Contents

APMBOK2004

著作権 3
*PMBOK*とは 4
*PMBOK*の構成 5
*39*のプロセス 6
開始プロセス・グループ 7
計画プロセス・グループ 8
実行プロセス・グループ 9
制御プロセス・グループ 10
終結プロセス・グループ 11
*APMBOK*とは 12

開始プロセス・グループ

2.1 立ち上げプロセス 14
プロジェクト・フォーミュレーション 15
プロジェクト・ビジョン 16

計画プロセス・グループ

2.2 スコープ計画プロセス 18
2.3 スコープ定義プロセス 19
ストーリ, フィーチャ, ... 20
3.1 活動定義プロセス 21
タスク 22
3.2 活動配列プロセス 23
*APM*における活動ネットワークの問題点 24
3.3 活動期間見積もりプロセス 25
プロジェクト速度 26
3.4 スケジュール策定プロセス 27
イテレーション期間 28
4.1 資源計画プロセス 29
*APM*における資源 30

4.2 コスト見積もりプロセス 31
APMにおけるコスト見積もり 32
4.3 コスト予算化プロセス 33
1.1 プロジェクト計画策定プロセス 34
獲得価値管理とAPM 35
プロジェクト・ホスティング・ツール 36
SPMP (IEEE1058) 37
SPEMによるプロセス記述 38
5.1 品質計画プロセス 39
アジャイル・プロジェクトにとっての品質 40
ISO/IEC9126 品質標準 41
6.1 組織計画プロセス 42
APMにおける組織, 人 43
6.2 スタッフ調達プロセス 44
APMにおける配員 45
7.1 コミュニケーション計画プロセス 46
APMにおけるコミュニケーション 47
8.1 リスク管理計画プロセス 48
8.2 リスク特定プロセス 49
APMにおけるリスク特定 50
8.3 定性的リスク分析プロセス 51
8.4 定量的リスク分析プロセス 52
APMにおけるリスク分析 53
8.5 リスク対応計画プロセス 54
リスク対処の方法 55
効果図式 56
9.1 調達計画プロセス 57
コンポーネント /ツール調達の問題点 58
持続可能な調達 59

9.2 要請計画プロセス 60
アジャイル・プロジェクトにおける契約 61
実行プロセス・グループ
1.2 プロジェクト計画実行プロセス 63
アジャイル・プロジェクト実行の原則 64
5.2 品質保証プロセス 65
6.3 チーム結成プロセス 66
アジャイルなチーム 67
ファシリテータ, コーチ 68
ワークアウト 69
7.2 情報配布プロセス 70
アジャイル・プロジェクトにおける情報共有 71
コミュニティ・サイト 72
情報壁 73
コロケーション 74
9.3 要請プロセス 75
9.4 調達先選定プロセス 76
9.5 契約事務プロセス 77
アジャイル・プロジェクトにおける外注管理 78
制御プロセス・グループ
7.3 実績報告プロセス 80
アジャイルな進捗報告の原則 81
1.3 全体的な変更制御プロセス 82
アジャイルな変更制御 83
プロジェクト・ホスティング・サイト 84
2.4 スコープ検証プロセス 85
アジャイルなスコープ検証 86
2.5 スコープ変更制御プロセス 87
アジャイル・スコープ制御の原則 88

3.5 スケジュール制御プロセス *89*
アジャイル・プロジェクトのスケジュール制御 *90*
4.4 コスト制御プロセス *91*
5.3 品質制御プロセス *92*
アジャイルな品質制御 *93*
8.6 リスクの監視と制御プロセス *94*
アジャイルなリスク監視 *95*
不測の事態のための予備 *96*

終結プロセス・グループ

9.6 契約完了プロセス *98*
アジャイルな検収 *99*
7.4 完了手続きプロセス *100*
レトロスペクティブ *101*
アジャイル・プロジェクトの終了 *102*

参照

1. 統合管理 *104*
2. スコープ管理 *105*
3. 時間管理 *106*
4. コスト管理 *107*
5. 品質管理 *108*
6. 人材管理 *109*
7. コミュニケーション管理 *110*
8. リスク管理 *111*
9. 調達管理 *112*
10. 知識管理 *113*

著作権 3
PMBOKとは 4
PMBOKの構成 5
39のプロセス 6
開始プロセス・グループ 7
計画プロセス・グループ 8
実行プロセス・グループ 9
制御プロセス・グループ 10
終結プロセス・グループ 11
APMBOKとは 12

著作権

このドキュメントの著作権はすべて有限会社メタボリックスに属します. このドキュメントの全体または一部分を著作権者に許可なく第3者に配布または譲渡することはできません.

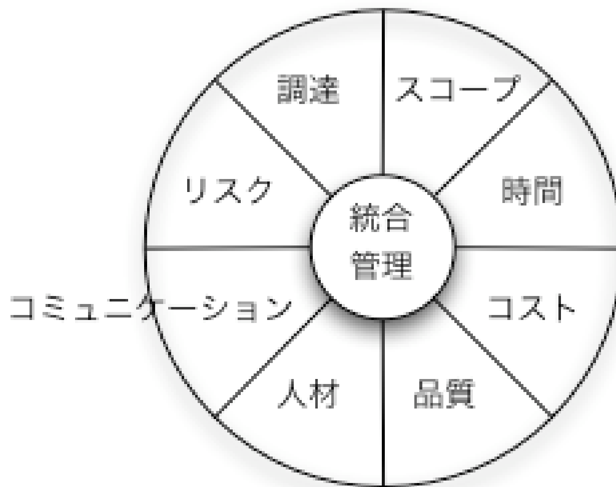
PMBOKとは

- PMBOK (Project Management Body of Knowledge)
- プロジェクト管理に関する知識体系
- 米PMI (Project Management Institute) が発行
  - @ http://pmi.org/
  - IEEE CS/SC (Computer Society / Standard Committiee) も批准している
    - IEEE 1490
  - 現在の最新版は2000 edition
  - 英語版はamazon.co.jpなどから購入可能
    - @ http://www.amazon.co.jp/exec/obidos/ASIN/1880410230/ref=sr_aps_eb_/249-3000937-4969931
  - 邦訳版はamazon.co.jpなどから購入可能
    - @ http://www.amazon.co.jp/exec/obidos/ASIN/1930699204/249-3000937-4969931
  - その他の日本語の関連書籍も最近は多数出版されている
- プロジェクト管理の標準的なガイドラインとなっている
- ソフトウェア開発プロジェクトを対象にしたものではなく, すべてのプロジェクトを対象にしている
  - 本来は建築などモノ作りプロジェクトが中心
  - 最近はソフトウェア開発プロジェクトも考慮されるようになってきているが, 基本的にはウォータフォール的な世界であることは否めない
- PMIが認証するPMP (Project Management Professional)のテキスト
  - CMM / CMMIとは違って
    - プロセス改善が主対象ではない
    - 組織のアセスメントが目的ではない
    - プロジェクト管理者のアセスメントが目的である
    - 利用できる方法論が具体的に挙げられている

PMBOKの構成

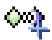
9つの知識分野

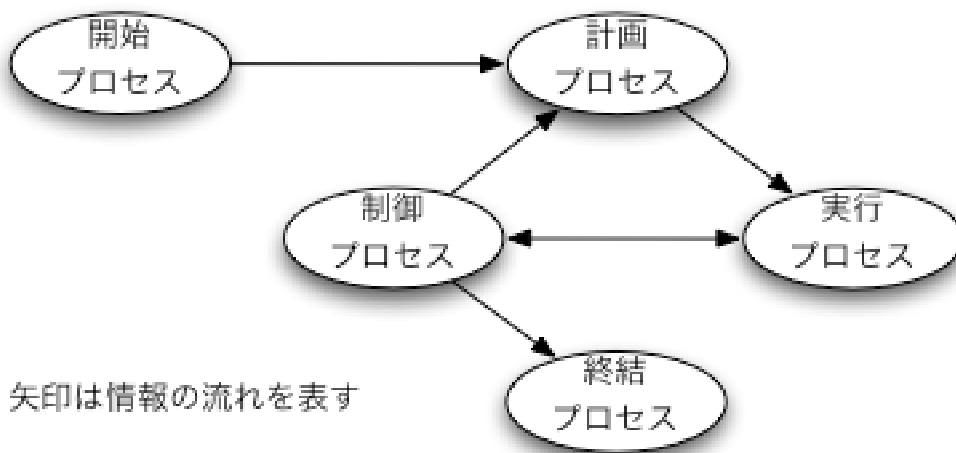
統合, スコープ, 時間, コスト, 品質, 人材, コミュニケーション, リスク, 調達



5つのプロセス・グループ

開始, 計画, 実行, 制御, 終結





39のプロセス

それぞれのプロセスごとに

入力

ツールと技法

出力

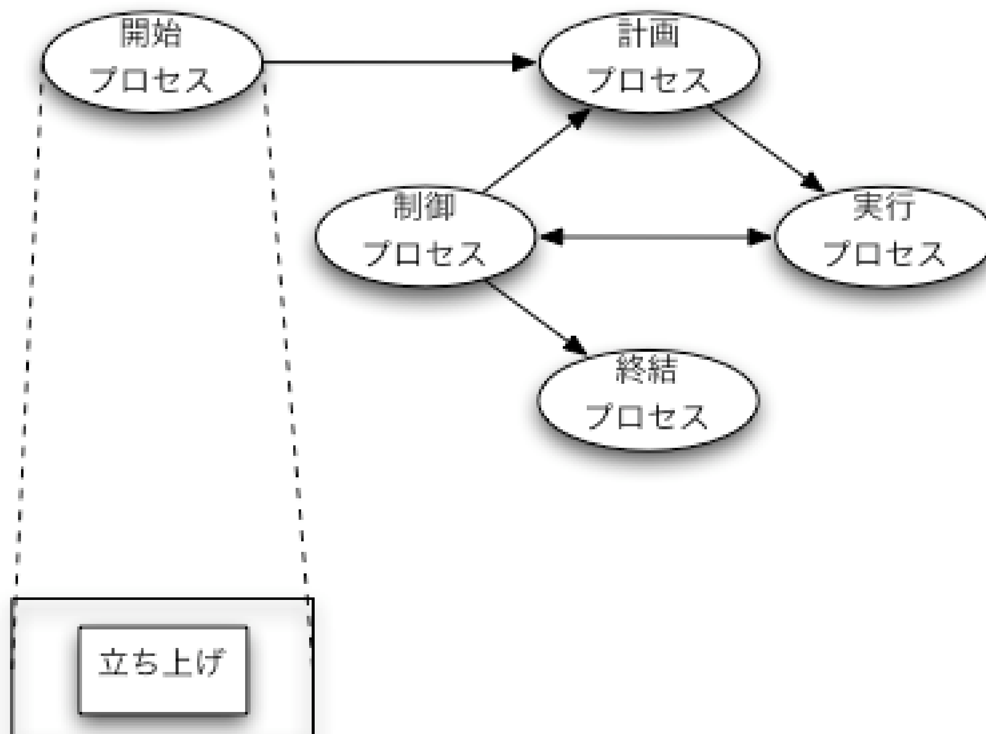
APMBOK2004
39のプロセス

全体で39のプロセスが
9つの知識分野
5つのプロセス・グループ
に分類されている

	開始	計画	実行	制御	終結
1. 全体管理		1.1 プロジェクト計画策定	1.2 プロジェクト計画実行	1.3 全体の変更制御	
2. スコープ管理	2.1 立ち上げ	2.2 スコープ計画 2.3 スコープ定義		2.4 スコープ検証 2.5 スコープ変更制御	
3. 時間管理		3.1 活動定義 3.2 活動配列 3.3 活動期間見積もり 3.4 スケジュール策定		3.5 スケジュール制御	
4. コスト管理		4.1 資源計画 4.2 コスト見積もり 4.3 コスト予算化		4.4 コスト制御	
5. 品質管理		5.1 品質計画	5.2 品質保証	5.3 品質制御	
6. 人材管理		6.1 組織計画 6.2 スタッフ調達	6.3 チーム結成		
7. コミュニケーション管理		7.1 コミュニケーション計画	7.2 情報配布	7.3 実績報告	7.4 運営完了
8. リスク管理		8.1 リスク管理計画 8.2 リスク特定 8.3 定性的リスク分析 8.4 定量的リスク分析 8.5 リスク対応計画		8.6 リスク監視と制御	
9. 調達管理		9.1 調達計画 9.2 要請計画	9.3 要請 9.4 調達先選別 9.5 契約事務		9.6 契約完了

開始プロセス・グループ

2.1 立ち上げ



計画プロセス・グループ

計画プロセス・グループ

- 1.1 プロジェクト計画策定
- 2.2 スコープ計画
- 2.3 スコープ定義
- 3.1 活動定義
- 3.2 活動配列
- 3.3 活動期間見積もり
- 3.4 スケジュール策定
- 4.1 資源計画
- 4.2 コスト見積もり
- 4.3 コスト予算化
- 5.1 品質計画
- 6.1 組織計画
- 6.2 スタッフ調達
- 7.1 コミュニケーション計画
- 8.1 リスク管理計画
- 8.2 リスク特定
- 8.3 定性的リスク分析
- 8.4 定量的リスク分析
- 8.5 リスク対応計画
- 9.1 調達計画
- 9.2 要請計画

計画プロセス・グループ

実行プロセス・グループ

実行プロセス・グループ

1.2 プロジェクト計画実行

5.2 品質保証

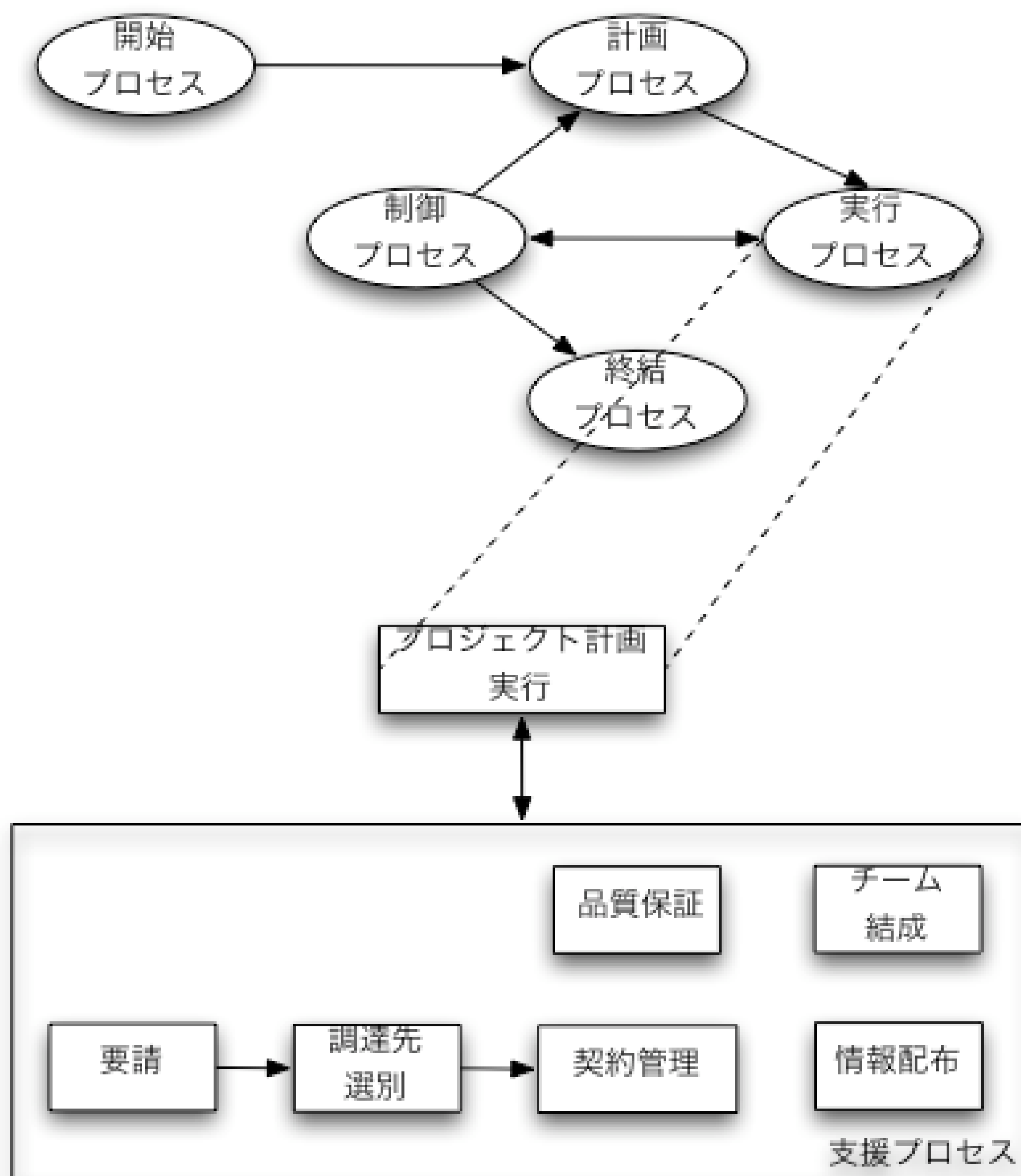
6.3 チーム結成

7.2 情報配布

9.3 要請

9.4 調達先選別

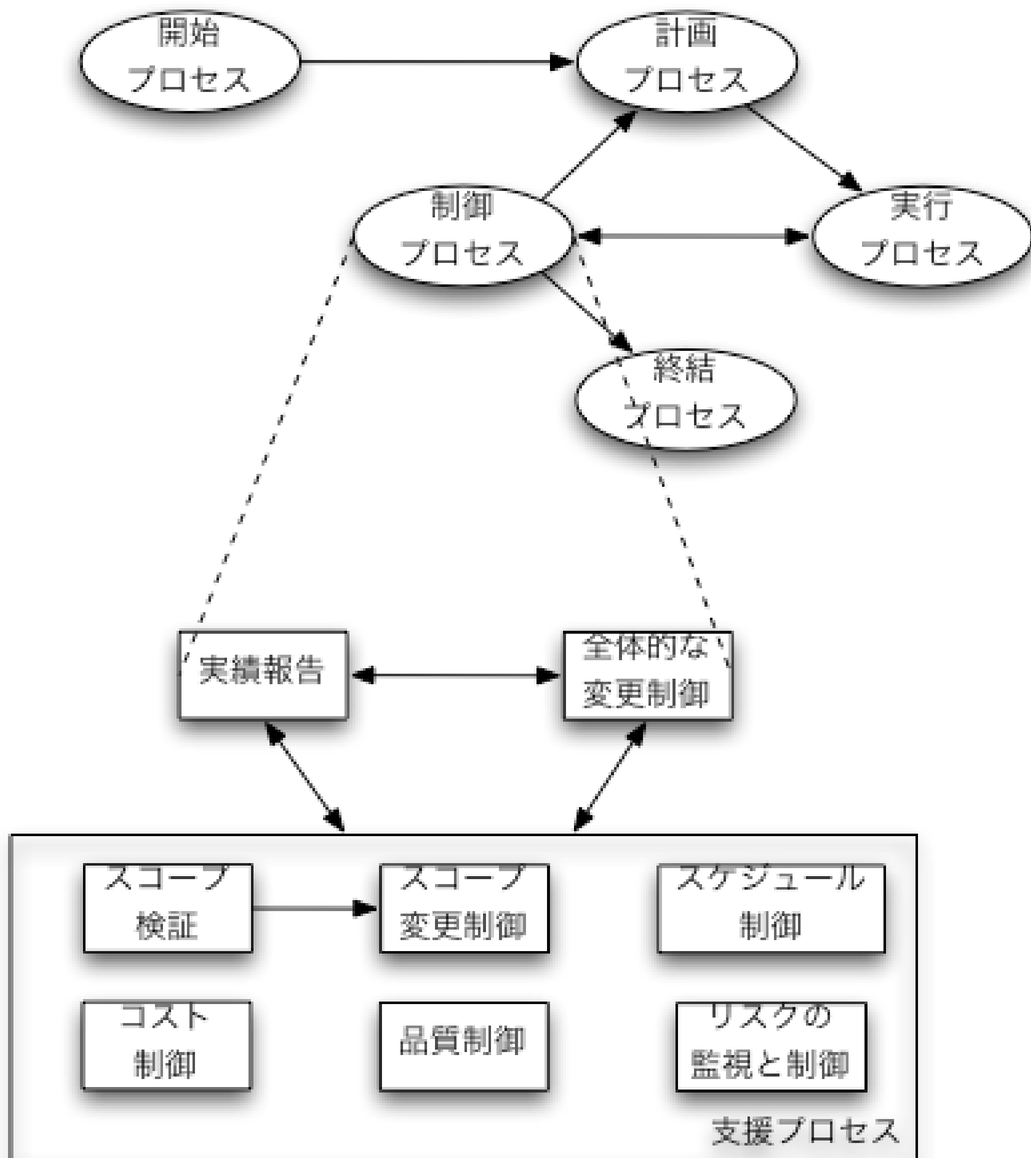
9.5 契約事務



制御プロセス・グループ

制御プロセス・グループ

- 1.3 全体の変更制御
- 2.4 スコープ検証
- 2.5 スコープ変更制御
- 3.5 スケジュール制御
- 4.4 コスト制御
- 5.3 品質制御
- 7.3 実績報告
- 7.6 リスク監視と制御

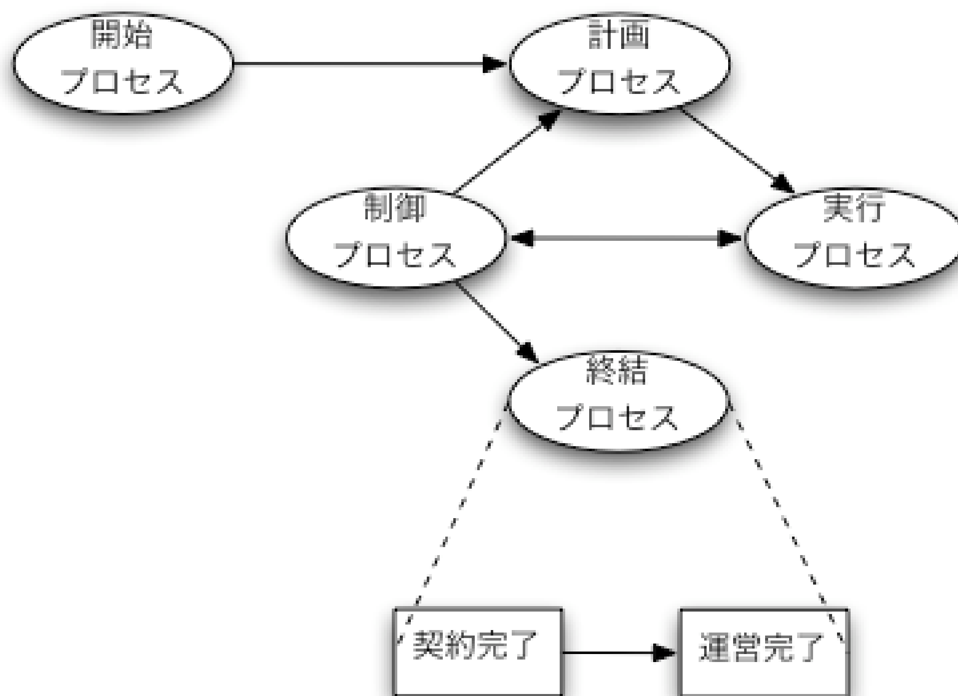


終結プロセス・グループ

終結プロセス・グループ

7.4 運営完了

9.6 契約完了



APMBOKとは

- Agile Project Management Body of Knowledge
- アジャイル・プロジェクト管理におけるPMBOK
  - アジャイル・プロジェクト管理に関する知識を体系的にまとめることができないか, という試み
  - 最近アジャイル・プロセス協議会のアジャイル・プロジェクト・マネジメントワーキング・グループ (APMWG) でも同様の試みが行われている
    - @ http://www.agileprocess.jp/
    - @ http://groups.yahoo.co.jp/group/apmwg/
  - 特に認証やアセスメントに用いるのが目的ではない
- PMBOKをアジャイル・プロジェクト管理に適用するときの問題点
  - PMBOKの全体的なプロジェクト管理プロセスはアジャイル・プロジェクト管理に適用できるか
  - PMBOKの個々のプロセスはアジャイル・プロジェクト管理に適用できるか
  - PMBOKの個々のプロセスの入力 / ツールと技法 / 出力はアジャイル・プロジェクト管理に適用できるか
  - アジャイル・プロジェクト管理にとって重要であるにもかかわらず, PMBOKに抜けている知識分野 / プロセスはあるか
- とりあえず
  - 全体的なプロジェクト管理プロセスは, アジャイル・プロジェクト管理でもほぼ同様と考えてよい
  - 個々のプロセスは, アジャイル・プロジェクト管理に適していないものも含まれていると考えられる
    - それに関してはその場で触れることにする
  - 個々のプロセスの入力 / ツールと技法 / 出力は必要ならばアジャイル・プロジェクト管理の側面から見直す
  - PMBOKにはないが, アジャイル・プロジェクト管理にとって重要な知識分野 / プロセスがあれば, 最後に触れる
- APMBOKは既知のプロジェクト管理技法, さまざまなアジャイル開発プロセスが提唱するプラクティスをまとめたものになる

開始プロセス・グループ 13

2.1 立ち上げプロセス *14*
プロジェクト・フォーミュレーション *15*
プロジェクト・ビジョン *16*

2.1 立ち上げプロセス

プロジェクトあるいはフェーズを確立する

入力

- プロダクト記述
- 戦略計画
- プロジェクト選定基準
- 実績情報

ツールと技法

- プロジェクト選定手法
- 専門家による判断

出力

- プロジェクト憲章
- プロジェクト・マネージャの選定と配属
- 制約条件
- 前提条件

プロジェクト・フォーミュレーション

アジャイル・プロジェクト管理では, 問題解決型プロジェクトあるいは価値創造型プロジェクトを対象としており, プロジェクト立ち上げ時点ではプロジェクトの目的や形態は明確になっていないのが普通である.

「プロジェクト・フォーミュレーション」はそのようなプロジェクト立ち上げ時に行うプロジェクト確立の儀式である.

　XPなどではアーキテクチャ・スパイキングと呼ばれることもある

プロジェクト・フォーミュレーションの期間をあらかじめ決める. 通常は予定される全プロジェクト (あるいはフェーズ) 期間の10~20%.

　例えば3ヶ月のプロジェクトならば, 1~2週間. 困難が予想されるプロジェクトでは長めに.

プロジェクト・フォーミュレーションに参加するメンバは, プロジェクトの核になるメンバを中心とする.

　人数に制約はないが, あまり大人数でない方がよい.

　特に初心者は特別な理由がなければ参加させない. ただし「修行」的意味があれば参加させてもよい. いずれにしろ, プロジェクト・フォーミュレーションでは初心者に手を掛けている余裕はない.

必要ならばプロジェクト・フォーミュレーションのための予算も決めておく.

プロジェクト・フォーミュレーションではプロジェクトがこれから行うべきことを探るために (予算内で) 何をやっても構わない. 例えば

　Webサーフィン

　オープン・ソース製品の調査, インストール, 試用

　顧客やユーザへのインタビュー

　専門家の訪問, インタビュー

　フィージビリティ調査のためのコーディング

　資料収集, 読書

　……

プロジェクト・フォーミュレーションでは, 計画は立てない.

個々のメンバが作成すべき成果物は一切ない.

ただしメンバ全員で基本的に毎日ミーティングを行う.

　参加者は何を言っても構わない.

　ある種のブレーンストーミングである.

　各メンバがこれから次のミーティングまでに行うべきことを自分の責任において決める.

　前回のミーティング以降自分がしたことを報告する.

プロジェクト・フォーミュレーション

- プロジェクト・フォーミュレーションはもう十分であると全員の意見が一致したら, 期間中であってもプロジェクト・フォーミュレーションを終了する.

プロジェクト・フォーミュレーションの最後の1日か2日を使って, 次に述べるプロジェクト・ビジョンを作成する.

- もしこの程度の期間内にプロジェクト・ビジョンを作成できないとしたら, それはプロジェクトの困難さを表していると考える. この場合の選択肢は以下の通り.
  - プロジェクトをキャンセルする
  - プロジェクト・フォーミュレーションを一回だけ延長する. 延長期間は本来のプロジェクト・フォーミュレーション期間を超えない (できれば半分程度) .
  - 高いリスクを承知の上で, プロジェクト・ビジョンなし (あるいは未完成) でプロジェクトを開始する.

プロジェクト・ビジョン

アジャイル・プロジェクト管理では, 必ずしも明確で厳密な計画を持たない場合が多い. その代わりにプロジェクト・メンバ全員がプロジェクトの行く末や目標について一致した将来像を持つべきである.

プロジェクトの将来像には二つの種類がある.

- ビジョン - 自分たちの野心は何か? (メンバ個人 / プロジェクト自身)
- ミッション - 自分たちの貢献は何か? (対顧客 / 組織 / 社会 ...)

この二つがプロジェクトの出発点になる. ビジョンとミッションは定期的に見直すとともに, いつも身近に意識する.

ビジョンとミッションを実現するための方法は必ずしも明確でない. また複数の可能な方法の中でどれが最適か決めることができない場合も多い. そのためには戦略を持っていなければならない.

ビジョンとミッションを明らかにするために, プロジェクト・フォーミュレーションに参加したメンバが1日か2日の作業を行う. プロジェクト・フォーミュレーションに参加していなかったプロジェクト・メンバが参加するのもよい.

成果物はプロジェクト・メンバ全員が共有する.

- 必要ならば成果物の適当なバリエーションを上司や顧客などのステークホルダに配布する.
- 頻繁に (例えば毎週) ビジョンをレビューする.
    - そんなに大げさにする必要はないが, 定例ミーティングの時に参照する.

具体的なビジョン・ドキュメントの例

- 適応型ソフトウェア開発 (J. Highsmith)
    - プロジェクト・ビジョン (憲章)
        - ビジネス上の主目的, 製品 / システム仕様, 市場での位置づけなどを簡潔にまとめたもの
        - 2~10ページ程度
    - プロジェクト・データ・シート (PDS)
        - ビジネス上の有利点, 製品 / システム仕様, プロジェクト管理情報などを一目で分かるようにまとめたもの
        - 1ページ
    - 製品ミッション・プロファイル
    - 製品仕様概要 (PSO)
        - 製品のフィーチャ，機能，目的，データ，性能，操作などを高いレベルでまとめたもの

参考資料

プロジェクト・ビジョン

"戦略思考プロフェッショナル - 思考モデル + 行動モデル" by 八幡紕芦史, 2003, PHP研究所

2.2 スコープ計画プロセス 18
2.3 スコープ定義プロセス 19
ストーリ, フィーチャ, ... 20
3.1 活動定義プロセス 21
タスク 22
3.2 活動配列プロセス 23
APMにおける活動ネットワークの問題点 24
3.3 活動期間見積もりプロセス 25
プロジェクト速度 26
3.4 スケジュール策定プロセス 27
イテレーション期間 28
4.1 資源計画プロセス 29
APMにおける資源 30
4.2 コスト見積もりプロセス 31
APMにおけるコスト見積もり 32
4.3 コスト予算化プロセス 33
1.1 プロジェクト計画策定プロセス 34
獲得価値管理とAPM 35
プロジェクト・ホスティング・ツール 36
SPMP (IEEE1058) 37
SPEMによるプロセス記述 38
5.1 品質計画プロセス 39
アジャイル・プロジェクトにとっての品質 40
ISO/IEC9126 品質標準 41
6.1 組織計画プロセス 42
APMにおける組織, 人 43
6.2 スタッフ調達プロセス 44
APMにおける配員 45
7.1 コミュニケーション計画プロセス 46

*APM*におけるコミュニケーション *47*
8.1 リスク管理計画プロセス *48*
8.2 リスク特定プロセス *49*
*APM*におけるリスク特定 *50*
8.3 定性的リスク分析プロセス *51*
8.4 定量的リスク分析プロセス *52*
*APM*におけるリスク分析 *53*
8.5 リスク対応計画プロセス *54*
リスク対処の方法 *55*
効果図式 *56*
9.1 調達計画プロセス *57*
コンポーネント / ツール調達の問題点 *58*
持続可能な調達 *59*
9.2 要請計画プロセス *60*
アジャイル・プロジェクトにおける契約 *61*

2.2 スコープ計画プロセス

プロジェクトの意志決定の拠り所となるべき, スコープ記述を作成する.

入力

- プロダクト記述 -> 2.1 立ち上げの入力
- プロジェクト憲章 -> 2.1 立ち上げの出力
- 制約条件 -> 2.1 立ち上げの出力
- 前提条件 -> 1.1 プロジェクト計画策定の入力

ツールと技法

- プロダクト分析
- 利益 / コスト分析
- 代替案抽出
- 専門家の判断

出力

- スコープ記述
- 詳細な補助資料
- スコープ管理計画

2.3 スコープ定義プロセス

プロジェクトの主要な成果物を管理可能なコンポーネントに分解する

入力

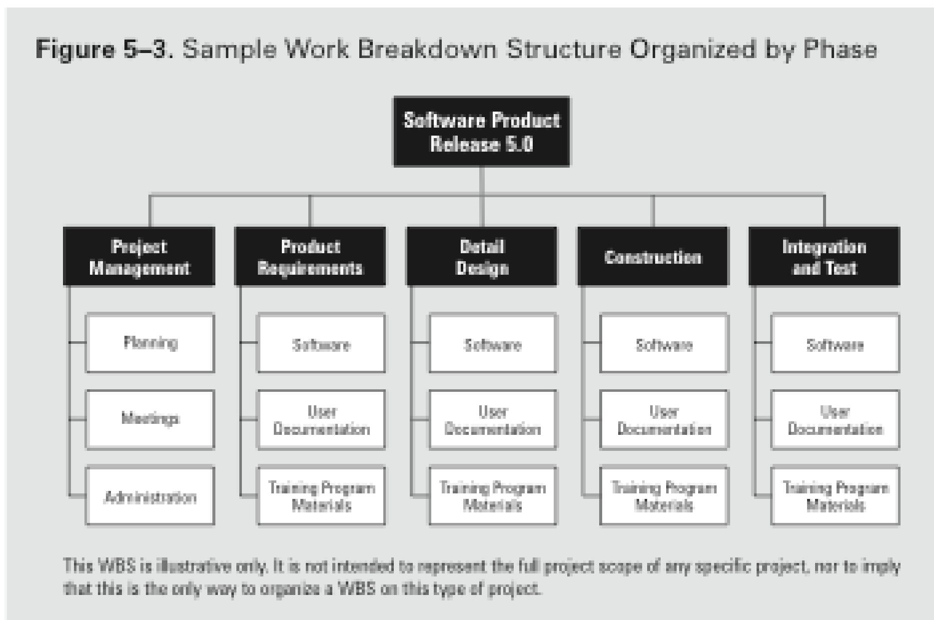
- スコープ記述 -> 2.2 スコープ計画の出力
- 制約条件 -> 2.1 立ち上げの出力
- 前提条件 -> 1.1 プロジェクト計画策定
- それ以外のスコープ計画プロセスの出力
- 実績情報

ツールと技法

- WBSテンプレート
- 分解

出力

- WBS



- スコープ記述の改訂

ストーリ, フィーチャ, ...

スコープは, ソフトウェア開発プロジェクトでは大まかに要件に対応すると考えていいだろう.

- 機能
- 品質
- 性能

アジャイル・プロジェクト管理では, スコープは常に変動しうる. そのためにスコープ記述をプロジェクト (あるいはフェーズ) 開始時に一度だけ作成して, 後はそれを遵守するというスタイルを取ることはできない.

- スコープの変更をどういうタイミングで許すかはいろいろ.

アジャイル・プロジェクトではスコープは以下のような形で定義される.

- XP - ストーリ
- Scrum - プロダクト・バックログ
- FDD - フィーチャ
- ......

どれも「ユーザに価値を与えるもの」というのがスコープ定義の基本.

WBSのように必ずしもツリー構造になる必要はないが, それぞれ短い記述項目の一覧として保存されていることが必要.

スコープの保存方法

- カード
- アウトライン・プロセッサ
- XPlannerなどのプロジェクト管理ツール
- Requsite Proのような要件管理ツール
- MSProjectのようなプロジェクト管理ツール

もし, 細分化が存在するならばその追跡可能性もやはり必要.

- 例えば
  - 入れ子にする
  - 付番する

優先順位が付けられていることも重要.

スコープは変わり得るので, それをちゃんと追跡することも重要.

- 例えば
  - 日付, バージョンを入れる.
  - 同じ紙の上に線を引いて変更を加える

スコープは必ずしも天啓のように誰かから授けられるものではない. 特にソフトウェアの場合には顧客にスコープを可視化する能力が足りない場合もある. 逆に

ストーリ, フィーチャ, ...

- 開発者には顧客の価値を理解することができない場合も多い. 顧客やユーザと一緒に作り上げていくもの.
  - そのときにはロール・プレイングやエンタープライズ・アーキテクチャ, UMLモデル, シナリオ・モデリングなど自分たちに適したさまざまな技法を活用すべし.
- アジャイル・プロジェクトであっても, スコープの内容はWBSとほぼ同様である.

3.1 活動定義プロセス

プロジェクトの成果物を作成するのに必要な作業活動を見極める

入力

- WBS -> 2.3 スコープ定義の出力
- スコープ記述 -> 2.2 スコープ計画
- 実績情報
- 制約条件
- 前提条件
- 専門家による判断 -> 2.1 立ち上げ, 3.3 活動期間見積もり

ツールと技法

- 分解
- テンプレート

出力

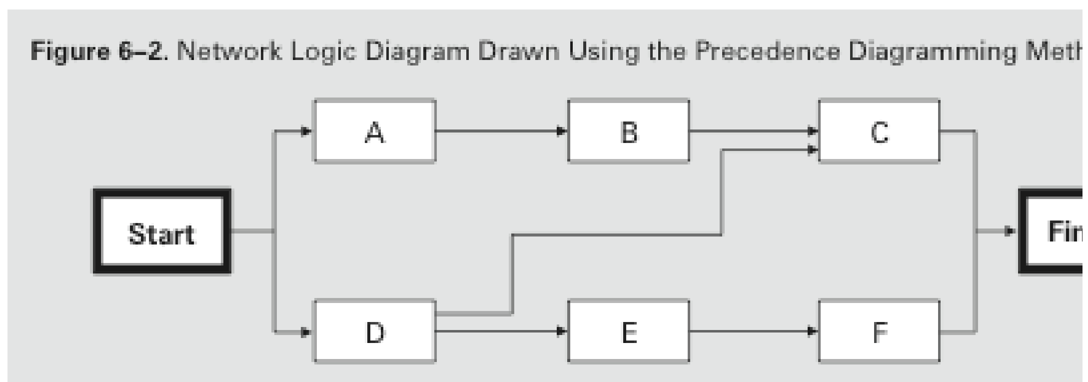
- 活動リスト
- 詳細情報
- WBSの改訂

タスク

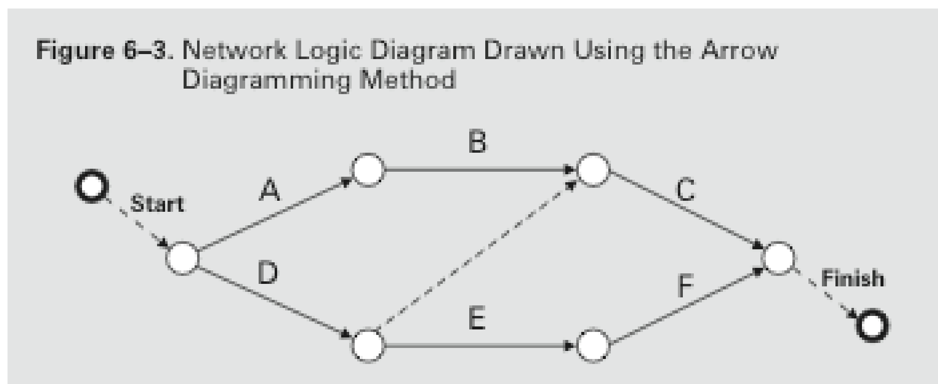
- 活動とはスコープを実現するために必要な作業のこと.
- アジャイル・プロセスでは以下のような形で定義される.
  - XP - タスク
  - Scrum - スプリント・バックログ (スコープも含む)
  - ……
- スコープ (ストーリ) などとの対応関係 (追跡可能性) が必要.
- 対応するスコープの優先順位にしたがって優先順位がつけられていることも重要.
- タスクは「誰が」「いつ」やるか, 決まっている必要がある.
  - 決めるタイミングはいろいろ.
  - 一人でやるとは限らない.
- タスクが抱えるリスクは何か
  - 必要ならばタスクのデータとして書き込んでおく
- タスクの完了条件は何か
- タスクの保存方法はスコープの保存方法と同様.
- ローリング・ウェイブ (時間の遠近法)
  - 遠い将来の計画はおおざっぱに決まっている
  - 近い将来の計画は詳細に決まっている
  - 例えば来月のタスクは大体分かっていればよい.
  - しかし今日やることはかなり細かいところまで分かっていなければ仕事できない.
- タスクのサイズ
  - プロジェクト・レベル - 1日 ~ 1週間一人程度
    - カードやツールに書く
    - 見積もりと測定をする
      - 最初どれだけの時間がかかると予測したか
      - 実際にどれだけの時間がかかったか
      - これはタスクのデータと一緒に記録しておく
    - できればタスクの途中には他の仕事を割り込ませない
  - 個人レベル - 最終的には一時間程度 (ToDo)
    - 手元の紙やポストイットに書く
    - ツールのジャーナル (ログ) に書く
- アジャイル・プロジェクトであってもタスクの内容はWBSの作業パッケージとほぼ同様.

3.2 活動配列プロセス

- 相互依存性を見極め, 文書化する
- 入力
  - 活動リスト -> 3.1 活動定義の出力
  - プロダクト記述 -> 2.1 立ち上げの入力
  - 必須の依存関係
  - 任意の依存関係
  - 外部の依存関係
  - マイルストーン
- ツールと技法
  - Precedence Diagramming Method (PDM, 活動ネットワークの書き方の一つ)
    - アクティビティをノードにする



  - Arrow Diagramming Method (ADM, 活動ネットワークの書き方の一つ)
    - アクティビティをアークにする



  - Conditional Diagramming Methods (活動ネットワークの書き方の一つ)
    - GERTやSystem Dynamicsなど. ループや条件を含む
  - ネットワークのテンプレート
- 出力

3.2 活動配列プロセス

プロジェクトのネットワーク図

活動リストの改訂

*APM*における活動ネットワークの問題点

活動ネットワークはWBS (タスクの山) に対して「時間」の概念を導入して, スケジュールを決めるために用いられる (PERT/CPM).

そのときに重要なのは, タスクの依存関係 (前後関係).

ただしソフトウェア開発の場合には, 一般的にタスク間の依存関係は明らかではないことが多い.

つまり, あるタスクがどのタスクに影響を及ぼすか, 実際にやってみないと分からない.

このためにちゃんとスケジュールを立てたはずなのに, 最後になって全然間に合わないという事態に陥る.

また各タスクのかかる時間を正確に見積もることが難しい.

これを回避するためにPERTの方法を用いるが, 面倒くさくなるだけで大して事情は変わらない (たいてい適当な3点を決めるだけになる).

活動ネットワークによってソフトウェア開発プロジェクトのスケジュールを決めるのは難しい.

活動ネットワークの表すリスク

クリティカル・パス

経路の最も長いパス

このパスがプロジェクト全体のスケジュールを決める

このパス上のタスクが遅れるとプロジェクト全体に影響を及ぼす

クリティカル・パスに近いパス

同上

したがってクリティカル・パスは短い方がよい.

これを突き詰めるとすべてのパスがクリティカル・パスになる

するとどのタスクが遅れてもプロジェクト全体に影響を及ぼす.

ファンイン / ファンアウト

ファンインの多いタスクは, 多くのうち一つでも遅れるとそのタスク自身が始められないので遅れやすい.

逆にファンアウトの多いタスクは, このタスクが遅れると他の多くのタスクに影響を与える.

これを突き詰めるとすべてのタスクは1-in / 1-outが望ましい.

これらを合わせると, すべてのタスクが依存関係なくパラレルなほどいいことになる.

-> コンカレント・エンジニアリング

プロダクトとプロセスのアーキテクチャが影響する

*APM*における活動ネットワークの問題点

- 必要なのは
  - 依存関係を少なくする / 明確にする
    - オブジェクト指向などモジュール性の高い基本アーキテクチャ
    - インタフェース・ベースの設計 / 実装
    - インタフェースの安定化
      - 単体テストとリファクタリングなど
    - 依存関係の自動的な解決
      - モデル駆動開発, コード自動生成, アスペクトなど

3.3 活動期間見積もりプロセス

個々の活動を実行するのに必要な作業期間を見積もる

入力

- 活動リスト -> 3.1 活動定義の出力
- 制約条件 -> 3.1 活動定義の入力
- 前提条件
- 資源要件 -> 4.1 資源計画プロセスの入力
- 資源の可用性
- 実績情報
- 特定されたリスク -> 8.2 リスク特定

ツールと技法

- 専門家による判断
- 類推による見積もり
- 数値に基づいた期間
- 不測の事態に対する予備

出力

- 活動期間の見積もり
- 見積もりの基
- 活動リストの改訂 -> 3.2 活動配列の出力

プロジェクト速度

アジャイル・プロジェクトでは, 今までに経験のないような作業を行うことになるので, 形式的な見積もりは難しい. 熟練者の経験と勘による見積もりも難しい. また予期しなかったことが起きて, 見積もりが狂うことはよくある. これは仕方がない. そこで全体としてはタイムフレームを決め, 個々には漸進的な見積もりを行う.

プロジェクト速度とは自分たちの見積もりと実際にかかった時間との比率を一定値に経験的に詰めていく手法.

タスクを大体同じ程度の期間に揃える.

タスクを見積もる.

タスクを実行して, 時間を測定する.

見積もりと実際値との比を算出する.

それを基に次のタスク見積もりを勘案する.

プロジェクト速度は一定値になるとは限らないが, 経験的な参考値にはなる

3.4 スケジュール策定プロセス

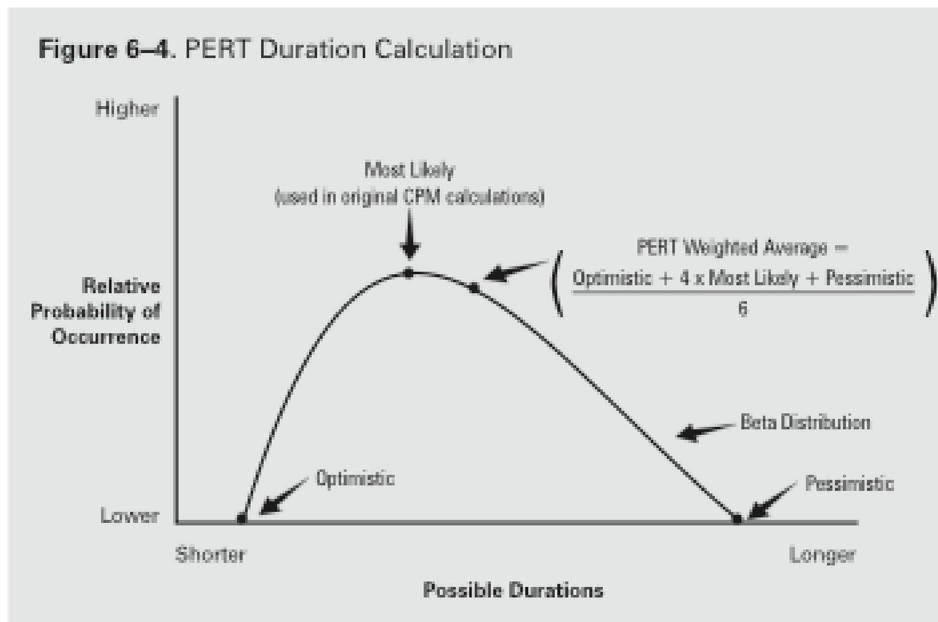
活動の配列, 期間, 資源要件を分析して, プロジェクトのスケジュールを作る

入力

- プロジェクト・ネットワーク図 -> 3.2 活動配列の出力
- 活動期間見積もり -> 3.3 活動期間見積もりの出力
- 資源要件 -> 3.3 活動期間見積もりの入力
- 資源プール記述
- 暦
- 制約条件
- 前提条件 -> 1.1 立ち上げの入力
- 先行時間 / 遅れ時間
- リスク管理計画 -> 8.1 リスク管理計画の出力
- 活動の属性

ツールと技法

- 数学的分析



- 期間圧縮
- シミュレーション
- 資源平準化のヒューリスクティクス
- プロジェクト管理ツール
- コーディング構造

出力

- プロジェクト・スケジュール

3.4 スケジュール策定プロセス

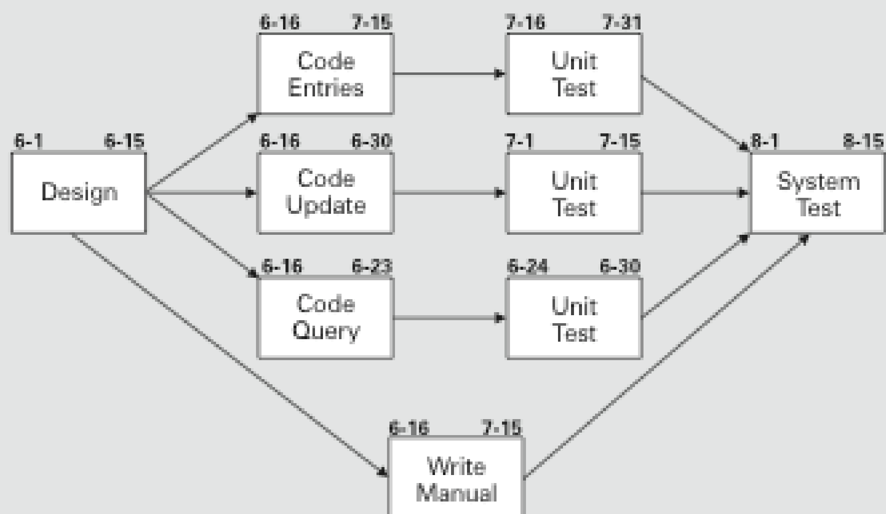


Figure 6–5. Project Network Diagram with Scheduled Dates

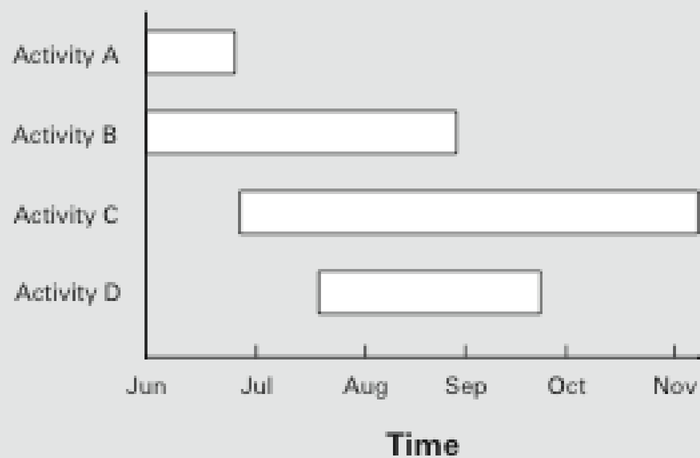


Figure 6–6. Bar (Gantt) Chart

3.4 スケジュール策定プロセス

Figure 6–7. Milestone Chart

Data Date

Event	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug
Subcontracts Signed			△ ▼					
Specifications Finalized				△ ▽				
Design Reviewed					△			
Subsystem Tested						△		
First Unit Delivered							△	
Production Plan Completed								△

There are many other acceptable ways to display project information on a milestone chart.

詳細

スケジュール管理計画

資源要件の改訂

イテレーション期間

- イテレーションの期間
  - 最適なイテレーション期間はプロジェクトの種類やプロセス, プロジェクト規模などによって変わってくる.
    - 一回のイテレーションで顧客やユーザにとって意味のある動作可能なソフトウェアを作り出すことができなければならない.
    - 抱えているリスクを考えるとあまり長くしてはならない.
    - 短すぎるとオーバヘッドが大きくなる可能性がある.
    - 開発者にとって適切なリズムを作り出す程度であってほしい.
  - 多くのアジャイル・プロセスでは2週間程度というのが一つの目安.
  - Scrumの場合には開発者にとってのリズムという側面を意識して, 30暦日固定.
  - いずれにしろ, いったん決めたイテレーション期間は変更しない.
    - すべてのイテレーション期間が同じである必要はない.
      - あんまりバラバラでなければ.
    - プロジェクトの最初にすべてのイテレーション期間を決めてしまう必要はない.
- スケジュールの表現
  - アジャイル・プロジェクトの場合, イテレーションの期間が決まれば特にこれを表示するダイアグラムがあるわけではない (イテレーションの期間がマイルストーンになる).
  - MSProject, XPlannerなど適当なツールを使えばよい.
  - あとは各イテレーションにスコープやタスクを割り当てる.
- イテレーションの内容
  - 通常はスコープ (ストーリ, プロダクト・バックログ, フィーチャなど) をイテレーションに割り当てる.
    - -> 機能ごとのイテレーション
  - それ以外にイテレーションにテーマを割り当てて, それを繰り返す場合もある.
    - 例えば
      - サイクル1 - スコープ / 概念
      - サイクル2 - 人間工学, 機能 , フィーチャ
      - サイクル3 - パフォーマンス, 整合性, 欠陥
      - サイクル4 - 製品 / システム全体のコンポーネント
  - この場合にはワークフローではなくワークステートが重要になってくる

イテレーション期間

コンポーネントの状態の例

- アウトライン (概念) 状態
- 詳細 (モデル) 状態
- レビュー済み (改訂済み) 状態
- 承認済み (利用可能) 状態

4.1 資源計画プロセス

プロジェクトの活動を実施するためにどのような資源 (人, 機材など) がどれだけ必要になるかを見極める

入力

- WBS -> 2.3 スコープ定義の出力
- 実績情報
- スコープ記述 -> 3.2 活動配列
- 資源プール記述
- 組織ポリシー
- 活動期間見積もり -> 3.3 活動期間見積もりの出力

ツールと技法

- 専門家による判断
- 代替案の考案
- プロジェクト管理ツール

出力

- 資源要件

*APM*における資源

- ソフトウェア開発プロジェクトの最大の資源 = コスト要因は通常, 人間である.
  - その他にはソフトウェア・ツール, ハードウェア, 場所, COTS (commercial-off-the-shelf)など
  - PMBOKでは対象となっていないが, プロセス, コンポーネント, フレームワーク, 内製ツール, ノウハウ, 知識なども重要な資源である.
  - これらの資源をどうするかという視点はPMBOKにはない. アジャイル・プロジェクト管理ではこれらの資源が最も重要.

4.2 コスト見積もりプロセス

プロジェクトの活動を行うのに必要な資源のコストを見積もる

入力

- WBS
- 資源要件
- 資源単価
- 活動期間の見積もり
- 見積もり資料
- 実績情報
- コスト科目表
- リスク

ツールと技法

- 類推による見積もり
- パラメトリック・モデリング
- 積み上げによる見積もり
- 見積もりツール
- その他のコスト見積もり法

出力

- コスト見積もり
- 詳細
- コスト管理計画

*APM*におけるコスト見積もり

- ソフトウェア開発プロジェクトでは多くの場合, コストの主要要因は人件費である. したがって, ほぼコストは人件費で決まる. その他のコストは期間ごとの一定額 (管理費毎月100万円) あるいはプロジェクト全体経費の一定割合 (管理費が全体の10%)とか.
- それ以外にはソフトウェア・ツール, 教育費などがある.
- 以上はソフトウェア開発に共通する.
- 人件費をどのように決めるか? アジャイルなやり方はあるだろうか
  - 現在は月当たりXXX万円というのが, 基本的な人件費の決め方.
  - XXXはその人の能力, 経験などを考慮に入れて何となく決まる (もちろん所属組織によって評価方法が厳密に決まっている場合もある).
  - それは正しいか
  - 作ったソフトウェアによって顧客が得た価値の一定割合を得るということは可能だろうか
    - どれだけの割合? 貢献度

4.3 コスト予算化プロセス

見積もったコストを個々の作業活動に割り当てる

入力

　コスト見積もり -> 4.2 コスト見積もりの出力

　WBS -> 2.3 スコープ定義の出力

　プロジェクト・スケジュール -> 3.4 スケジュール策定の出力

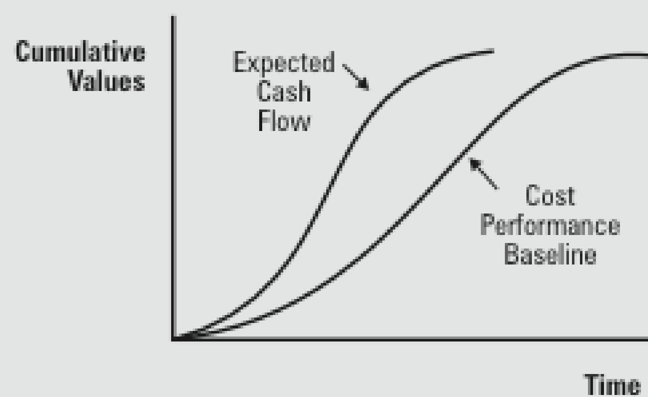
　リスク管理計画 -> 8.1 リスク管理計画の出力

ツールと技法

　コスト予算化のツールと技法 -> 7.2 コスト見積もりのツールと技法と同じ

出力

　コストのベースライン



1.1 プロジェクト計画策定プロセス

すべてのプロジェクト計画を調整・統合して, 一貫性のあるドキュメントを作る

入力

- 他の計画
- 実績情報
- 組織のポリシ
- 制約条件
- 前提条件

ツールと技法

- プロジェクト計画方法論
- 利害関係者のスキルと知識
- プロジェクト管理情報システム(PMIS)
- 獲得価値管理 (EVM)

出力

- プロジェクト計画
- 詳細

計画プロセス・グループ

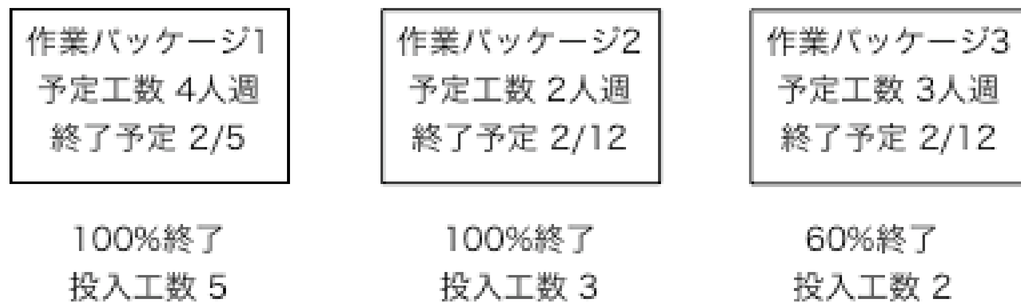
35-1

獲得価値管理と*APM*

Earned Value Managementのあらまし

例えば次のようなWBSの作業パッケージがあったとする.

今日は2/12.



このときこの三つの作業の価値は

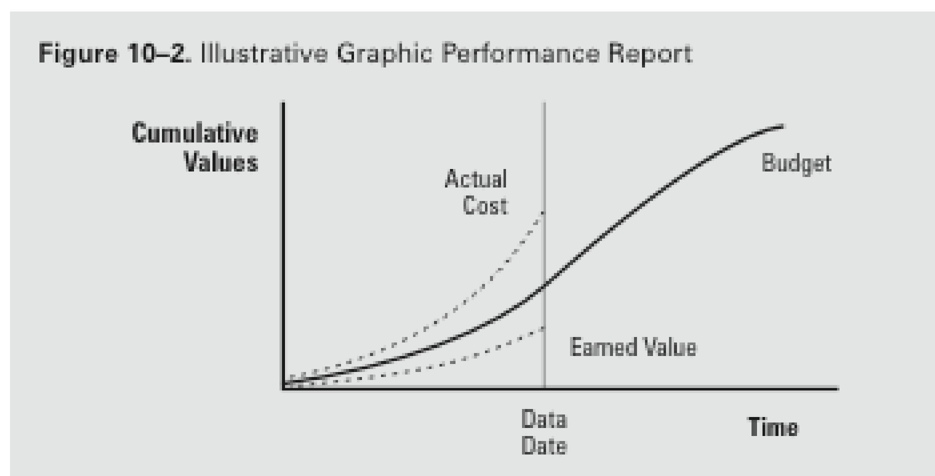
値札では4+2+3 = 9 <- BCWS (Budgeted Cost of Work Scheduled)

実際に支払ったのは5+3+2 = 10 <- ACWP (Actual Cost of Work Performed)

手元にあるのは4+2 = 6 <- BCWP (Budgeted Cost of Work Performed)

CPI = ACWP / BCWP = 1.7 => 工数は予定の1.7倍必要

SPI = BCWS / BCWP = 1.5 => 期間は予定の1.5倍必要



EVMは, 直線的な進捗を前提としているので, 非線形的な進捗を特徴とするソフトウェア開発プロジェクトには使いにくい.

アジャイルでは以下のような方法をとった方がいいのではないか

プロジェクト速度に基づく漸進的な精度向上手法

獲得価値管理と*APM*

イテレーションを小さくすることによる精度と見積もりの繰り返し

プロジェクト・ホスティング・ツール

-> 1.3 全体的な変更制御

SPMP (IEEE1058)

- IEEE 1058 Software Project Management Plan (SPMP) はソフトウェア開発計画のよいテンプレートになっている.
  - @ Annotated Outline for the P1058 Standard for Software Project Management Plans
- ただし
  - 最初からすべての計画を立てるのは無理 or 意味がない.
  - 毎週レビューして, 必要なところを少しずつ埋めていく and 改訂していく.
    - 毎週の計画レビュー・ミーティング (30分程度, 全員)
    - これがリスク・ミーティングにもなっている
  - できない / まだやらないところは空白にしておき, 必要ならばいつ頃書くかを書いておく
  - そのためにテンプレートをPlone (コミュニティ・サイト) の上に載せている.
    - @ Dev Cats - Sign in
  - 目標はプロジェクト終了時に計画ができあがっていること (!)
  - いったん作られたSPMPはこれ以降のプロジェクトでもかなり再利用できる.
- 「計画は守るべきルールではなく, 事実を知るための定規」

SPEMによるプロセス記述

SPEM

Software Process Engineering Metamodel

UML(Unified Modeling Language) をソフトウェア・プロセス記述に応用したもの

UML と同じOMG によって制定

http://www.omg.org/technology/documents/formal/spem.htm

SPEM (http://www.metabolics.co.jp/SoftwareProcess/SPEM.html)

SPEMとアジャイル開発プロセス

SPEM自体はアジャイル開発プロセスと直接の関係はない

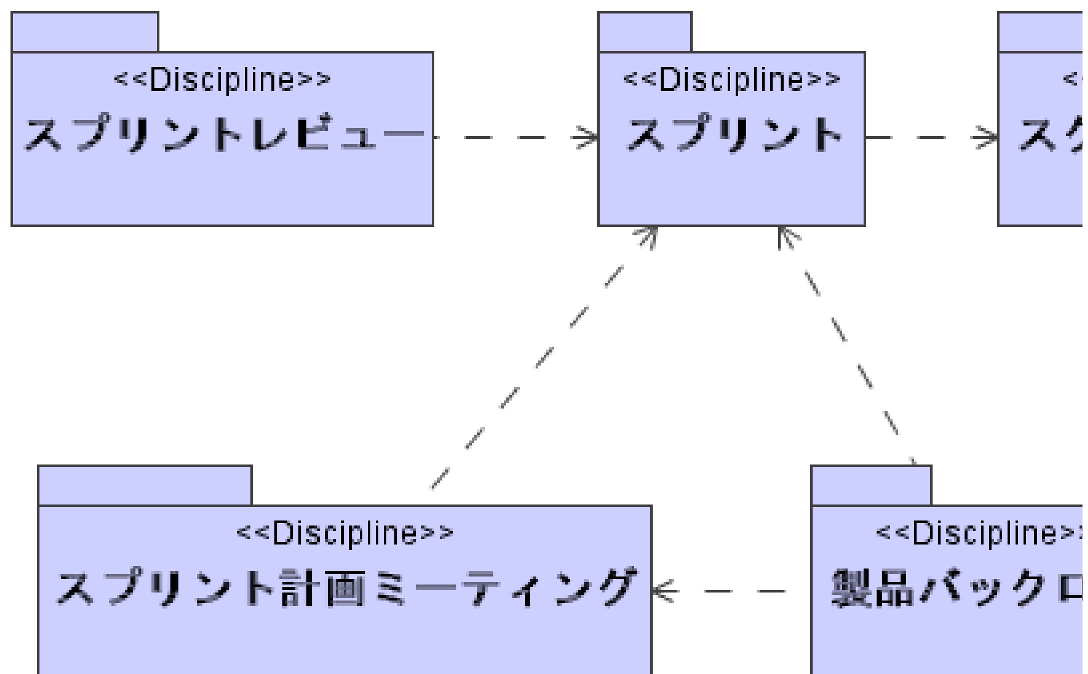
従来型ソフトウェア開発プロセスやUnified Processなどの記述にも使われる

これからプロセス記述やプロセス・ツールなどで広く使われるようになるだろう

SPEMによる記述からプロセス・ツールのインスタンスを自動生成するなど

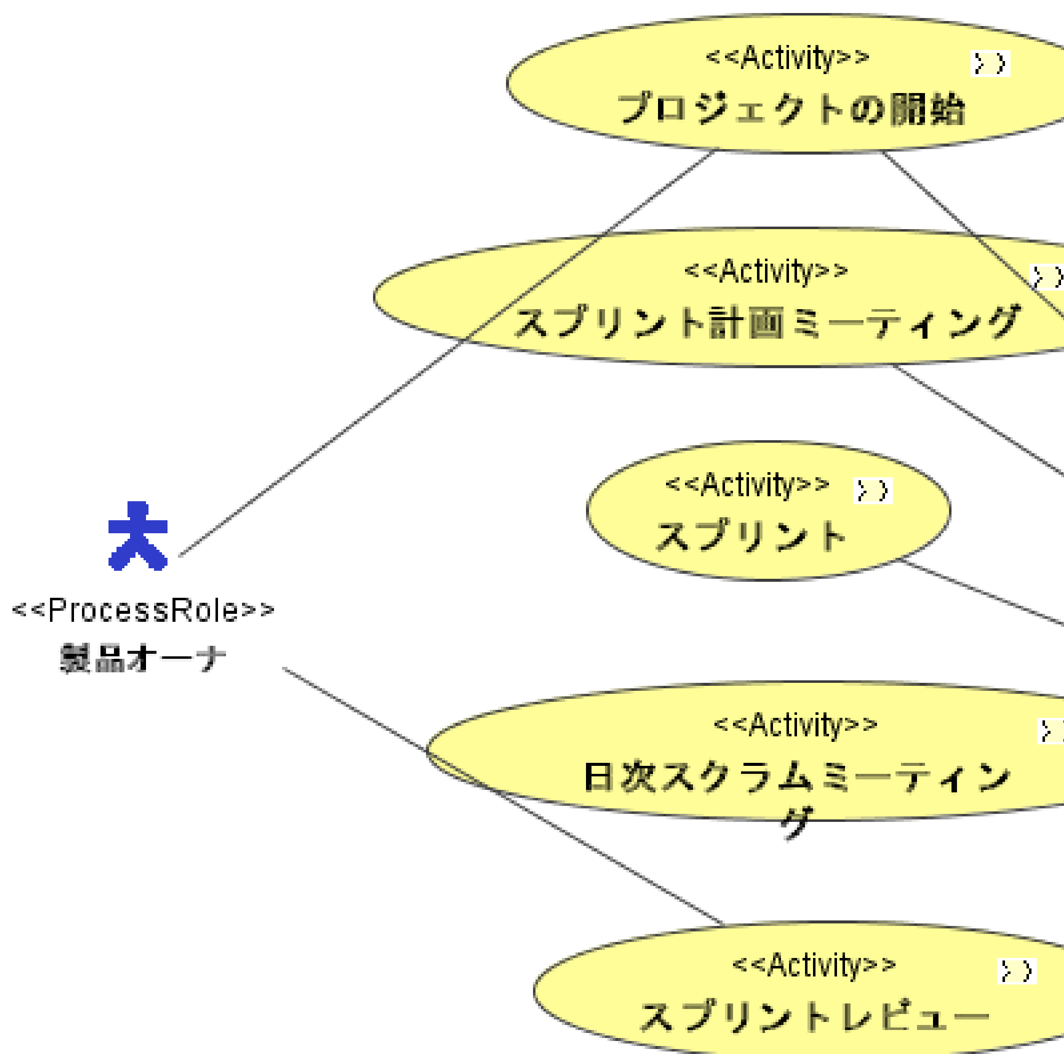
プロセス・チューニングやカスタマイズなどで役に立つ

SPEMによるScrumの記述例



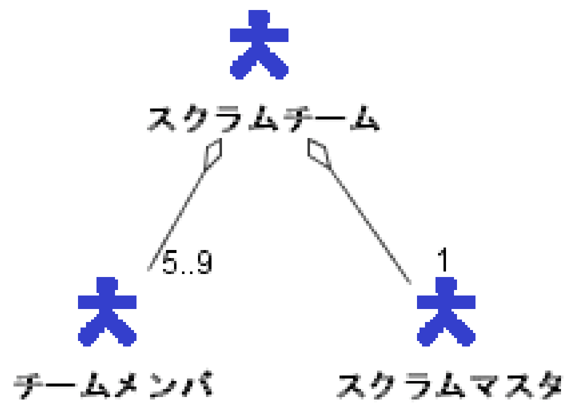
Scrum-Practices.png

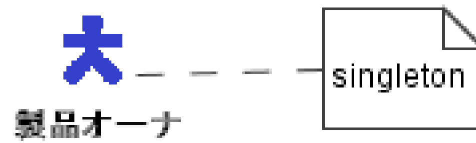
*SPEM*によるプロセス記述



Scrum-Process.png

Scrum-Products.png

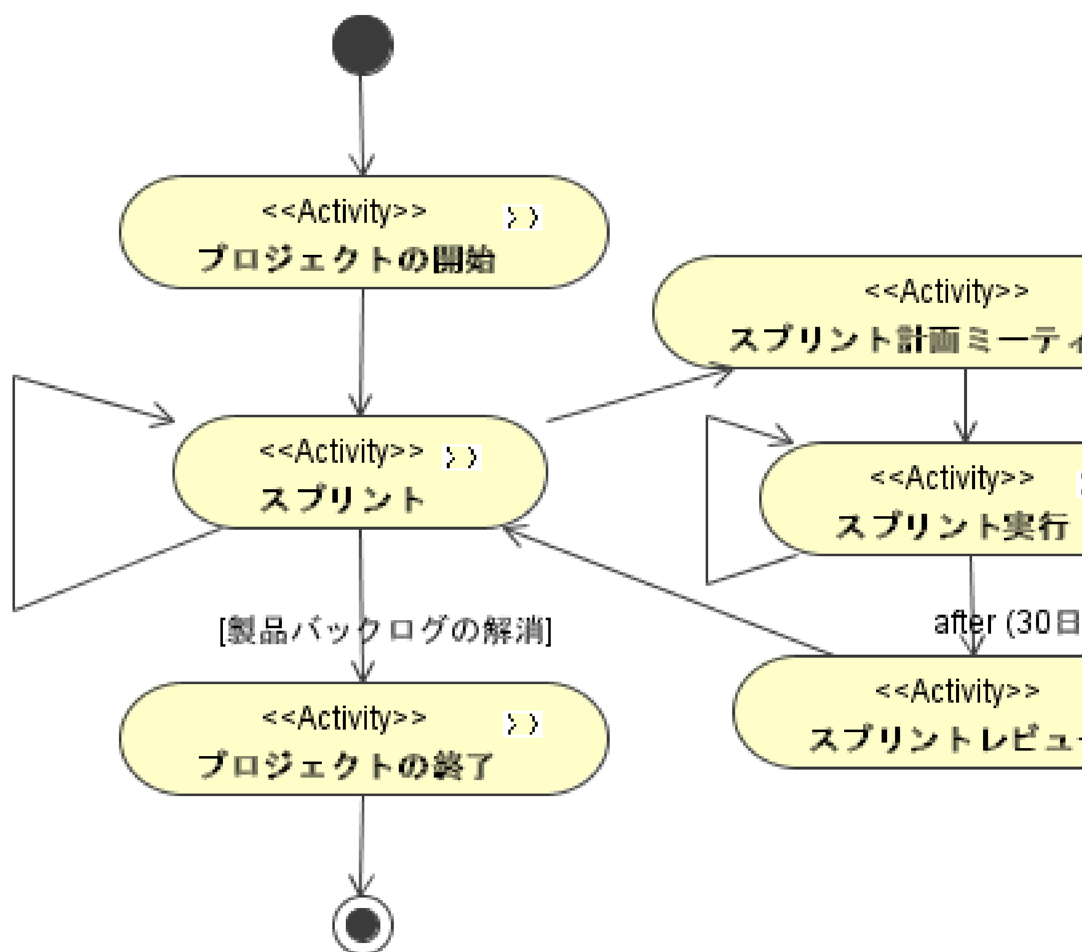




Scrum-Roles.png

計画プロセス・グループ

*SPEM*によるプロセス記述



Scrum-Workflow.png

5.1 品質計画プロセス

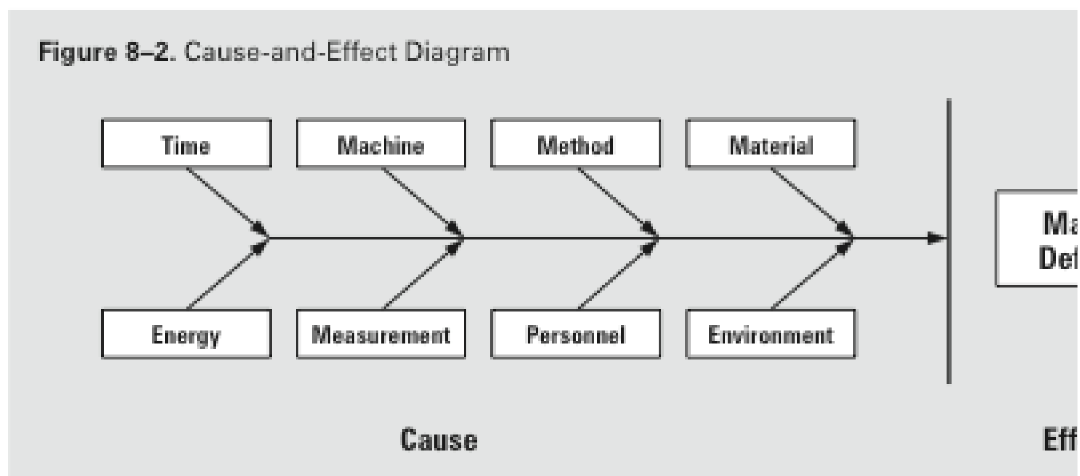
プロジェクトに関連する品質基準を見定め, それを満たすにはどうすればよいか考える.

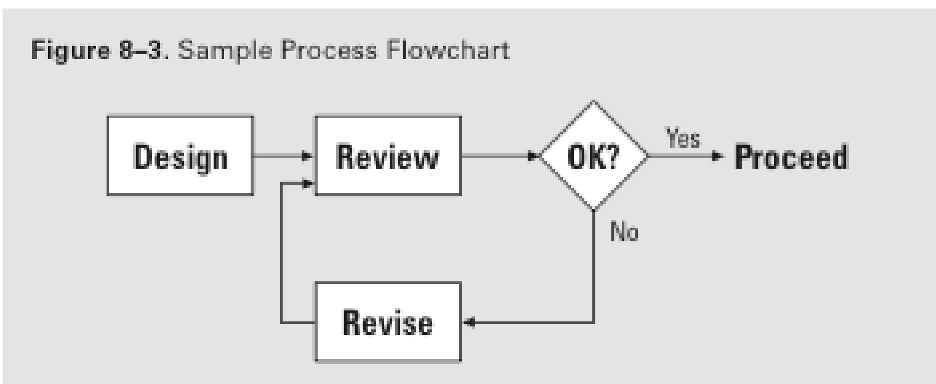
入力

- 品質ポリシ
- スコープ記述 -> 2.2 スコープ管理計画の出力
- プロダクト記述 -> 2.1 立ち上げの入力
- 標準と法規
- その他のプロセスの出力

ツールと技法

- 利益 / コスト分析 -> 2.2 スコープ管理計画
- ベンチマーキング
- フローチャート





- 実験の設計
- 品質のコスト

出力

- 品質管理計画

5.1 品質計画プロセス

操作定義

チェックリスト

その他のプロセスへの入力

アジャイル・プロジェクトにとっての品質

- 品質が高いとは必ずしも欠陥の少ないことではない.
  - 「品質とは誰かにとっての価値だ」
  - 顧客満足度
    - 誰もが満足するわけではない
- ISO品質標準 -> 次ページ
- QWAN Quality with Anonymous Name
  - 名づけえぬ質
  - これは単目的的な仕事では作り出すことができないだろう.
- 直接顧客やユーザのいないプロジェクトでの品質とは
  - 自分たちで「価値」を作り上げていかなければならない.
  - もちろんインタビューやマーケット調査も含まれるだろうが, 最終的には判断するのは自分.
- 参考までにアレグザンダーの生産プロセス
  - 建築家と施工は分離されていてはならない (アーキテクトビルダー)
  - 生産システムは, 地域的で高度に分散化された職人群を利用すべきである (ビルダーズヤード)
  - 共有地はユーザーにとってもっとも大切な場所であり, 彼らの管理下にあるべきである (共有地の共同設計)
  - ユーザーは, 自分自身の住まいの間取りについて, 現代建築のような受け身の姿勢ではなく, 核家族によってそれぞれ異なる間取りができるような積極的な姿勢で参加しなければならない (ここの住宅のレイアウト)
  - 施行のシステムや技術やディテールは, プロセスそのものが必要とする, 連続的な巧妙かつ微妙な修正の利くものを選ばなくてはならない (一歩一歩の建設)
  - コストコントロールは, 柔軟な設計施工のプロセスに適応しなければならない (コストコントロール)
  - 建物の細工は, 手作りの楽しいものでなくてはならない (プロセスの人間的リズム)
- ちなみにGoogleのプロセス

  - CNET Japan Blog - 梅田望夫・英語で読むITトレンド：天才社員が支えるGoogleのマネジメント手法
  - 平均3人のグループ (設計からビジネス化まで)
  - 何でも共有する
  - 軽量級のグループ間コミュニケーション (ツールを用いる)

アジャイル・プロジェクトにとっての品質

- どこからでもアイデアを集めて, 議論し, 優先順位を付けて実現する
- 採用を重視する (時間とコストを掛ける)
  - 自信に溢れた挑戦的な若いやつ (ハッカー)
  - すでに一流の実績を持っている研究者

参考資料

"ソフトウェア文化を創る 1 ワインバーグのシステム思考法" by G. M. ワインバーグ, 1994, 共立出版
"クリストファー・アレグザンダー - 建築の新しいパラダイムを求めて" by スティーブン・グラボー, 1989, 工作舎
"パターンランゲージによる住宅の建設" by C. アレグザンダー他, 1991, SD ライブラリー, 鹿島出版会

ISO/IEC9126 品質標準

- 機能性 - 全体としてのユーザ・ニーズの充足度
  - 合目的性
  - 正確性
  - 接続性
  - 整合性
  - 機密性
- 信頼性 - 所定の条件下で正しく稼働している割合
  - 成熟性
  - 障害許容性
  - 回復性
- 使用性 - 理解, 習得, 運用の容易さの度合い
  - 理解性
  - 習得性
  - 操作性
- 効率性 - 所定の機能を実現する処理性能と必要資源の効率
  - 実行効率性
  - 資源効率性
- 保守性 - 修正と改善の容易さの度合い
  - 解析性
  - 変更作業性
  - 安定性
  - 試験性
- 移植性 - 他の環境へ移す場合の容易さの度合い
  - 環境適応性
  - 移植作業性
  - 規格準拠性
  - 置換性
- "実力派SE養成コース 決定版 プロジェクト管理 - 成功するソフトウェア開発の最新スタイル - RUP, CMM/CMMI, Agile... 「うまくいかない」過去からの脱却" by 橋本隆成, 2004, 技術評論社

計画プロセス・グループ

6.1 組織計画プロセス

プロジェクトの役割, 責務, 報告の関係を特定し, 文書化し, 割り当てる

入力

- プロジェクトのインタフェース (組織, 技術, 人間)
- 配員要件 -> 4.1 資源計画
- 制約条件

ツールと技法

- テンプレート
- 人材のプラクティス
- 組織化理論
- 利害関係者の分析

出力

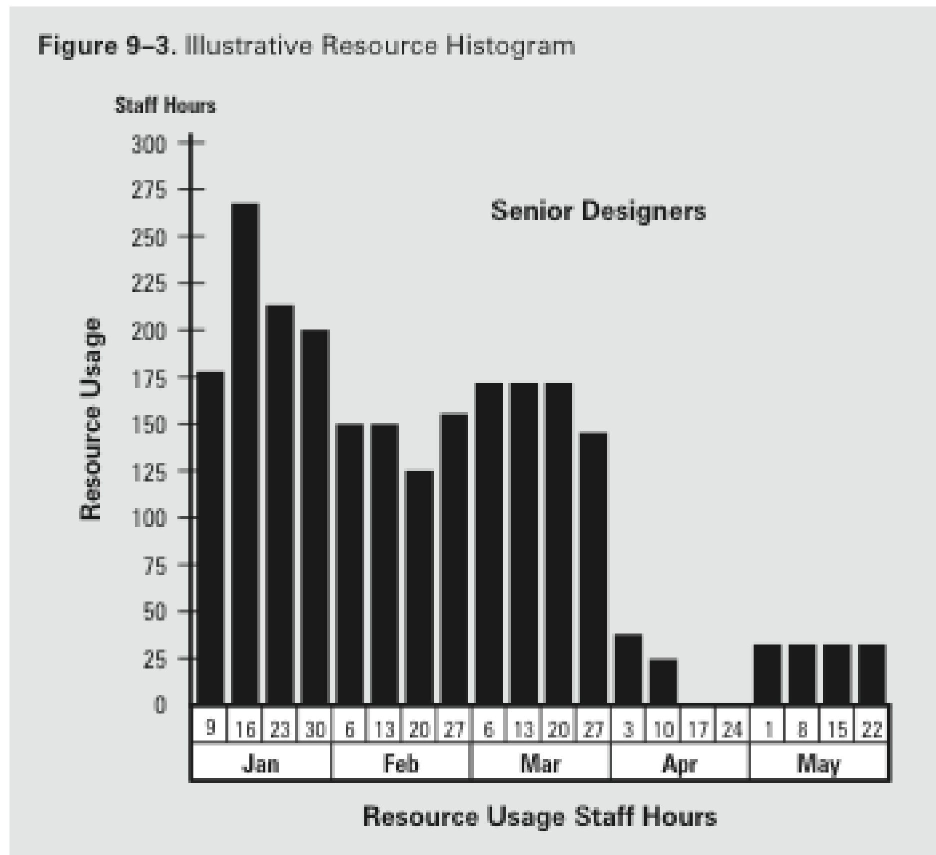
- 役割と責務の割り当て

Figure 9–2. Responsibility Assignment Matrix

PHASE \ PERSON	A	B	C	D	E	F	...
Requirements	S	R	A	P	P		
Functional	S		A	P		P	
Design	S		R	A	I		P
Development		R	S	A		P	P
Testing			S	P	I	A	P

P = Participant　A = Accountable　R = Review required
I = Input required　S = Sign-off required

- 配員管理計画



組織チャート

詳細

*APM*における組織, 人

- 役割
  - できるだけ最初から役割を固定しない.
    - 創発的に役割が生まれ, 担当が決まっていく
    - 何が起こるか分からないから, 何かが起きたときには全員で自分の得意な面を活かしてチームに貢献する. もちろん時には不得意なことも.
    - 責任を限定することによって, 無責任になっていく場合がある
    - もちろん役割が決まっていないことによって無責任になっていく場合もある
  - 全員がいろいろな役割を担えるようになるとよい
    - 個人の成長 / 組織の成長
  - マネージャ / ファシリテータ / コーチなどに相当する役割はあった方がいいだろう
- チームの規模
  - できるだけ小さく.
  - 例えば5~9人. もっと小さくてもよい.
    - 少ない人数でも実行が可能なように環境を整えていく.
  - それだけでは足りない場合には, 複数の小さいチームからなるグループを作る.
    - グループ間コミュニケーションが重要.
      - ツールなどを活用, グループ / プロジェクト間の垣根を作らない.
    - 例えばときどきグループ間で「トレード」があってもいい.
- 人
  - 人は入れ替え可能な資源 (人材) では実はない.
  - しかし, チームは一つ.
    - トラック・ナンバの考え方
      - チームの何人がトラックに轢かれてもチームは存続できるか
  - プロジェクトの目的の半分は実は人 (ひいては組織) を育てること.
    - 教育ではなく学習
    - ペア・プログラミング, レトロスペクティブ, ...
  - 人と組織の間に長期的な関係を築く (時代遅れな考え方?)
- コロケーション

6.2 スタッフ調達プロセス

必要な人材を調達してプロジェクトに任命する

- 入力
  - 配員管理計画 -> 6.1 組織計画プロセスの出力
  - 人材プール記述
  - 雇用習慣
- ツールと技法
  - 交渉
  - 固定配員 (プロジェクトの最初から決められているメンバ)
  - 調達
- 出力
  - プロジェクト・メンバの任命
  - プロジェクト・チーム一覧

*APM*における配員

- できるだけマルチアサインをしない
  - モチベーションも集中力も低くなる
  - もしマルチアサインするならば, イテレーションの区切りなど
- できるだけ途中での入れ替えをしない
  - 同上
- 急激なメンバ追加をしない
  - ソフトウェア開発プロジェクトの常識

7.1 コミュニケーション計画プロセス

利害関係者に必要な情報とコミュニケーションを決める

入力

- コミュニケーション要件
- コミュニケーション技術
- 制約条件
- 前提条件 -> 1.1 プロジェクト計画策定の入力

ツールと技法

- 利害関係者の分析

出力

- コミュニケーション管理計画

*APM*におけるコミュニケーション

- コロケーション
- 構成管理ツール
- ファシリテータ, コーチ
- プロジェクト・ホスティング・サイト
- コミュニティ・サイト
- email, chat
- 情報壁
- 飲み会, 食事会, 遊び

8.1 リスク管理計画プロセス

プロジェクトのリスク管理活動の方法と計画を決める

入力

- プロジェクト憲章 -> 2.1 立ち上げの出力
- 組織のリスク管理ポリシ
- 決められた役割と責務
- 利害関係者のリスクへの寛大さ
- 組織のリスク管理計画のテンプレート
- WBS -> 2.3 スコープ定義の出力

ツールと技法

- 計画ミーティング

出力

- リスク管理計画

8.2 リスク特定プロセス

プロジェクトに影響を及ぼすリスクを洗い出し, 文書化する

- 入力
  - リスク管理計画 -> 8.1 リスク管理計画の出力
  - -> 1.1 プロジェクト計画の出力
  - リスク分野
  - 実績情報
- ツールと技法
  - 文書レビュー
  - 情報収集技法
    - ブレーンストーミング
    - デルファイ法
    - インタビュー
    - SWOT分析
      - 強み / 弱み, 機会 / 脅威のマトリックスで, シナリオ / オプションを分析する
  - チェックリスト
  - 前提条件の分析
  - ダイアグラム作成技法
    - 原因-結果図 -> 5.1 品質計画プロセス
    - システム / プロセス フローチャート -> 5.1 品質計画プロセス
    - 影響ダイアグラム
- 出力
  - リスク
  - トリガ
  - その他のプロセスへの入力

*APM*におけるリスク特定

- アジャイル・プロジェクト管理全体がリスクを最小化する (と同時に機会を最大化する…矛盾している?) 仕組みになっている.
- プロジェクト・リスクはさまざまなプラクティスによって発見できる
- 重要なのは
  - 多くの目で見る
  - 可視化する

8.3 定性的リスク分析プロセス

リスクと条件を定性的に分析して, プロジェクトの目的に対する影響を優先度付ける

入力

- リスク管理計画 -> 8.1 リスク管理計画
- 特定されたリスク
- プロジェクトの状態
- プロジェクトの種類
- データの精度
- 確率と影響の範囲
- 前提条件

ツールと技法

- リスクの確率と影響
- 確率 / 影響のリスク格付けマトリクス
- プロジェクト前提条件のテスト
- データ精度の格付け

出力

- プロジェクトのリスク全体の格付け
- 優先度付けられたリスクの一覧
- その他の分析や管理のためのリスクの一覧
- 定性的リスク分析結果のトレンド

8.4 定量的リスク分析プロセス

リスクの確率と結果を計算して, プロジェクトに及ぼす影響を見積もる

入力

- リスク管理計画 -> 8.1 リスク管理計画の出力
- 特定されたリスク -> 8.2 リスク特定の出力
- 優先度付けられたリスクの一覧 -> 8.3 定性的リスク分析プロセスの出力
- その他の分析や管理のためのリスク一覧 -> 8.3 定性的リスク分析プロセスの出力
- 実績情報 -> 8.2 リスク特定の入力
- 専門家による判断 -> 2.1 立ち上げ
- その他の計画の出力

ツールと技法

- インタビュー
- 感受性分析
- 決定ツリー分析
- シミュレーション

出力

- 優先度付けられた, 数値化されたリスクの一覧
- プロジェクトの確率分析
- 目的を達成するのに必要なコストと時間の確率
- 定量的リスク分析の結果のトレンド

APMにおけるリスク分析

定量的リスク分析の罠

ソフトウェア開発プロジェクトを対象とした定量的なリスク分析を行うツールも存在する.

例えばCOCOMO / CrystalBall

Crystal Ball 2000

CrystalBallはExcelのセルに確率分布を割り当て, モンテカルロ・シミュレーションによってリスクを計算するリスク分析ツール

計算式としてCOCOMOを用いることによって, ソフトウェア開発の期間, 工数, 規模に関するリスクを定量化できる

しかし正直に言って大して役に立たない. (他の定量化ツールも同じ)

まずソフトウェア開発定量化の尺度が決まっていない (決められない).

ソフトウェア開発を行うのは人間だから (他の工学と異なり自然科学的因子はほとんどない), 定量分析の結果に影響を受ける.

ソフトウェア開発プロジェクトは複雑系であると考えられているが, このような系を定量的にうまく扱う方法はまだ応用されていない.

ただし「何でもいいから数字の裏付けが欲しい」(それによって顧客を納得させる)とか600人が5年間動くプロジェクト (すでにプロジェクト自身が統計的に扱える可能性がある) ならば何らかの役に立つかもしれない.

リスクの性質

確率と影響

確率をおおざっぱに例えば4段階で見積もる

影響をおおざっぱに金額に換算して4段階で見積もる

プロジェクト規模によって例えば10万円, 50万円, 250万円, 1,000万円

そのマトリックスを書くことによっておおざっぱな危険度 (確率 x 影響) を見積もれる

	¥100K	¥500K	¥2.5M	¥10M
0%~25%	¥13K	¥63K	¥310K	¥1.3M
25%~50%	¥38K	¥190K	¥930K	¥3.8M
50%~75%	¥63K	¥310K	¥1.6M	¥6.3M
75%~100%	¥88K	¥440K	¥2.2M	¥8.8M

大中小でも構わない!

これによってリスクに優先度を付けることができる

8.5 リスク対応計画プロセス

プロジェクトの目標に対する脅威を減らし, 機会を増大させるための手続きと技法を作成する

入力

- リスク管理計画 -> 8.1 リスク管理計画の出力
- 優先度付けられたリスクの一覧 -> 8.3 定性的リスク分析の出力
- プロジェクトのリスク格付け -> 8.3 定性的リスク分析の出力
- 数値化されたリスクの優先度付き一覧 -> 8.4 定量的リスク分析の出力
- プロジェクトの確率分析 -> 8.4 定量的リスク分析の出力
- 目標達成のためのコストと時間の確率 -> 8.4 定量的リスク分析の出力
- あり得る対応の一覧
- リスク閾値 -> 8.1 リスク管理計画
- リスク担当者
- 一般的なリスクの原因
- 定性的 / 定量的リスク分析結果のトレンド -> 8.3, 4 定性的 / 定量的リスク分析

ツールと技法

- 回避
- 移転
- 緩和
- 受容

出力

- リスク対応計画
- 残っているリスク
- 二次的なリスク
- 契約
- 必要な不測の事態のための予備の量
- その他のプロセスへの入力
- 改訂したプロジェクト計画への入力

リスク対処の方法

アジャイル・プロジェクト管理ではリスクは必ずしも避けなければならないものではない. リスクは裏返せばチャンスにつながるから.

- リスクをチャンスにつなげるためには, リスクを乗り切れなかったときにその責任を末端に押しつけるような文化ではやっていけない.

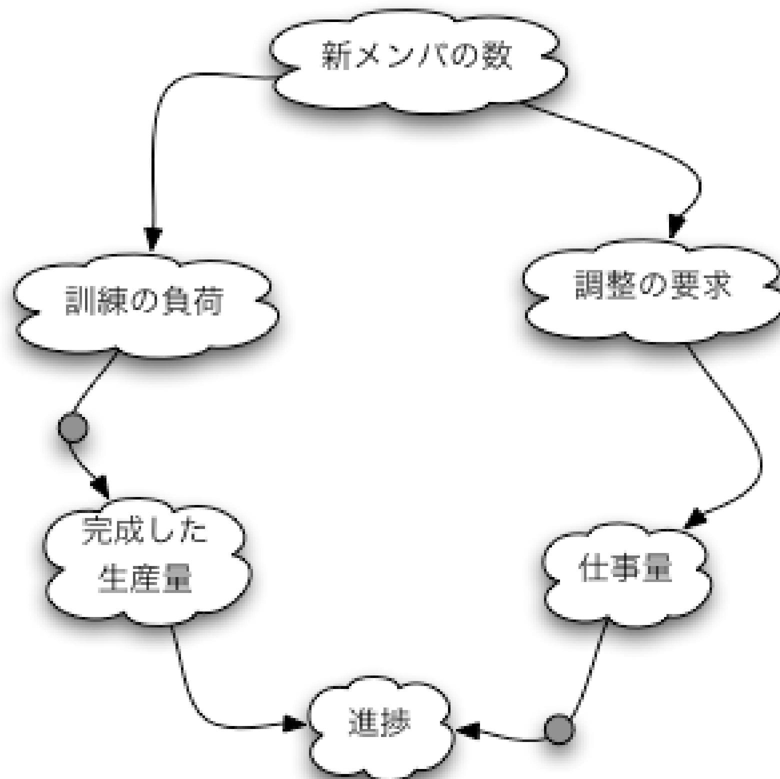
リスク対処の原則
- 過剰反応しない
- 引き出しを多く持つ (多様性, 多面性, いろいろな解法)
- すべてのリスクの同時に対応することはできない.
  - 最も重要なリスクに資源を集中して, スループットを上げる.

リスクの危険度と対処コスト
- リスクの危険度より対処コストの方が高ければ, リスクに対処する意味がない.

リスク対処の方法
- 回避する
  - リスクのないことだけをやっていてはチャンスはない
- 転移する
- 低減する
  - 確率を下げる
  - 影響を下げる
- 時期をずらす
- 受け入れる

リスク対処方法を考える
- 効果図式 -> 次のページ

効果図式 (ワインバーグ)
雲形のノードは計測可能な量を表す
楕円のノードを使う場合もある
ノードAからノードBへの矢印はAがBに対して影響を持つことを表す
矢印上にマークがなければ, 影響は同じ方向に働く (positive feedback)
灰色の丸 -> 反対方向に働く (negative feedback)
四角は人間が行う行為を表す
白 -> 同じ方向
灰 -> 反対方向
混合 -> まだ選択されていない
これはフィードバックをもつシステムを表す



ブルックスの法則

新メンバの数が増えると, 訓練の負荷が増え, 生産量が減り, 進捗も減る. また, 新メンバの数が増えると調整の要求が増え, 仕事量が増え, 進捗が減る. したがって新しいメンバが増えれば, 進捗は減るばかり.

対処の結果を (フィードバックを持つ) システムとして考え, シミュレートする

効果図式

一つの方法.

9.1 調達計画プロセス

何をいつ調達するかを決める

入力

- スコープ記述 -> 2.2 スコープ計画の出力
- プロダクト記述 -> 2.1 立ち上げの入力
- 調達のための資源
- 市場状況
- その他の計画の出力
- 制約条件
- 前提条件

ツールと技法

- 自製 / 購入分析
- 専門家による判断
- 契約種類選択

出力

- 調達管理計画
- 作業指示書

コンポーネント /ツール調達の問題点

ソフトウェア開発における主な調達は以下のようなものになる.

- 下請負業者 (いわゆる外注)
  - 受託開発
  - 派遣
- ベンダ
  - ソフトウェア・コンポーネント
  - ソフトウェア・ツール
- 下請負業者 -> 9.5 契約事務
  - そもそもなんのために外注するのか
    - 技術がない
      - 一緒に作業することによって技術移転が得られる場合がある
    - 人がいない
      - アジャイル・プロジェクトでは, 実は思ったより少ない人数で成果を上げられる場合がある
    - コストを下げたい
      - 実は思ったほどコストが下がらない場合がある
        - 管理コストなど (外注コストの2~5倍かかる)
      - そもそも管理ノウハウがなかったりする
      - 空洞化が進む
- ソフトウェア・コンポーネント
  - コンポーネントの購入にかかるコスト / リスクはコンポーネントの費用だけではない.
    - コンポーネントのインタフェースに我々のシステムを合わせる必要があるかもしれない.
    - コンポーネントの機能や品質が我々の要求にあっているかどうかを調査しなければならない.
    - ソース・コードが提供されていない場合には, 将来のプラットホームのバージョンアップ時にコンポーネント (あるいはベンダそのもの) が消えているかもしれない.
    - ソース・コードが提供されていない場合には, コンポーネントに問題があったらベンダと連絡を取ってベンダに修正を依頼しなければならない.
  - 通常コンポーネント使用にかかるコストは, コンポーネント購入費用の2倍から5倍かかることを考慮しておいた方がよい.
- ソフトウェア・ツール

コンポーネント /ツール調達の問題点

本質的にソフトウェア・コンポーネントと同じような問題を抱えている.
どちらの場合もオープン・ソース・プロダクトの場合にはリスクが緩和される.

その代わり誰かが責任を持ってくれると言うことはない.

オープン・ソース・プロダクトでもプロフェッショナル・サービスが有償で提供されているものもある.

持続可能な調達

アジャイルな契約では下請負業者との間にまだ高い信頼関係がない場合には, 次のようにする.

- 契約期間を短くする (1~2回のイテレーション程度, 例えば1~3ヶ月).
- その代わりに契約時に要件を確定しない.
  - 顧客側から言えば確定できない, 請負側から言えば正確に見積もりきれない.
- 最初の契約で信頼を得ることができれば, 契約を延長する.
- もし最初の契約で満足できなければ, 条件を変えて (例えば価格の引き下げ) 延長するか, 他の下請負業者に変更する.

その代わりに, 信頼関係の確立されたベンダや下請負業者とは長期的に相互成長できるような関係を築く.

9.2 要請計画プロセス

製品要件と潜在的な調達先を決める

入力

- 調達管理計画 -> 9.1 調達管理計画の出力
- 作業指示書 -> 9.1　調達管理計画の出力
- 他の計画の出力 -> 9.1　調達管理計画

ツールと技法

- 標準形式
- 専門家による判断

出力

- 調達文書
- 評価基準
- 作業指示書の改訂

アジャイル・プロジェクトにおける契約

- 契約
  - 価格の問題
  - 期間の問題
  - 納品物の問題
- 一般的な契約
  - 定率報酬加算式実費精算契約
    - 実コスト+ 実コストに対する一定割合の報酬
      - 請負者のリスク == 0
      - 0 < 発注者のリスク < ∞
    - コストを掛ければ掛けるほど，請負者が儲かる
      - -> コストは増加する一方
  - 定額報酬加算式実費精算契約
    - 実コスト+ 一定額の報酬
      - 請負者のリスク == 0
      - 0 < 発注者のリスク < ∞
    - 請負者はコストを削減するために努力する必要がない
      - -> コストは削減できない
  - 最高額保証付実費精算契約
    - 実コスト+ 一定割合あるいは一定額の報酬ただし上限付
      - 0 < 請負者のリスク < ∞
      - 0 < 発注者のリスク < 上限額
    - 請負者はコストを一定以下に抑える努力をする
      - 発注者はそれに協力する必要はない
  - 固定価格+ 奨励手数料契約
    - 実費+ 報酬+ コスト削減分の一定割合 or 実費+ 報酬- コスト増加分の一定割合
      - 0 < 請負者のリスク < ∞
      - 0 < 発注者のリスク < ∞
    - 請負者，発注者ともにコスト削減に努力する
  - 一括請負契約
    - 固定価格
      - 0 < 請負者のリスク < ∞
      - 発注者のリスク == 0
    - コストの削減はすべて請負者の利益になる

アジャイル・プロジェクトにおける契約

最高額保証付実費精算契約や固定価格+ 奨励手数料契約はこのバリエーション

単価契約

単価× 数量

単価の中にコストと一定割合の報酬がすでに含まれていると考えれば，定率報酬加算式実費精算契約と同じ

定率報酬加算式実費精算契約や定額報酬加算式実費精算契約はこのバリエーション

ソフトウェア開発の場合

よくある契約形式

一括請負契約あるいは単価契約

問題点

契約時にスコープが確定している場合はほとんどない

コストだけではなく，成果物の満足度の重要性が高い

コスト(ほぼ人件費) の規準が曖昧である

アジャイル開発の場合

スコープの変化を前提とするので，一括請負契約は馴染まない

単純な単価契約では，期間がいくらでも伸びてい

-> アジャイル開発プロセスの契約

期間を区切った単価契約

例えば

開発チームA/3ヶ月間/100%稼働

開発チームAの月単価 ¥5百万ならば

¥5,000,000x3x1.0 = ¥15,000,000

最初は比較的な短期的な契約でもよい

必要ならば契約を更新する

信頼が得られれば長期的な関係に育てる(sustainable procurement)

発注者から見て

要求の実現可能性をどう評価するか

開発者の見積もりをどこまで信頼するか

-> 発注者と開発者の信頼関係が必要

-> 発注者側の努力/ 学習も必要

どれだけコストを掛けてもどうしても欲しい要求がある，あるいは要求ベースでないと稟議が通らない

アジャイル・プロジェクトにおける契約

-> アジャイル開発プロセスには適合しないかも知れない

- 単価の妥当性をどう判断するか
  - -> 最終的には全体コストを引き下げられることを納得してもらうしかない
  - -> 実績やアジャイル開発プロセスのアピール
- 稼働率をどう評価するか
  - -> 顧客同室など発注者と請負者の密接な関係が必要
- 発注者の支払うべき金銭以外のコストをどう考えるか
  - 顧客同室, 開発への参加(JAD), 検収テストの作成, イテレーションの評価, レビューなど
  - 発注者も学習/努力しなければ, 全体的なコストを引き下げ, 要求に適合するものを得ることはできないのは真実だが…
  - 発注者側代理人の存在
    - Construction Management方式
    - 開発監理者

納品物件の問題

- 一般的にソース・コードの他に多くのドキュメントやモデルの納品を求められる場合が多い
- アジャイル開発プロセスでは, 「よけいな」ドキュメントやモデルは作らない
- 「よけいな」ドキュメント
  - 開発者にとって
    - 開発において必要ない
    - 製品と一致していない
  - 発注者にとって
    - 「不安」を少しでも取り除きたい
    - 保守に使う
  - ドキュメントの納品
    - 製品と一致していないドキュメントは，発注者にとっても意味のないことを説明する
    - まともなドキュメントが必要ならば，それにはよけいな時間と費用がかかることを説明する
    - コードを実際に見せて，製品に一致しないドキュメントよりも役に立つことを説明する

アジャイル・プロジェクトにおける契約

その上でドキュメントを納品する
できるだけ製品からのドキュメント抽出ツールを利用する

1.2 プロジェクト計画実行プロセス *63*
アジャイル・プロジェクト実行の原則 *64*
5.2 品質保証プロセス *65*
6.3 チーム結成プロセス *66*
アジャイルなチーム *67*
ファシリテータ, コーチ *68*
ワークアウト *69*
7.2 情報配布プロセス *70*
アジャイル・プロジェクトにおける情報共有 *71*
コミュニティ・サイト *72*
情報壁 *73*
コロケーション *74*
9.3 要請プロセス *75*
9.4 調達先選定プロセス *76*
9.5 契約事務プロセス *77*
アジャイル・プロジェクトにおける外注管理 *78*

1.2 プロジェクト計画実行プロセス

活動を行うことによって計画を実行する

入力

- プロジェクト計画 -> 1.1 プロジェクト計画の出力
- 詳細 -> 1.1 プロジェクト計画の出力
- 組織のポリシ
- 予防アクション
- 是正アクション

ツールと技法

- 一般的な管理スキル
- 製品に関するスキルと知識
- 作業権限のシステム
- 状態レビュー・ミーティング
- プロジェクト管理情報システム
- 組織上の手続き

出力

- 作業結果
- 変更要求

アジャイル・プロジェクト実行の原則

- チームの自律性を最大限優先する.
  - チームのやり方に上司や顧客は文句を付けない.
  - 文句を付けるべきは, 自分にとって価値のあるソフトウェアを作ってくれない場合
  - Scrum
    - スプリント期間中はチーム・メンバ以外からの干渉は一切排除される.
    - その代わりにスプリント終了時には結果を出すことを求められる.
- チームは目的を達成するためにできることは何でもやる.
- チームはプロジェクト実行を通じて成長する.

5.2 品質保証プロセス

定期的にプロジェクト全体の実施を評価して, プロジェクトが品質標準を満たしていることに対する信頼を提供する

入力

- 品質管理計画 -> 5.1 品質計画の出力
- 品質制御測定の結果
- 操作定義 -> 5.1 品質計画

ツールと技法

- -> 5.1 品質計画のツールと技法
- 品質の監査

出力

- 品質改善

6.3 チーム結成プロセス

プロジェクト全体の能力を高めるために個人とグループの能力を高める

入力

- プロジェクト・スタッフ -> 6.2 スタッフ調達の出力
- プロジェクト計画 -> 1.1 プロジェクト計画策定の出力
- スタッフ管理計画 -> 6.1 組織計画の出力
- 能力報告 -> 7.3 能力報告の出力
- 外部からのフィードバック

ツールと技法

- チーム結成活動
- 一般的な管理スキル
- 褒賞と認知のシステム
- コロケーション
- 教育

出力

- 能力改善
- 能力評価への入力

アジャイルなチーム

- チーム力は自律的なプロジェクトにとって非常に重要.
  - お互いがお互いのことをよく知っていること.
  - お互いに仕事を助け合うこと.
  - お互いに自分を高め合うこと.
  - これらは「近代的な」チーム作業の原則とは必ずしも一致していないかもしれない.
- 青臭いようだが, 実際にこれらはどうしても必要になる.
  - 実体験:
    - どんなに優秀なメンバを集めても, よい人間関係が築けなければプロジェクトは大失敗する
    - メンバが平凡であっても, よい人間関係が築ければプロジェクトは大失敗はしない (あまり目だった成果は出せないとしても:-)
- チームは自ら学習して進化する組織になる必要がある.
- 例えば
  - 一日に一回はすべてのメンバと言葉を交わす (日常会話でも構わない)
  - 作業中にはヘッドフォンはしない, あるいは個室のドアは開けておく
  - 尋ねられたら必ず答える
  - コーチング -> 次のページ
  - ワークアウト -> 次のページ
  - 飲み会 (やりすぎない:-)
- 参考資料
  - "トヨタ生産方式 改善魂を求めて" by 山田日登志, 1998, 日刊工業新聞社
  - "フィールドブック 学習する組織「5つの能力」- 企業変革をチームで進める最強ツール」 by ピーター・センゲ他, 2003, 日本経済新聞社

ファシリテータ, コーチ

ファシリテータとはよりよくチーム作業やミーティングを進められるように補助する役目. 実際の作業や発言はメンバが行うのだが, それに方向を与えたり, 整理したりする.

- 似たような役割にコーチ, メンタ, コンサルタント, カウンセラなどがある.
- Scrumのスクラム・マスタも似たような面がある.
- どれもチーム力を高めるために非常に重要.

最近では「マネージャ」の代わりに「ファシリテータ」, 「コーチ」と呼ぶ場合もある.

チーム作業やミーティングは, ある意味でルールもゴールも明確でないゲーム. それをうまくやるにはどうすればいいか

コーチングの基本プロセス

- 信頼関係への入り口を構築する
- 会話へ導入する
- 現状を確認する
- 問題・課題を特定する
- 「望ましい状態」をイメージする
- 解決法を検討する
- 課題を達成するためのプランを作成する
- プランを確認する
- 力づける
- フォローを約束する

参考資料

- "Facilitator's Guide to Participatory Decision-Making" by Sam Kaner, 1996, New Society Publishers
- "コーチングの技術 - 上司と部下の人間学" by 菅原裕子, 2003, 講談社現代新書
- "メンタリングの奇跡 - 最速で人が変わる, 組織が変わる!", by マーゴ・マリー, 2003, PHP研究所
- "ファシリテータ型リーダの時代" by フラン・リース, 2002, プレジデント社
- "IT技術者キャリアアップのためのメンタリング技法" by 大浦勇三, 2003, ソフト・リサーチ・センター

ワークアウト

ワークアウトとはチーム力を高めるためのミーティングの一つ (ブレーンストーミングなどと同じようなもの).

ワークアウトの目的はその組織にとって
- 相互信頼を築く
- 権限委譲を進める
- 不必要な仕事を取り除く
- 新たなパラダイムを創造する

ワークアウトのシナリオ例
- 目標を掲げる
- 課題をリストアップする
- 最重要課題を言語化する
- 原因を探求し, 特定する
- 取り組みポイントを絞り込む
- 解決のためのアイデアを出す (ブレーンストーミング)
- 実行プランを確定する
- プランを立てる
- フォローアップする

ワークアウトの実行例
- プロジェクトの最初, 中期, 後期の節目に
- プロジェクト・メンバ全員 (数名 ~ 10名)で
- 半日~2日掛ける

ファシリテータ (コーチ) の役割
- 効果的に進行するために必要な準備を行う
- 方向性を示す
- 参加意欲を高める
- 誰かに場を独占されないよう配慮する
- 積極的に聞く
- 反対意見を歓迎する
- 参加者の発言と進行状況を記録する
- 異なる意見の共通項を見つけるよう促す
- 実行すべきことはその場で決断し, すぐに行動できるよう手配する
- フォローアップを計画する

7.2 情報配布プロセス

プロジェクトの利害関係者にプロジェクトの情報をタイミング良く配布する

入力

- 作業結果 -> 1.2 プロジェクト計画実行の出力
- コミュニケーション管理計画 -> 7.1 コミュニケーション計画の出力
- プロジェクト計画 -> 1.1 プロジェクト計画策定の出力

ツールと技法

- コミュニケーションのスキル
- 情報活用システム
- 情報配布方法

出力

- プロジェクト記録
- プロジェクト報告
- プロジェクト発表

実行プロセス・グループ

アジャイル・プロジェクトにおける情報共有

- 情報共有の方法としてはいくつかの方法がある
  - 構成管理ツール -> アジャイルな変更制御
  - プロジェクト・ホスティング・サイト -> アジャイルな変更制御
  - email (メイリング・リスト)
  - chat
  - Webによるコミュニティ・サイト -> 次のページ
  - 情報壁 -> 次のページ
  - コロケーション -> 次のページ

コミュニティ・サイト

- 例えばPlone

  - plone.org - Welcome to plone.org
  - Python / Zope / CMFベース
  - オープン・ソース
  - 日本語も大体はOK
  - 今もっとも活発に開発が行われているCMSの一つ.
  - 基本的にはほとんどout-of-box (箱から出して即使える)
- その他にもXOOPS (PHPベース) など使えるオープン・ソースのコミュニティ・サイト構築キットはいくつかある
- コミュニティ・サイトの内容例
  - プロジェクト計画
  - ディスカッション
  - Q&A
  - 関連技術情報
  - 上流ドキュメントなど

情報壁

情報壁とは部屋の一部の壁を広く取って, そこにさまざまな情報を張り出す.

例えば進捗グラフ -> アジャイルな進捗報告の原則

モデルやアーキテクチャを表すポンチ絵

ストーリやタスク一覧

情報壁の周りには人が集まりやすい場所を作っておく.

コーヒー・サーバやお菓子を置いたり, ガジェットを置いておいてもいい.

Thoughtworks

Thoughtworks

「生な / 手触りのある」情報

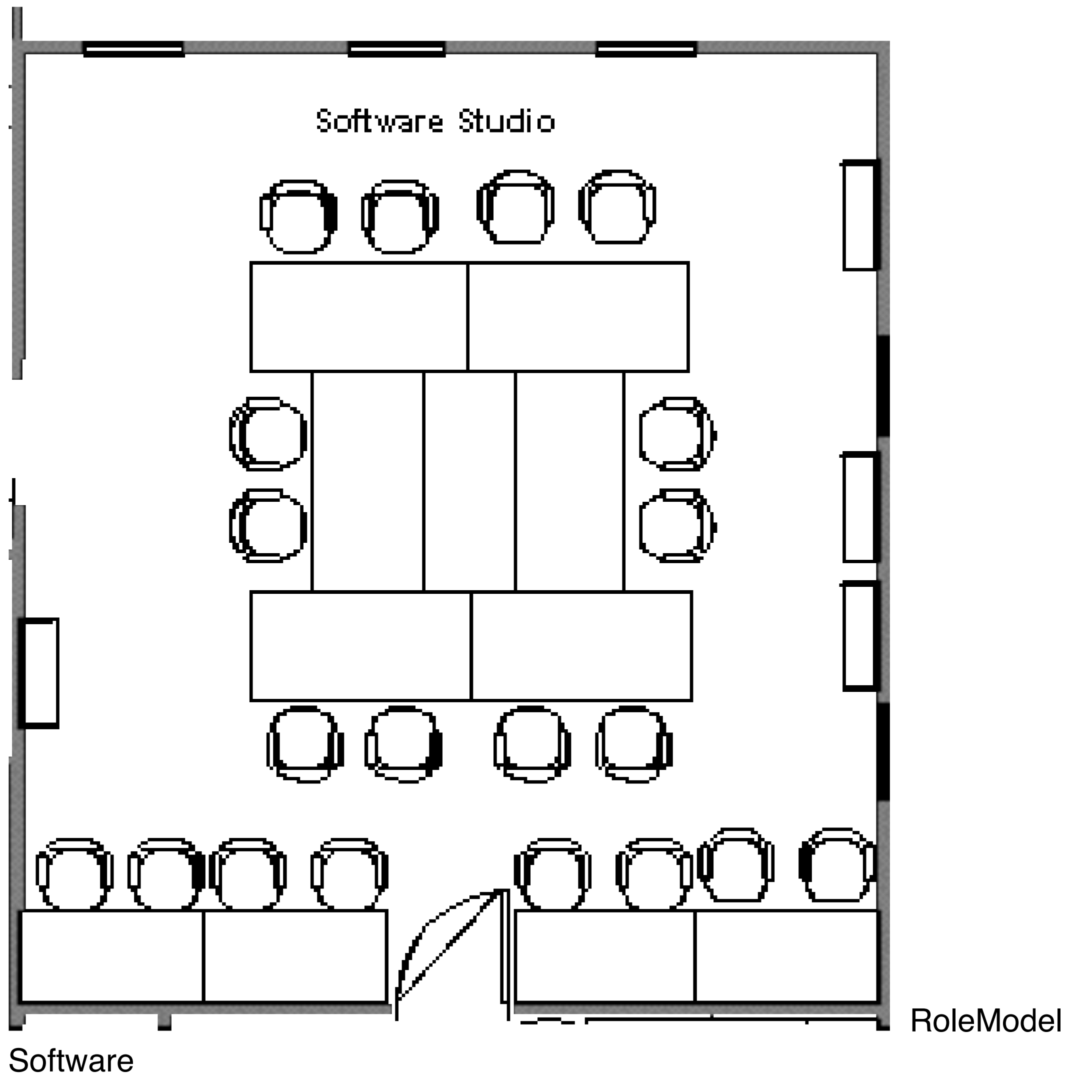
コロケーション

コロケーション

メンバが近い場所に集まって仕事をすること.

情報伝達のコストを大幅に下げ, コミュニケーション阻害による機会損失を低減させる.

コロケーション



RoleModel

Software

9.3 要請プロセス

見積もり, 入札, 提案などを得る

- 入力
  - 調達文書 -> 9.1 調達計画
  - 資格のある販売者の一覧
- ツールと技法
  - 入札会議
  - 広告
- 出力
  - 提案

9.4 調達先選定プロセス

潜在的な販売者の中から選択する

入力

- 提案 -> 9.3 要請の出力
- 評価基準 -> 9.2 要請計画の出力
- 組織ポリシ

ツールと技法

- 契約交渉
- 加重システム
- 選別システム
- 第三者による評価

出力

- 契約

9.5 契約事務プロセス

販売者との関係を管理する

入力

- 契約 -> 9.4 調達先選別の出力
- 作業結果
- 変更要求
- 販売者の送付状

ツールと技法

- 契約変更制御システム -> 1.3 全体的な変更制御
- パフォーマンス報告 -> 7.3 パフォーマンス報告
- 支払いシステム

出力

- 文書
- 契約の変更
- 支払い要求

アジャイル・プロジェクトにおける外注管理

- 外注をアジャイル・プロジェクトにどうやって組み込むかは大きな問題.
  - 引き入れ外注 / 常駐
    - この場合は比較的アジャイル・プロジェクト管理が行いやすい. 他のプロジェクト・メンバと同じ行動を取ることができる
    - ただし契約や就業慣行によっては一部のプラクティスは実施できない場合がある.
      - 例えば残業なしでは売り上げにならないとか...
  - 国内非常駐
    - 一つのやり方は完全に任せてしまう. ただし発注はアジャイル的に行う. つまり
      - 短いイテレーション (1~数週間) ごとに納品してもらう.
      - 取引の浅い下請負業者の場合には最初は短い契約期間.
      - SoW (作業指示書) はストーリ (フィーチャ, バックログ) + 非機能的要件の形で渡すが, 変更があり得ることを確認する.
        - 変更があった場合にどうするかはいろいろなやり方がある.
          - -> アジャイル・スコープ制御の原則
        - いずれにしろスコープではなくタイムボックス優先.
      - できれば発注側メンバが開発側サイトに (部分的) 常駐する.
        - できないとしても週に一回は発注側担当者が開発サイトを訪問する (来てもらうのではなく).
      - 開発は好きにやってね. でもxUnitも一緒に納品してね.
      - つまり発注側はアジャイル・プロジェクト管理パラダイムに基づくが, 開発側には強制しない.
    - もう一つのやり方はもっと密に提携する.
      - CVSレポジトリを共有する.
        - あるいはnightly buildを提供してもらう.
      - コミュニケーション・ツールも共有する.
      - chatなどを用いてデイリー・スクラムに参加してもらう.
      - つまり開発側にもアジャイル・プロジェクト管理パラダイムを強制する.
  - 海外非常駐
    - いちばん難しい.
    - 国内非常駐と同じような手段が取れれば望ましい.
    - できなければ
      - 開発側からコミュニケーションのできる担当者を発注側に常駐させる.

アジャイル・プロジェクトにおける外注管理

発注側から開発サイトに常駐する.
最終的にはアジャイル以前に, コミュニケーションと開発側のモチベーションに依る.

ちなみにCMMレベルは当てにならない.

7.3 実績報告プロセス *80*
アジャイルな進捗報告の原則 *81*
1.3 全体的な変更制御プロセス *82*
アジャイルな変更制御 *83*
プロジェクト・ホスティング・サイト *84*
2.4 スコープ検証プロセス *85*
アジャイルなスコープ検証 *86*
2.5 スコープ変更制御プロセス *87*
アジャイル・スコープ制御の原則 *88*
3.5 スケジュール制御プロセス *89*
アジャイル・プロジェクトのスケジュール制御 *90*
4.4 コスト制御プロセス *91*
5.3 品質制御プロセス *92*
アジャイルな品質制御 *93*
8.6 リスクの監視と制御プロセス *94*
アジャイルなリスク監視 *95*
不測の事態のための予備 *96*

7.3 実績報告プロセス

パフォーマンス情報を収集して, 配布する

入力

プロジェクト計画 -> 1.1 プロジェクト計画策定の出力

作業結果 -> 1.2 プロジェクト計画実行の出力

その他のプロジェクト記録 -> 7.2 情報配布の出力

ツールと技法

パフォーマンス・レビュー

変動分析

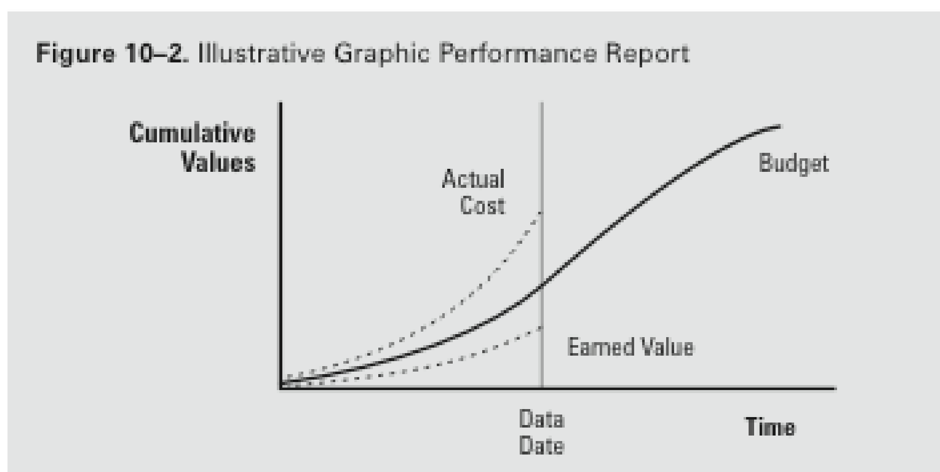
傾向分析

獲得価値分析

-> 7.2 情報配布のツールと技法

出力

パフォーマンス報告



変更要求

アジャイルな進捗報告の原則

アジャイル・プロジェクトでももちろん進捗は測定される. ただしプロジェクトの状態を可視化することが目的で, プロジェクトの状態を計画に合わせるためではない.

原則

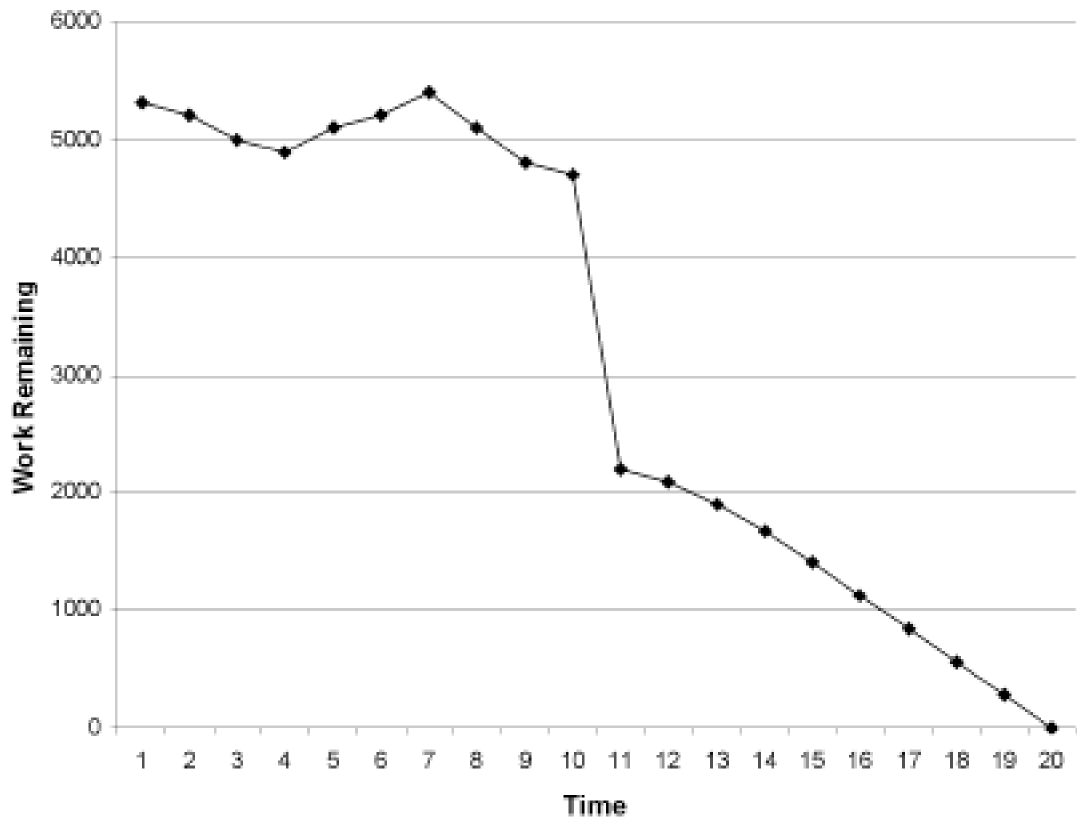
- 進捗報告にストレス / コストを掛けない.
  - できる限り自動化ツールを使う.
- 進捗報告はメンバの能力を測定するためのものではない.
  - 公表するときにはプロジェクト全体の値として公表する.
- 進捗はときどき測るものではなく, 常に測定する.
  - 週1回, 月1回の報告ではそれに基づいてプロジェクトの舵をきることができない.
- 進捗の尺度は客観的で, 直接的で, 計測可能でなければならない.
  - 「だいたい1/3くらい終了」は進捗ではない, 思い込み.
    - -> 90%完了シンドローム
  - ソフトウェアの価値とは直接関係のないコード行数を測っても意味はない.
  - 検証規準をパスしていないものは進捗ゼロと測る.
    - -> バイナリ・トラッキング (二値追跡)
  - 基本的にはスコープの達成度合いを, スコープの検証基準に基づいて測定する.
- 報告された結果はちゃんとフィードバックする.
  - 管理者が隠し持っていても意味はないし, 報告者のモチベーションにならない.
  - 全員がいつでも気にしていられるように.

プロジェクト・ホスティング・サイトなどを使うと上記のことがほぼ実現できる.

進捗を可視化した例 (Scrum, スプリント当たりの残存バックログの推移)

- このような表を廊下や情報壁に貼りだしておく

アジャイルな進捗報告の原則



ただし非機能的要件については別に考える必要がある.

1.3 全体的な変更制御プロセス

プロジェクト全体にわたる変更を調整する

入力

- プロジェクト計画 -> 1.1 プロジェクト計画策定の出力
- パフォーマンス報告 -> 7.3 パフォーマンス報告
- 変更要求

ツールと技法

- 変更制御システム (正式な手順書のこと)
- 構成管理
- パフォーマンス測定
- 計画の追加
- プロジェクト管理情報システム -> 1.1 プロジェクト計画策定

出力

- プロジェクト計画の改訂 -> 1.1 プロジェクト計画策定の出力
- 是正アクション -> 1.2 プロジェクト計画実行の入力
- 教訓

アジャイルな変更制御

- 「変更」はアジャイル・プロジェクト管理の肝である.
- 中心になるのは成果物の変更管理.
  - 「餅は餅屋」なのだから, 我々は自分たちが作るものをソフトウェア・ツールで管理するべきである.
  - 往々にして「紺屋の白袴」だが:-)
- 変更管理ツール - 必須
  - できるだけレポジトリを共有できるものがよい.
    - CVS - デファクト・スタンダード
      - Unix由来で非常に広く使われている
      - オープン・ソースで低コスト (Unixのノウハウがないと多少無形のコストが必要かも)
      - 枯れている, ただし制約がある (特にディレクトリ関係)
      - ほとんどの開発環境と統合されている
    - Subversion - これからのデファクト・スタンダード
      - WebDAVベース, オープン・ソース
      - CVSの欠点を克服
      - 未成熟
    - 商用製品
      - IBM Rational I 製品 | ソフトウェア構成管理 (ClearCase)
      - ”高速”ソフトウェア構成管理ツール PERFORCE

プロジェクト・ホスティング・サイト

- プロジェクト・ホスティング・サイト - 必要
  - プロジェクトに関する情報を集中的に管理できるWebベース・ツール
    - オリジナルはSourceForge
      - @ SourceForge.net: Welcome
      - 以前はオープン・ソースだったが今は開発停止.
      - 最新版は商用化されて, 販売されている
      - 同様のものにSourceCastがある.
        - @ CollabNet - CollabNet® SourceCast®
        - こちらはASP形態での提供 (約30万円 / 人年)
        - 日本では住商エレなどが代理店
    - GForge - オープン・ソースのデファクト・スタンダード
      - @ GForge CDE : Collaborative Development Environment
      - オープン・ソース版SourceForgeからフォーク
      - インストール / 管理はそれないに大変だが, 非常に強力
      - 複数プロジェクトのサポート
      - 日本語も大体OK
      - Apache / PostgreSQL / PHP
      - 機能
        - フォーラム (掲示板)
        - バグ追跡
        - タスク定義 / 割り当て / 進捗追跡
        - ドキュメント・データベース
        - 投票
        - ニュース
        - CVSインタフェース
        - リリース・マネージャ
    - XPlanner
      - @ XPlanner Home
      - XPに適したオープン・ソース・ツール
      - Java (Tomcat) / MySQL, PostgreSQL
      - 日本語を通すためにはパッチが必要
        - @ Tomcat+MySQLで日本語化されたXPlannerをセットアップ
      - 機能
        - イテレーション

プロジェクト・ホスティング・サイト

- ストーリ
- タスク
- メトリックス, 統計, 状態
- Wiki

重要なのはこれらをインストールしてメンバに与えるだけでは「絶対に」機能しないということ.

- だれかがお手本として使ってみせる.
- ミーティングではこのサイトをプロジェクタを使って映しながら, 書き込みながら進行させる.
- 多少うるさいかもしれないが, サイトの内容がアップデートされたらメンバにemailが飛ぶような設定にできればしておく.

2.4 スコープ検証プロセス

プロジェクト・スコープを正式に検収する

入力

- 作業結果 -> 1.2 プロジェクト計画実行
- プロダクト文書
- WBS -> 2.3 スコープ定義の出力
- スコープ宣言 -> 2.2 スコープ定義の出力
- プロジェクト計画 -> 1.1 プロジェクト計画策定の出力

ツールと技法

- インスペクション

出力

- 正式な検収

アジャイルなスコープ検証

ここで言っているスコープの検証とは, スコープ (要件) が達成されていることをどうやって検証するか, ということである.

PMBOKでは「インスペクション」と言っているが, 非常に広い範囲のインスペクションであって, いわゆるソフトウェア・インスペクションには限らない.

アジャイル・プロジェクトにおける原則は「間違って作ったものを後から直すのではなく, 最初から正しいものしか作らない」

- CMM (Level 5) と同じ
- ただし「何が正しいかは後から変わり得る (あるいは最初は分からない)」
  - 顧客の都合で
  - 開発者の理解が進歩したため
  - 技術が変化したため
  - 状況が変わったから
  - 「何が正しいか分からないところは放っておく」
    - むりやり分からない / 検証不可能なドキュメントを作らない
- だから「正しさが変わったら (分かったら) 躊躇なくそれに追随する」
  - それを制御するのがアジャイル・プロジェクト管理

そのためのアジャイル技法

- テスト駆動開発
- xUnit
  - この二つはどの分野のソフトウェアにも使えるというわけではない
- 継続的な統合
- 機能テスト

それ以外の技法

- ソフトウェア・インスペクション
- 契約による設計 (Design by Contract)
  - テスト駆動開発と共通点がある
- 形式的仕様記述
- モデル駆動開発
  - @ Project Technology, Inc. - BridgePoint, xtUML
  - @ Eclipse Projects (Eclipse Modeling Framework)
  - @ Eclipse Technology Projects - Generative Model Transformer
- @ FrontPage - MockObjects

2.5 スコープ変更制御プロセス

プロジェクト・スコープの変更を制御する

入力

- WBS -> 2.3 スコープ定義の出力
- 実績報告 -> 7.3 実績報告の出力
- 変更要求
- スコープ管理計画 -> 2.2 スコープ計画の出力

ツールと技法

- スコープ変更制御システム -> 1.3 全体的な変更制御
- 実績測定 -> 7.3 実績報告
- 計画の追加 / 変更

出力

- スコープ変更
- 是正アクション
- 教訓
- ベースラインの調整

アジャイル・スコープ制御の原則

- アジャイル・スコープ制御の原則
  - 誰がスコープ変更を要求し, 受け入れるのか
    - 顧客が要求するスコープ変更と, 開発側の事情から生じるスコープ変更がある.
    - Scrum
      - 要求 - 顧客側ではプロダクト・オーナただ一人, 開発側では任意のメンバ
      - 受け入れ - スプリント計画ミーティング (顧客+開発)
    - XP
      - 要求 - 顧客側のチーム, 開発側のチーム
      - 受け入れ - 計画ゲーム (顧客+開発)
  - どういうタイミングでスコープを変更するか
    - Scrum
      - スプリント (30暦日のイテレーション) の境界のみ
    - XP
      - 任意のタイミング
  - スコープ変更とタイムボックス
    - Scrum
      - タイムボックスは変えない. つまり一定のタイムボックス (30暦日) 内に実現可能と予想されるスコープ変更のみが受け入れられる
    - XP
      - 変更されたスコープを実現可能と予測されるタイムボックスに変更する. タイムボックスが変わらないとしたら, タイムボックスに合わせてスコープを調整する.
  - スコープが実現不可能な場合の対処
    - Scrum, XP
      - タイムボックスは変えない. 実現できないスコープは後のイテレーション (スプリント) に回される.
    - FDD
      - タイムボックスとスコープを一定の範囲で同時に調整する.
      - 両方を変えるとなると「いつ変えるべきか」「どう変えるべきか」という問題が発生する.
        - これらは自由度を大きくする代わりに管理コストを大きくする.
        - 実際FDDでは進捗の追跡を工夫している.

アジャイル・スコープ制御の原則

ただしいつ, どう変えるべきかについてはFDDは何も言っていない.

- スコープとタイムボックスの関係
  - Scrum
    - 勘と経験に基づく予測 (他の方法を用いても構わないが)
  - XP
    - プロジェクト速度の予測と測定による漸進的精度向上

3.5 スケジュール制御プロセス

プロジェクト・スケジュールの変更を制御する

入力

- プロジェクト・スケジュール -> 3.4 スケジュール策定の出力
- 実績報告 -> 7.3 実績報告の出力
- 変更要求 -> 1.3 全体的な変更制御の入力
- スケジュール管理計画 -> 3.4 スケジュール策定の出力

ツールと技法

- スケジュール変更制御システム -> 1.3 全体的な変更制御
- 実績測定 -> 7.3 実績報告
- 計画の追加 / 変更
- プロジェクト管理ソフトウェア -> 3.4 スケジュール策定
- 変動分析

出力

- スケジュールの改訂
- 是正アクション
- 教訓

アジャイル・プロジェクトのスケジュール制御

- アジャイル・プロジェクトでは, 基本的にスケジュールを固定して, スコープを変化させる.
  - なぜならばあるスコープがどれだけの時間で実現できるか, 予測が難しいから.
- あるスケジュールで機能や品質, 性能が不足している場合には, 次の二つの方法がある.
  - スケジュールを変える, つまりイテレーションを追加する.
    - 普通は余計なコストがかかる
  - 機能や品質, 性能が不足したままリリースする, あるいはプロジェクトを中止する.
- スケジュール変更のリスク
  - どちらも許されないとしたら, 問題解決型 / 価値創造型のプロジェクトではリスクが高すぎて手を付けることができない.
  - スコープの変更は許されないが, スケジュールの変更が許されるのならば従来のプロジェクト管理手法を用いることができる.
  - 顧客との間の信頼が必要.
  - 短い複数のイテレーションに分割して, リスクを下げる (前倒しにする).
- 人, 品質, 時間, スコープの4つのパラメータがあるとしたら, スケジュールが押してきたときにソフトウェア開発プロジェクトで効果的に動かせるのはスコープだけ.
  - 人を増やせばプロジェクトはもっと遅れる (ブルックスの法則)
  - 品質を少しだけ下げてもいいと思うと品質は劇的に下がる
  - 時間を延ばしてもいいとすると時間はいつまでも延び続ける
    - ときには違うメンバでスクラッチから作り直した方が低いコストでよい結果が得られる場合もある

4.4 コスト制御プロセス

プロジェクト予算の変更を制御する

入力

- コスト・ベースライン -> 4.3 コスト予算化の出力
- 実績報告 -> 7.3 実績報告の出力
- 変更要求
- コスト管理計画 -> 4.2 コスト見積もりの出力

ツールと技法

- コスト変更制御システム -> 1.3 全体的な変更制御
- 実績測定 -> 7.3 実績報告
- 獲得価値管理 -> 7.3 実績報告
- 計画の追加 / 変更
- コンピュータ・ツール

出力

- コスト見積もりの改訂
- 予算の改訂
- 是正アクション
- 完成時の見積もり (獲得価値管理)
- プロジェクト終結
- 教訓 -> 1.3 全体的な変更制御

5.3 品質制御プロセス

プロジェクトのある側面を監視して, 適切な品質標準を満たしていることを確認し, 不適切な部分を減らす方法を特定する

入力

作業結果 -> 1.2 プロジェクト計画実行の出力

品質管理計画 -> 5.1 品質計画の出力

操作定義 -> 5.1 品質計画の出力

チェックリスト -> 5.1 品質計画の出力

ツールと技法

インスペクション

制御チャート

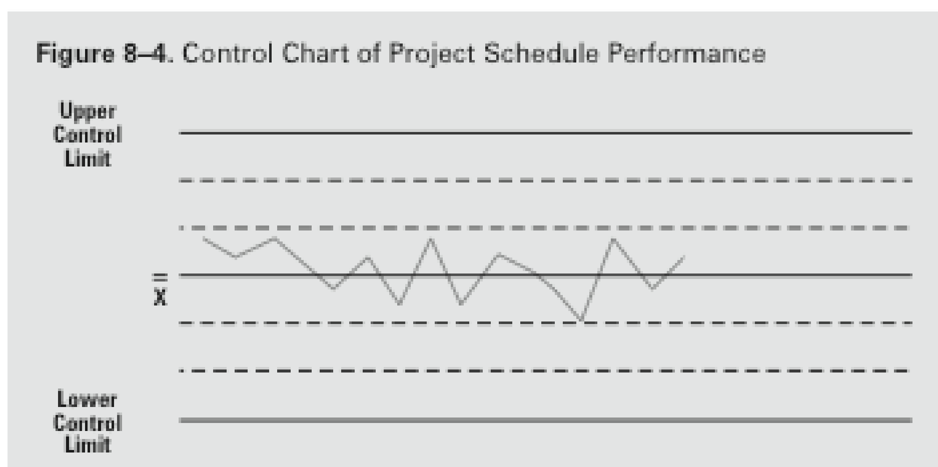
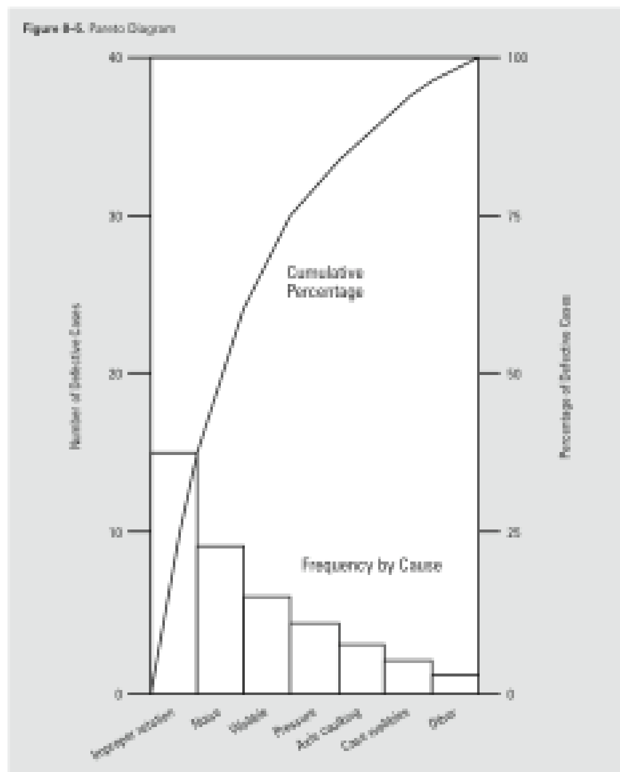


Figure 8–4. Control Chart of Project Schedule Performance

パレート図

5.3 品質制御プロセス



- 統計的サンプリング
- フローチャート -> 5.1 品質計画のツールと技法
- 傾向分析

出力

- 品質改善 -> 5.2 品質保証の出力
- 受け入れ判定
- 再作業
- チェックリストの完成 -> 5.1 品質計画
- プロセスの調整 -> 1.3 全体的な変更制御

アジャイルな品質制御

アジャイル・プロジェクトにおいても, 重要な品質要件があれば同様の品質制御が必要になる.

ソフトウェアにおける品質制御には次の二つの方法がある.

1. 機能がすべて揃って稼働するソフトウェアについて品質要件を実現していく
2. 機能要件を実現しながら, 品質要件も同時に実現していく

1の方法では, 品質要件を実現するために再作業が大きく必要になる可能性がある.

2の方法では, 再作業の可能性が低くなるが, 通常は機能が完備していないソフトウェアの品質属性を測定することは難しい.

繰り返し的な開発手法を使えば, 機能要件の実現と品質要件の実現を並行して行っていくことができる (2).

2の方法で品質を制御するためには, 機能コンポーネントに対して品質属性を割り当てるが必要になる.

- 例えばコンポーネントごとに消費メモリ量を割り当て, ソフトウェア全体で64KBのROMに収まるようにする.
- 今までに作られてコンポーネントの合計消費メモリ量が予定の一定範囲を超えていたら, リスクが顕在化したことになる.

このような方法を技術パフォーマンス測定 (Technical Performance Measurement) と呼ぶ.

20-80の法則

- 問題の80%は20%の場所で生じている (経験則)
  - 例えば
    - 英単語の80%は20%の語彙で占められている.
    - バグの80%は20%のモジュールに存在する.
    - 仕事の80%は20%にメンバが行っている.
- したがってその20%の場所に集中すれば, 80%の問題を解決することができる.
- 20%の場所を見つけるためにはパレート図などを使う.

8.6 リスクの監視と制御プロセス

残っているリスクの監視, 新たなリスクの特定, リスク低減計画の実施, その効果の測定などをプロジェクトのライフサイクル全般にわたって行う

入力

- リスク管理計画 -> 8.1 リスク管理計画の出力
- リスク対応計画 -> 8.5 リスク対応計画の出力
- プロジェクト・コミュニケーション -> 7.3 実績報告の入力
  - 問題ログ, アクション項目一覧など
- 追加されたリスク特定と分析
- スコープ変更 -> 2.5 スコープ変更制御の出力

ツールと技法

- プロジェクト・リスク対応の監査
- 定期的なプロジェクト・リスクのレビュー
- 獲得価値分析
- 技術パフォーマンス測定
- リスク対応計画の追加

出力

- 回避計画
- 是正アクション
- プロジェクトの変更要求
- リスク対応計画の改訂
- リスク・データベース
- リスク特定チェックリストの改訂

アジャイルなリスク監視

アジャイル・プロジェクト管理では, リスクは必ずしも避けなければならないものとは限らない. またリスクを避けるために大きすぎるコストを掛けることはできない.

リスク監視にもあまりコストを掛けすぎるわけにはいかない. リスク監視は特別なことではなく, 日常的に行う.

リスク監視対象は大きく分けて二つある. 一つはプロジェクト内部, もう一つはプロジェクト外部.

- プロジェクト内部のリスク監視
  - 常にメンバ全員の状態を把握する.
    - -> Scrumのデイリー・スクラム (毎日の短時間の進捗ミーティング)
    - 毎日, プロジェクト管理者が必ずメンバ全員の一人一人と直接話をする機会を持つ.
      - 数分程度で構わない, 軽い話題でも構わない.
    - 本来はプロジェクト管理者に限らず, すべてのメンバが互いに他のメンバの状況を大体把握しているとよい.
    - 開発ログ (ジャーナル, 日記) をメンバ全員が付ける. 簡単でいいので, 毎日. 単に習慣.
      - ツールとしてはForge系ツール, Wiki / Blogなどが使える.
      - 代替案としては「CVSの変更ログをきっちり付ける」でも許す.
      - emailはうっとうしいかも.
      - リスク監視だけでなく, 作業の自覚化やレトロスペクティブにも大変役立つ.
  - 常に成果物の状態を把握する.
    - -> XPの継続的統合と機能テスト, 共同所有, ペア・プログラミング, リファクタリング
    - 従来の全員が集まってやるコード・レビューは効率がよくない場合が多い.
      - インスペクションはやる気持ちがあれば, 悪くないかもしれない.
    - チーフ・アーキテクトが定期的にすべてのコードをレビューする.
      - 問題のあるコードはそのコードを書いた人と一緒に書き直す.
      - そのような意味では, 直接の実装担当を持たないシニア・エンジニアがいる価値のある場合がある.
    - 本来はチーフ・アーキテクトに限らず, すべてのメンバがプロジェクトの全成果物について一通り把握しているとよい.

アジャイルなリスク監視

- 最低限のパラメタについて傾向を把握する.
  - 例えば検収条件をパスしたフィーチャの数の推移
  - 簡単にグラフ化して, みんなの目に触れるようにする
  - あまりに多くのパラメタを対象にしても意味がない
- 常に計画と比較し, 「穴」を見つける.
  - -> 毎週の計画レビュー・ミーティング

プロジェクト外部のリスク監視

- 外部ステークホルダ (顧客, 上司, 顧客組織, 自組織, 社会状況, メディア…) と接触し, 情報を取り入れる.
- やりすぎる必要はないが, アンテナの感度を上げておくのはプロジェクト管理者の役割.
- 他のメンバはそんな余裕がない場合が多い.

発見したリスクに対してどのように対処するかは, リスク対応の問題.

不測の事態のための予備

- 不測の事態のための予備 (Contingency Reserve)
  - 常に予測不能あるいは計測不能なリスクというものが存在する.
  - それに対応するためにはまとめてどこかにバッファ (予備) を取っておく必要がある.
  - これが不測の事態のための予備
- 予備を取ればそれだけ効率が悪くなったり, コストが高くなったりする.
- 予備が全くなければ, プロジェクト内容によってはリスクが高くなる.
- 予備はこの二つのトレードオフで決まる.
- 不確定性と予備

不確定性	予備
0%	15%
10%	25%
20%	50%
30~50%	100%
50%以上	200%

- もし問題が起きなければ, 予備は開発側の取り分になる.
- もう一つの方法
  - 問題が起きたら, それを率直に顧客に話し, そのコストを顧客が負担する.
  - 開発側は予備を取らない.
  - 信頼関係があれば可能
  - 余計な予備を取られずに済む.
- -> どちらがリスクを負うかという契約の問題にも絡んでくる.
- 参考資料
  - "Agile Management for Software Engineering" by David J. Anderson, 2004, Pearson Education, Inc.

9.6 契約完了プロセス *98*
アジャイルな検収 *99*
7.4 完了手続きプロセス *100*
レトロスペクティブ *101*
アジャイル・プロジェクトの終了 *102*

9.6 契約完了プロセス

契約の終了と解決

入力

- 契約文書

ツールと技法

- 調達の監査

出力

- 契約ファイル
- 正式な検収と終結

アジャイルな検収

アジャイル・プロジェクトでは検収は最後の一回だけではなく, 以下のタイミングで行われる.

1. 「常に」
2. イテレーションごとに
3. 最後のリリース時

1は必ずしも顧客側に公開されているとは限らない. しかし, 何かの事情で急にプロジェクトが中断することがあっても何らかの成果を得ることができる.

- 契約上の問題は別として.
- なぜならば請負側から言って, 急に契約が中断されてなおかつそれまでの成果をみんな持って行かれてはたまらないから.

2は顧客側と請負側で同意された客観的な検収条件が必要.

- 例えばテスト・コード, あるいは性能一覧など.
- このようなものがないと請負側は安心して, 焦点を絞ったアジャイルな作業ができない.
- 顧客側にも決断力と勉強が必要になる. それは最終的に顧客の価値になるはず.
- 問題点
  - 誰が検収条件を用意するのか
    - 現状では顧客側では用意しきれないことが多い.
      - なぜならばそもそも顧客側にリソースが足りないため, あるいは知識が足りないために外注している.
    - XPでは顧客側と請負側が協力して検収条件 (機能テスト) を作成する.
      - そのためにはそれなりのコストが必要 -> ある意味当然のはずだが.
      - 顧客側と請負側の信頼関係が必要.
  - どこまで検収条件を信頼できるのか
    - 一部の分野を除いて, 現状ではテスト・コードに「すべてを」盛り込むことはできない.
    - 顧客側が疑心暗鬼になると非常に手間ばかりかかって, 意味のないテスト条件になる.
      - -> コスト増大 / 時間増加 / モチベーション低下
    - テスト・コードには含まれないが当然あるべき (と顧客が思っていた) 機能や品質がなかった場合, それは誰がどう補償するのか
    - これもやはり顧客側と請負側の信頼関係及び共同作業が必要.

2がちゃんとできていれば, 3は形式的なものになる.

アジャイルな検収

アジャイルな調達では, 下請負業者 (sub-contractor) やベンダ (vendor) との長期的な相互成長関係を維持 / 育成することが重要になる.

-> 例: トヨタ生産方式

ただしケイレツ化, 馴れ合いを避ける工夫は必要

下請負業者にどのようにモチベーションを与えるか

7.4 完了手続きプロセス

情報の生成, 収集, 配布によってプロジェクトあるいはフェーズを終結させる

入力

- 実績測定文書
- プロダクト文書
- その他のプロジェクト記録 -> 7.2 情報配布の出力

ツールと技法

- 実績報告のツールと技法 -> 7.3
- プロジェクト報告 -> 7.2 情報配布の出力
- プロジェクト発表 -> 7.2 情報配布の出力

出力

- プロジェクト・アーカイブ
- プロジェクト終結
- 教訓 -> 1.3 統合変更管理の出力

レトロスペクティブ

レトロスペクティブ (retrospective) とは, 従来のポストモータム・ミーティング (post-mortem meeting) やいわゆる反省会と呼ばれているものと大体同じ. 振り返り.

- 我々のレトロスペクティブがポストモータムなどと異なるのは, プロジェクトの最後に一回だけ行うわけではないこと.
- 検屍とか反省とかネガティブになる必要はない.
  - 戦犯捜しではない.
  - 「何が悪かったか」ではなく「どうすればもっと良くなるか」
- インスペクションの形式でやる場合もある.
  - それなりにコストがかかる.

目的

- 最後の一回だけではなく, プロセス中に (in-process) 頻繁にレトロスペクティブを行うことによってプロジェクトの自律的な組織化の創発をもとめる.
- ただしプロジェクトが何かの結果を出していなければ机上の空論になるので, イテレーションごとに行うのがよい.
- あまり頻繁すぎるとプロジェクトが安定しない可能性がある.

時間

- プロジェクトによる. 短ければ半日, 長ければ2日間.
- 長ければいいというものではない.
- アジェンダ例 (午後半日, その後飲み会)
  - 開始(説明、計画) - 約30分
  - 事実の洗い出し - 約1時間
  - 問題点整理 - 約1時間
  - 改善提案 - 約1時間
  - まとめ - 約30分

タイミング

- イテレーションの終了ごと.
- 全イテレーションの終了時には少し大がかりにしてもよい.

参加者とその役割

- 全プロジェクト・メンバ (顧客や上司は参加しない)
- 誰かがファシリテータの役割を果たす.
- 基本的に全員対等

準備

- イテレーションの記録やソースコード, テスト記録など, プロジェクト・デー

タに必要に応じてアクセスできるようにしておく.

内容

重要なのは以下の3点.

- 今回やったことの中でうまく働かなかったのは何か
  - うまく働かせるためにはどうすればよいか
- 今回やったことの中でうまく働いたのは何か
  - もっとよく働かせるためにはどうすればいいか
- 今回やらなかったけれど今度やってみたいことは何か
  - どうやってやるか

エンジニアリングに関するものも, プロセスに関するものも, 人間関係に関するものもすべてOK.

その代わり「問題がある」だけでは駄目で, 「じゃあどうするか」が必要

ルール例

- 人の言っていることを遮らない
- 人の言っていることは素直に受け止める
- 常に自分の立場から発言する
- 人をからかったり, 揚げ足を取ったりしない

結果

内容で挙げた3点をまとめる.

結果はできるだけ清書し直すなどの無駄を省いて, そのまま公開できるとよい (ホワイトボード + プリンタ / デジカメ, イーゼル・パッド + ポストイット, プロジェクタ + ソフトウェア・ツールなど).

次のイテレーションの間, 常に全メンバの目に触れるようにしておく.

参考資料

"Project Retrospective - a Handbook for Team Reviews" by Norman L. Kerth, 2001, Dorset House

"アジャイルソフトウェア開発" by アリスター・コーバーン, 2002, ピアソン・エデュケーション

アジャイル・プロジェクトの終了

- アジャイル・プロジェクトの終了時には以下のような作業を行う.
  - プロジェクトの記録や成果物をまとめて, 今後に活かせるようにする.
    - 組織にとっての最大の資産はプロダクトそのものではなくて, プロダクトを作り上げたという経験である.
    - 今までにプロジェクト・ホスティング・サイトやコミュニティ・サイトを使っていたら, それをまとめればよい.
    - できれば (ソース・コードも含めて) いつでもアクセスできるようになっているとよい.
      - 意外とソース・コードの再利用が行われる.
      - ドキュメントのフォーマットも再利用される.
      - その中からテンプレートやコンポーネントが自然発生的にできてくる.
        - 最初から再利用を意識したコンポーネントを作るためには, そうでない場合に較べて2~3倍のコストが必要.
        - 再利用コンポーネントを使うことによってコストが1/5になったとして, 元を取るためには何回再利用しなければならないか
        - 手直しなしで3回以上再利用されるコンポーネントの数は実はそれほど多くない.
          - 最初から汎用的な利用を見込んだコンポーネント設計は難しい.
  - 打ち上げをする.
    - だらだら終わるとメンバの中で区切りがつかない.
    - リズムを作ることができる.
- プロジェクトの引き継ぎ
  - ソフトウェア開発は基本的には終わりのない作業.
    - 組み込み制御系でも最近はプロダクト・ラインやオンライン・アップデートのせいでメンテナンスが必要になっている.
  - ソフトウェア開発を知識変換過程として考えると, ソフトウェアのリリースによって開発メンバから保守メンバに一気に入れ替わってしまうと (再学習) コストが高くつく.
  - かといって開発メンバがいつまでも過去のプロジェクトを引きずっていると, 常に割り込みがかかって開発効率を大きく落とすことになる.
  - プロジェクト・メンバの交代は徐々に行う.
    - 例えば終了2ヶ月前, 1ヶ月前, 終了時, 1ヶ月後, 2ヶ月後のタイミングで, メンバを二人ずつ入れ替えていく.
    - こうすれば10人のチームを4ヶ月で入れ替えることができる

アジャイル・プロジェクトの終了

- 開発メンバは2ヶ月後にはこのプロジェクトから完全に手を引いて新しいプロジェクトに専念できる
- プロジェクトの文化を継承することができる

保守メンバは複数のプロジェクトの掛け持ちでもよい.

- 開発メンバはできるだけ一つのプロジェクトに専念させる.
- 保守だけではモチベーションを維持できない場合には, 定期的に開発に参加させる.

1. 統合管理 104
2. スコープ管理 105
3. 時間管理 106
4. コスト管理 107
5. 品質管理 108
6. 人材管理 109
7. コミュニケーション管理 110
8. リスク管理 111
9. 調達管理 112
10. 知識管理 113

1. 統合管理

1.1 プロジェクト計画策定
1.2 プロジェクト計画実行
1.3 全体の変更制御

2. スコープ管理

2.1 立ち上げ
2.2 スコープ計画
2.3 スコープ定義
2.4 スコープ検証
2.5 スコープ変更制御

3. 時間管理

3.1 活動定義
3.2 活動配列
3.3 活動期間見積もり
3.4 スケジュール策定
3.5 スケジュール制御

4. コスト管理

4.1 資源計画
4.2 コスト見積もり
4.3 コスト予算化
4.4 コスト制御

5. 品質管理

- 5.1 品質計画
- 5.2 品質保証
- 5.3 品質制御

6. 人材管理

6.1 組織計画
6.2 スタッフ調達
6.3 チーム結成

7. コミュニケーション管理

[7.1 コミュニケーション計画]
[7.2 情報配布]
[7.3 実績報告]
[7.4 完了手続き]

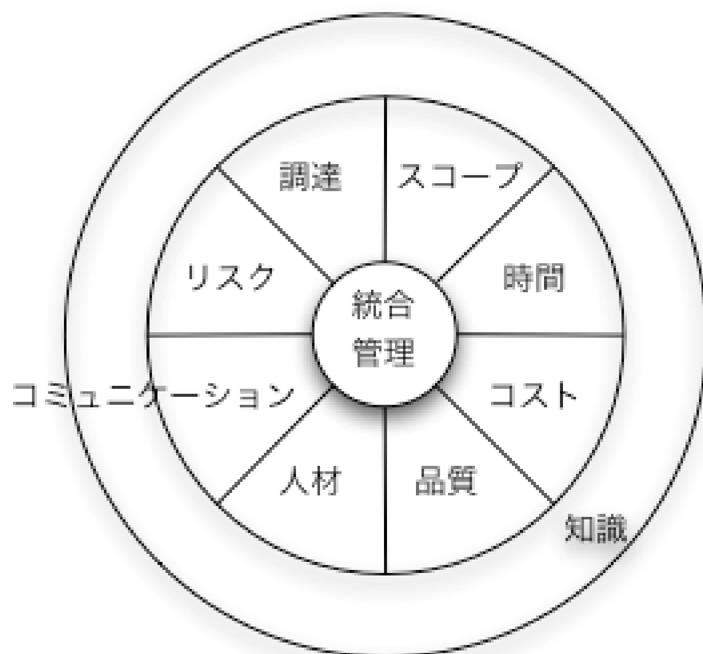
8. リスクマネジメント

- 8.1 リスク管理計画
- 8.2 リスク特定
- 8.3 定性的リスク分析
- 8.4 定量的リスク分析
- 8.5 リスク対応計画
- 8.6 リスク監視と制御

9. 調達管理

9.1 調達計画
9.2 要請計画
9.3 要請
9.4 調達先選別
9.5 契約事務
9.6 契約完了

10. 知識管理



ソフトウェア開発の過程を知識変換過程としてとらえる

プロジェクト管理とは戦略を立て, 戦略を実行することであると考える