ソフトウェア工学

11: 管理 — プロジェクト管理と品質管理

理工学部 経営システム工学科
庄司　裕子

今回のテーマ

管理 — プロジェクト管理と品質管理

- ソフトウェア開発の管理的側面
- プロダクトの管理
- プロセスの管理
- 要員の管理

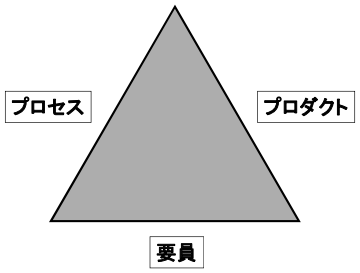
管理 — プロジェクト管理と品質管理

⇨ソフトウェア開発の管理的側面
- プロダクトの管理
- プロセスの管理
- 要員の管理

ソフトウェア開発の三位一体

- 誰が（要員）
- 何を（プロダクト）
- どのように（プロセス）

開発するか



ソフトウェア工学が目指すものは何であったか

- ソフトウェア/システムの以下の側面を向上させること
  - プロダクトの品質
  - プロセスの生産性

つまり、

- 要求仕様を満たすシステムを、より少ない工数で開発できる方法論と技術の開発がテーマ

ソフトウェア開発におけるもう1つの重要なファクタ

- ヒューマン ファクタ
  - 開発要員の能力（属人性）
  - ステークホルダ（利害関係者）の利害の交錯
- 純粋な技術論ではないため一般にソフトウェア工学のテーマとはみなされないが、ソフトウェア開発の成否に大きく影響する

ソフトウェア開発の管理的側面

- プロダクトの管理
  - 品質管理
  - 変更管理
- プロセスの管理
  - プロジェクト管理
- 要員の管理
  - 要員管理

7

管理 — プロジェクト管理と品質管理

✓ソフトウェア開発の管理的側面
⇨プロダクトの管理
- プロセスの管理
- 要員の管理

8

プロダクトとは

- プロセスから生成されるアーティファクト（artifact）
  - 最終成果物
    - 製品、運用システム、…
  - 中間成果物
    - 要求仕様、設計、プロトタイプ、ソースコード、…

9

プロダクトの管理

- 品質管理
  - ☞ 最終成果物の品質
- 変更管理
  - ☞ 最終／中間成果物の管理
    - 要求管理
    - 構成管理
    - …

10

ソフトウェアの品質

- ソフトウェアの品質を客観的に定義するための基準
  - 機能と性能に関する要求仕様
  - 開発にかかるコストと期間

⇨要求仕様を満たし短期間に開発できる安価なソフトウェアが「高品質な」ソフトウェア

- これらの基準値が顧客の要望にどれだけ沿っているかを、各種のソフトウェア評価尺度を用いて定量的に表す

11

品質管理

- 品質管理システム（Quality Management System：QMS）
  - 「プロセスの監視と改善を通して、そのプロセスに従うメンバーが高品質なソフトウェアを生産する確率を高める」という考え方
    - プロジェクトごとに品質計画を立て、品質管理タスクを規定する
    - 組織における品質監査の実施と、改善の提案、承認、および実行を指定する
- トータル品質管理（Total Quality Management：TQM）
  - 「承認されたプロセスにメンバーがどの程度準拠しているかよりも、製品の生産に関わっている要員の品質の高さを保証するほうが、最終製品の品質に大きく影響する」という考え方
    - メンバーのトレーニングが不可欠

12

変更管理

- 中間成果物にも最終成果物にも適用される
  - 最終成果物を対象にした場合は通常、バージョンアップに相当する
  - 中間成果物の変更管理は、反復型開発プロセスにとって不可欠
- 変更管理の中核機能
  - 変更の伝播
  - 矛盾（不整合）の検出
- 特に大規模なシステムの場合、これを人手で行うのは非常に手間がかかり、不正確になるおそれがある

⇨ ベンダからツールが提供されている
  - 開発環境の一部になりつつある

13

管理 — プロジェクト管理と品質管理

✓ソフトウェア開発の管理的側面
✓プロダクトの管理
⇨プロセスの管理
- 要員の管理

14

プロセスの管理

- プロジェクト管理
  - 実施前
    - プロジェクトの立案（コスト見積り、リソースの調達、スケジュール、品質計画、要員配置、タスクの割り当て）
    - 利害関係者同士のコミュニケーション パスの確立
    - 開発環境（再利用／コミュニケーション環境を含む）のセットアップ
  - 実施中
    - 進捗の測定
    - リスクの特定、分析、解消
    - 利害関係者間の調整
    - コンポーネントの再利用の促進

15

プロジェクトの見積りと進捗

反復型開発プロセスでは、
- 「〇〇フェーズの終了」というマイルストーンは設定できない
- ある機能を実現する反復サイクルの完成をマイルストーンとして設定できる

☞機能がユースケースで記述されていれば、そこから工数を見積り、進捗を判断できる

16

コスト見積りモデル

- プロジェクトのコスト見積りに定量的モデルが用いられることがある
  - COCOMO（COnstructive COst MOdel）
  - COCOMO-Ⅱ
  - …
- COCOMOの見積り式
  - 人月 Effort = C1 EAF (Size)^{P1}
  - 月数 Time = C2 (Effort)^{P2}

17

リスクの特定、分析、解消

反復型開発プロセスでは
- 各反復サイクルでリスクを評価し解消するため、リスクには対処しやすい

ウォーターフォール型開発プロセスでは
- 実行可能な成果物が出来上がるプロジェクト後期になるまで致命的なリスクが顕在化しないおそれがある
- 文書上はすべて順調であるかのように見える

18

ソフトウェアプロセスの成熟度を評価するための手段

- SEI（Software Engineering Institute）のプロセス成熟度モデル（Capability Maturity Model：CMM）
  - 重要なプロセス領域（Key Process Area：KPA）を組織がサポートしているかどうかで、5段階のソフトウェアプロセス成熟度を定義
    - レベル1：明確に定義されていないプロセス
    - レベル2：再現可能なプロセス
    - レベル3：定義されたプロセス
    - レベル4：管理されたプロセス
    - レベル5：最適化されたプロセス

19

管理 — プロジェクト管理と品質管理

✓ソフトウェア開発の管理的側面
✓プロダクトの管理
✓プロセスの管理
⇨要員の管理

20

要員の管理

- 要員管理
  - チーム構成
    - ヒューマン リソース（人材）の適正配置
  - チーム ダイナミクス
    - 要員の動的再配置
  - 要員の育成
    - 長期的なビジョン

21

チーム構成

- ウォーターフォール型開発プロセスであろうと反復型開発プロセスであろうと、タスクのタイプによって必要な人材とその数は異なる
- 各タスクに必要なロール
  - 要求分析／定義：アナリスト（コンサルタント）
  - システム設計：アーキテクト
  - プログラム設計～コーディング：プログラマ
  - テスト：テスター
  - プロジェクト管理：プロジェクト管理者（プロマネ）

22

各ロールに必要な能力（理想）

- アナリスト（コンサルタント）
  - コミュニケーション スキル
  - ドメイン（業務）知識
  - 分析的思考（analytic mind）
- アーキテクト
  - 先端技術に関する幅広い実用的知識と経験
  - 構成的思考（synthetic mind）
- プログラマ
  - 使用するプログラミング言語に関する深い知識とコーディング経験
  - データ構造とアルゴリズムに関する幅広い実用的知識
- テスター
  - 綿密さ
  - 細部重視（detail-oriented）
- プロジェクト管理者
  - システム開発の管理に関する豊富な知識と経験
  - 洞察力
  - 実行力
  - コミュニケーション スキル
  - 調整力

23

プロジェクト管理者とプロジェクト リーダ

両者はよく混同されるが、

- プロジェクト管理者とプロジェクト リーダの役割は、本質的に異なる
  - 管理者はなだめる人、リーダは奮起させる人
  - 管理者は検証する人、リーダは推進する人
  - 管理スキルはある程度までは習得できるが、リーダーシップは習得できるものかどうか不明
- 必ずしも同一の要員が担当する必要はない（同一の要員であるべきでないとする考え方もある）

24

プロジェクト リーダに必要な資質

- リーダはタスクに結び付いているものではなく、資質さえ備わっていれば、どのメンバーが担当してもよい
- 必要な資質
  - リーダーシップ（カリスマ性）
  - システム開発に関する豊富な知識と経験
  - 洞察力と企画力
  - 実行力
  - 情熱

25

要員の適正配置

- 実用的知識や経験の有無など、年齢と相関する側面もあるが、原則的に各ロールは年齢とは別個に考えるべき
- コーダーに始まり、プログラマ、SE、上級SEへとプロモートする「出世魚」モデルは本来あまり根拠がなく、特に技術の陳腐化の激しい近年においてはすでに意味を失っている

⇨真の能力に基づいた配置が必要

26

必要な要員の数

タスク	ロール	要員数
要求分析／定義	アナリスト（コンサルタント）	少数
システム設計	アーキテクト	少数
プログラム設計～コーディング	プログラマ	一般に多数
テスト	テスター	中間
プロジェクト管理	プロジェクト管理者	1人

27

要員数の変化

- ウォーターフォール型開発プロセス
  - タスク タイプが1つずつ逐次的に進むため、チームのサイズ（必要な要員数）は時間的に変化する
- 反復型開発プロセス
  - 程度の差こそあれ、どの時点でもすべてのタスク タイプが実行されるため、チームのサイズは時間的にはそれほど変動しない

28

要員の追加投入のジレンマ

- 問題が発生したときや、進捗が遅れているときなど、要員を追加投入することがある
- その結果、かえって問題が複雑化したり、余計に進捗が遅れたりすることがある
- その原因は…
  - 要員の増加により要員間で必要なコミュニケーションが増え、要員増加の効果を相殺する
  - 新任要員の習熟に時間がかかり、途中で投入しても即戦力にならない

29

チーム構成の例：チーフ プログラマ制

- 少数精鋭主義によるソフトウェア開発チーム
- 1人の卓越したプログラマがチーフ プログラマとして、設計からコードの中核部分をすべて担当し、他のメンバーはチーフ プログラマの補佐に徹する
- チーム メンバー間のコミュニケーションはチーフ プログラマと各メンバー間でのみ行えばよく、効率的
- 大規模なシステムでも、かなりの部分は安全性や耐障害性のメカニズムであり、担当すべき中核部分はそれほど多くないので、1人ですべてカバーするのも可能

しかし、

☞ それほど有能なスーパー プログラマは少ないため確保が難しく、極めて高い能力が求められるため育成も容易ではない

30

現実のチーム構成

- チーフ プログラマ制やそこから派生したチーム構成はソフトウェア業界の主流とはならず、部分的に適用されているのみ（特に、言語やツールなど。ソリューション系ではない）
- 現実には、モジュール構成と似た組織のチームになることが多い
  - 利点： モジュール間の独立性が高ければ、要員間のコミュニケーションも少なくて済み、生産性が上がる
  - 欠点： モジュール構成に合わせてチームを作るのはよいが、チームに合わせてモジュール構成が作られることが往々にしてあり、いびつな設計になるおそれがある

31

まとめ

- ソフトウェア開発プロジェクト全体に及ぶ管理的側面の概要を解説した
  - プロダクトの管理
  - プロセスの管理
  - 要員の管理
- 要員の管理は、ソフトウェア工学というよりはむしろ組織論的な問題であるが、ソフトウェア開発の多くの部分が未だ人間の創造活動に依存しているため、属人性が強く、ヒューマン ファクタを無視できない

32

参考文献

- 有沢誠：「岩波コンピュータサイエンス ソフトウェア工学」, 岩波書店
- Perdita Stevens with Rob Pooley, “Using UML: Software Engineering with Objects and Components, Updated Edition”, Pearson Education
  - （邦訳：「オブジェクト指向とコンポーネントによるソフトウェア工学 — UMLを使って —」ピアソン・エデュケーション）
- Walker Royce : “Software Project Management: A Unified Framework”, Addison-Wesley Pub Co.
  - （邦訳：「ソフトウェアプロジェクト管理： 21世紀に向けた統一アプローチ」, アジソン・ウェスレイ・パブリッシャーズ・ジャパン）

33