ 明らかなまたは十分良い知的なツール(と方法論的アプローチ, すなわちプロセスモデル)がないかもしれない.
- また一方で,
  - ⋆ 適切な知的ツール(とプロセスモデル)が利用可能なときでも
  - ⋆ 適切なソフトウェア支援ツールが利用可能でないかもしれない.
- ツールの不足は適切な開発プロセスへの深刻な妨げである.

受容の不足

- 適切な開発プロセス —
  - ⋆ トリプティックプロセスモデルによって提案されるような —
  - ⋆ 適切に評価され
  - ⋆ 連続的な改善可能性が存在するプロセス —

  への最も一般的な妨げは
  - ⋆ (1) "formal methods" と呼ばれるものの受容の不足, そして
  - ⋆ (2) それを行うことの必要性の受容の不足である.
- ここはこれらの "事実" を嘆く時と場所ではない.

まとめ

まとめに入る.

要約

- 我々は理解しやすいプロセスモデルであるトリプティックモデルの全体像を見てきた.
  - ⋆ トリプティックモデルは3つの開発段階:
    - ◇ ドメイン工学,
    - ◇ 要求工学,
    - ◇ ソフトウェアデザイン,
  - ⋆ を規定し, 段階の中で多くのステージ, ステージの中で多くのステップを規定する.
- 段階, ステージ, そしてステップは繰り返し可能であり, それらとそれらの間の推移はドキュメントとして記述される.

- 我々はプロセスモデルを
  - ⋆ サイクルを含まないグラフと,
  - ⋆ その上に定義される波(wave)の任意に長い跡の無限集合として, 定義した.
    - ◇ 波はグラフの節と線分の集合である.
    - ◇ 跡の続きは繰り返されるかもしれない(プロセス繰り返し(“前の”タスクを再び行うこと)によって)

- 我々は
  - ⋆ 7つのソフトウェアプロセス評価の種類のクラスと
  - ⋆ 8つのプロセス改善の種類のクラスを

  全てトリプティックプロセスモデルの構文論と意味論に関係付けて識別した.
- そしてプロセス評価と改善への妨げについて簡単にふれた.

将来の展望

- 講演者にとってトリプティックモデルを
  - ⋆ SPA(Software Process Assessment) とSPI(Software Process Improvement),
  - ⋆ そしてWatts HumphreyのCMM

  に関連付けたのは今回が初めてであった.
- このような試みは有益なものになった.
- 明らかに
  - ⋆ トリプティックアプローチを適用するための
  - ⋆ そして今回の講演の中で提案した評価と改善を実行するための実際のプロジェクトについて
  - ⋆ より明らかな方向性と
  - ⋆ 開発された支援ツールを示さなければならない.

ソフトウェア調達
ソフトウェア

- ここでソフトウェアとは
  - ⋆ 実行可能なコードや
  - ⋆ インストール方法のマニュアルやコードの使用や修正だけではなく
  - ⋆ コードを開発するプロジェクト全体から発生する全てのドキュメントのことを意味する.

調達

- ソフトウェア調達で買い上げられるべきものは,
  - ⋆ 先のいくつかの図の中で示したすべてのドキュメントの中で,
  - ⋆ CMMのレベルと関係づけられ評価で合格したものある, とするのが自然であろう.

謝辞

- 講演者は
  - ⋆ 日本SPIコンソーシアムの方々のために今回の論文を書く機会を与えて下さった岸田孝一氏[8],
  - ⋆ 日本語の通訳をおこなって下さった二木厚吉教授,
  - ⋆ 英語から日本語のスライドへ翻訳をした有本泰仁

  に感謝します.

[8] 岸田孝一氏が今回の講演にいないのがただ残念だ!しかし11/22に彼の誕生日を祝して私は今日の内容の講演を東大のセミナーで行う!