ソフトウェア工学

13：ソフトウェア開発の
ベスト プラクティス

理工学部 経営システム工学科
庄司　裕子

今回のテーマ

ソフトウェア開発のベストプラクティス

- 開発プロセス モデルと支援ツールの現状
  - 「現状」と言いつつ、ちょっと古い…
- 開発プロセスとベスト プラクティス
- 開発方法論、支援ツール

2

開発プロセスと
ベスト プラクティス

ソフトウェア開発のベスト プラクティス（最善の実践原則）とは

- ソフトウェア開発上の問題の根本原因を解決できることが開発現場で実証されている開発アプローチ
- 「ベスト プラクティス（最善の実践原則）」と呼ばれるゆえんは、ソフトウェア業界で成功を収めている組織で広く利用されているから

4

ベスト プラクティスの例

- Rationalベスト プラクティス6か条
  - Rational Software（現IBM Rational事業部）
- 16 Critical Software Practices™
  - SPMN（Software Program Managers Network）
  - http://www.spmn.com/16CSP.html
- eXtream Programming（XP）プラクティス12か条

5

Rational ベスト プラクティス6か条

1. ソフトウェアを反復的に開発する
2. 要求を管理する
3. コンポーネントに基づくアーキテクチャを使用する
4. ソフトウェアを視覚的にモデリングする
5. 継続的にソフトウェアの品質を検証する
6. ソフトウェアへの変更を管理する

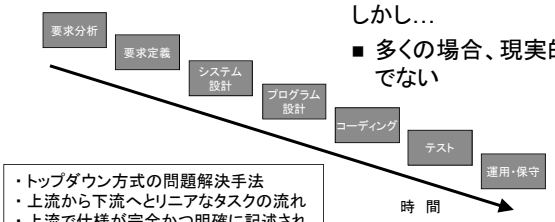
6

1. ソフトウェアを反復的に開発する

- 開発プロセスの二大モデル
  - ウォーターフォール型開発プロセス
  - 反復型開発プロセス
- ウォーターフォール型開発プロセスでは、リスクを後のフェーズに先送りする
  - リスクの潜在についての認識が欠けている
- 反復型開発プロセスでは、リスクの解消に積極的に取り組む
  - 解消できるリスクは先送りしない

7

ウォーターフォール型開発プロセス



しかし…

- 多くの場合、現実的でない

・トップダウン方式の問題解決手法
・上流から下流へとリニアなタスクの流れ
・上流で仕様が完全かつ明確に記述される必要がある
・仕様が上流工程のみで明確化できる場合には、最も合理的なアプローチ
・文書ベース

8

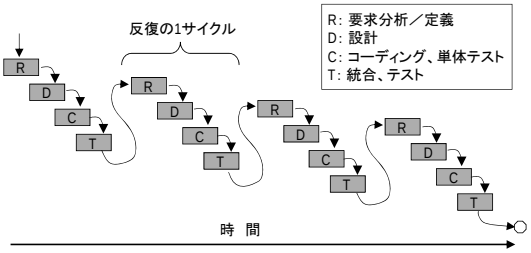
ウォーターフォール型モデルの問題

- 最初に要求を完全かつ明確に記述できると仮定している
  - ⇨そもそも要求は変化するため、プロジェクトの初期に要求を固定できると考えるのは非現実的
- 設計の妥当性を紙上の検討だけで検証できると仮定している
  - ⇨現実の問題に対してこれを可能にする技術はなく、人間による検証では見落しが発生する

⇨要求仕様も設計も適切に検証されないまま、それらに基づいて作業が進むため、リスクが先送りされ、プロジェクト終盤で致命的な問題が発生することになる

9

反復型開発モデル



出典：フィリップ・クルーシュテン著『ラショナル統一プロセス入門』

10

反復型開発モデルの特徴

- 対象とする問題の複雑さを段階的に緩和
  - ⇨扱う要求を少しずつ増やしながら、R→D→C→Tのサイクルを繰り返し、各サイクルでリスクの一部を解消する
  - ⇨各サイクルのマイルストーンとして、実行可能なプロトタイプを作成し検証する
- トップダウン方式（1つのサイクル内のR→D→C→T）とボトムアップ方式（次のサイクルへのステップアップ）の組み合わせ

11

反復型開発のメリット

- 致命的な誤解を早期に明らかにし、対処できる
- ユーザからのフィードバックが促進され、真の要求を引き出すことができる
- 開発チームはプロジェクトの最も重大な問題に専念できる
- プロジェクトのステータスを客観的に評価できる
- 要求、設計、実装の間の矛盾を早期に発見できる
- チームの作業量がライフサイクル全体に均等配分される
- 前の反復サイクルの教訓を活かせるので、プロセスが絶えず向上する
- プロジェクトの利害関係者が、プロジェクトのライフサイクル全体を通して、進捗ステータスを表す具体的な物証を得ることができる

12

反復型開発モデルの課題

- 最終的なプロダクトへの収束をどう保証するか
  - ⇨各反復が最初からの作り直しにならずに、インクリメンタルな繰り返しになるにはどうするか。
- 各反復で行う作業（扱う要求）と対処するリスクをどう選択するか
- 前の反復で見つかった重大な問題をどう解決するか

⇨個々の開発プロセス モデルごとにその手段が用意される必要がある（例: Rational Unified Process®）

13

2. 要求を管理する

- 小規模なシステムやよく理解されているドメイン以外では、開発に着手する前に、システムに対する要求を完全かつ網羅的に記述することは不可能
- 要求は時間と共に変化する

⇨要求を体系化し、変化を検出し、影響を評価して他の成果物に伝播させる

- ツールによるサポートが必須

14

要求管理のメリット

- 要求の検索、優先順位づけ、フィルタリング、追跡ができる
- 矛盾を容易に発見できる
- 変更の影響を適切に伝播させることができる

⇨ツールによるサポートが不可欠（人手では十分な管理は難しい）

15

3. コンポーネントに基づくアーキテクチャを使用する

- アーキテクチャは、プロジェクトの利害関係者（エンドユーザと開発チーム メンバー）の異なる視点を管理するための最も重要な成果物
  - これをベースに、反復と追加（インクリメント）に基づく開発を制御できる
- コンポーネントを利用することで、部品のより大規模な再利用が可能になる

16

コンポーネントに基づくアーキテクチャのメリット

- 変更に強いアーキテクチャになる
- 変更の対象となるシステム要素を、それらが果たすべき役割に応じて明確に分離できる
- 標準化されたフレームワーク（COM+、EJBなど）と市販コンポーネントにより、再利用が容易になる
- コンポーネント単位に構成管理できる
- ビジュアル モデリング ツールを利用すれば、開発の自動化が可能になる

17

4. ソフトウェアを視覚的にモデリングする

- モデルとは、特定の視点からシステムを包括的に記述することで現実世界を単純化したもの
- モデリング対象のシステムをよりよく理解することが目的
- ビジュアル モデリング ツールを利用すれば、モデルの管理が容易になる

18

ソフトウェアの視覚的モデリングのメリット

- ユースケースとシナリオによって、振る舞いに関する仕様を明確に表現できる
- ソフトウェアの設計を明確に理解できる
- モジュール化されていない、柔軟性が欠けているといったアーキテクチャ上の問題を検出できる
- モデルの詳細度を必要に応じて変更できる
- 設計の矛盾が比較的容易に明らかになる
- 標準化されたモデリング言語を使用すれば、ライフサイクル全体にわたって、モデルの管理やコミュニケーションが容易になる
  - UMLが事実上の標準になっている

19

5. 継続的にソフトウェアの品質を検証する

- システムの完成後にシステムの問題を発見して修正しようとすると、100～1000倍のコストがかかる

⇨機能、信頼性、性能の観点から、システムの品質を継続的に評価することが不可欠

- 反復型開発プロセスでは、各反復サイクルで具体的なテストを行うので、これに適している

20

品質の継続的検証のメリット

- プロジェクトのステータスが客観的に評価される
- 要求、設計、実装の間の矛盾が明らかになる
- 最もリスクの高い部分にテストと検証が集中するため、その部分の品質と効率が向上する
- 欠陥を早期に発見できるので、それらを修正するためのコストを大幅に削減できる
- 自動テスト ツールを利用すれば、機能、信頼性、性能のテストが可能になる

21

6. ソフトウェアへの変更を管理する

- 開発者は一般に異なる組織に所属しており、それらの要員をうまく連携させなければ、開発プロセスは混乱に陥る
- ソフトウェア開発の成果物への変更を管理するための反復可能なワークフローを確立して、開発チーム内の作業や成果物を取りまとめることが必要

⇨プロジェクトの優先度やリスクに基づいたリソースの有効な割り当てが可能になる

⇨ 変更に伴う作業を積極的に管理できる

22

変更管理のメリット

- 要求の変更に対処するためのワークフローが定義され、反復可能になる
- 変更依頼に基づくことで、コミュニケーションがより明確になる（履歴も残る）
- 作業現場が分離されるため、並行して作業を行っているチームメンバー間の干渉が減る
- 変更率の統計がプロジェクト ステータスの客観的評価尺度になる
- すべての成果物が共有されるので、整合性が保たれる
- 変更による影響を評価し、制御できる

⇨ ツールによるサポートが不可欠（人手では十分な管理は難しい）

23

開発方法論、支援ツール

ソフトウェア開発プロセスの役割

- チームの開発作業の順序に関するガイダンスを提供
- どの成果物をいつ開発すべきかを規定
- 個々の開発者の作業とチーム全体の業務を統轄
- プロジェクトのプロダクトとプロセスを監視・測定するための基準を提供

⇨ 明確に定義された開発プロセスがあってこそ、ソフトウェア開発のベスト プラクティスが実践可能になり、促進される

25

利用可能な開発プロセスの例

- RUP®（Rational Unified Process®）
- eXtreme Programming（XP）

26

RUP®（Rational Unified Process®）

- Rational Software社（現IBM Rational事業部）が提供している製品
- 実態はWebベースの知識ベースとプロセス ガイドライン
  - 紙ベースでないところが重要
- Rationalブランドのソフトウェア開発ツール群（ビジュアル モデリング、要求管理、変更管理、構成管理など）と統合されている

27

eXtreme Programming（XP）

- Kent Beckらが考案し提唱しているソフトウェア開発手法で、アジャイル ソフトウェア開発（Agile Software Development）手法と総称される一連の手法の先駆けとなったもの
- 開発の初期段階の設計よりもコーディングとテストを重視し、常にフィードバックを行ってコードを修正・再設計していくことを基本としている
- 開発チームが共有すべき価値として、コミュニケーション（communication）、シンプルさ（simplicity）、フィードバック（feedback）、勇気（courage）の4つを示し、経験に基づいた具体的な実践項目（プラクティス）を12か条挙げている
- 10人程度くらいまでの小チームに適用するのが最も適していると言われ、小～中規模のソフトウェアの開発に向いた手法とされている

28

ソフトウェア ライフサイクル管理を実現する開発ツール群

- 要求管理ツール（要求と成果物のリンク）
- 変更管理ツール（成果物の一貫性を保つ）
- 構成管理ツール（種々のバージョンを管理）
- 統合開発環境（IDE）
- 統合テスト ツール
- コミュニケーション支援ツール
  - メーリング リスト
  - ディスカッション グループ

29

統合開発環境（IDE）の主な機能

- UML ビジュアル モデリング
- フォワード/リバース エンジニアリング
- 入力支援機能付きプログラミング エディタ
- デバッガ
- ソースコード検査＆メトリクス測定
- リファクタリング
- 単体テスト
- ドキュメント生成

30

まとめ

- ソフトウェア開発のRational ベスト プラクティス6か条を紹介した
  1. ソフトウェアを反復的に開発する
  2. 要求を管理する
  3. コンポーネントに基づくアーキテクチャを使用する
  4. ソフトウェアを視覚的にモデリングする
  5. 継続的にソフトウェアの品質を検証する
  6. ソフトウェアへの変更を管理する
- 明確に定義された開発プロセスがあってこそ、ソフトウェア開発のベスト プラクティスが実践可能になり、促進される
- 開発プロセスの例: RUP、XP

31

参考文献

- Philippe Kruchten,「The Rational Unified Process: An Introduction, Second Edition」, Addison-Wesley Pub Co.
  - （邦訳:「ラショナル統一プロセス入門　第2版」, ピアソン・エデュケーション）
- Kent Beck, “Extreme Programming Explained: Embrace Change”, Addison-Wesley Pub Co.
  - （邦訳:「XPエクストリーム・プログラミング入門―ソフトウェア開発の究極の手法」, ピアソン・エデュケーション）

32