13 ソフトウェア工学

Software Engineering

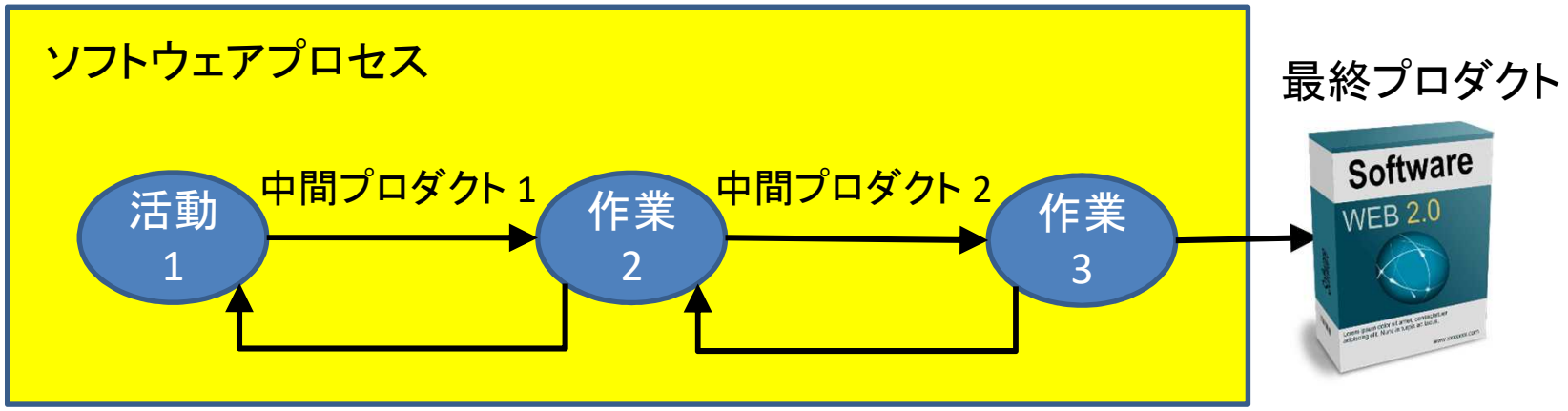
ソフトウェアプロセス

SOFTWARE PROCESS

ソフトウェアプロセスとは

■ソフトウェアプロセス：

ソフトウェアプロダクト（製品）を作り出すための，互いに関連する活動(activity)の集合



ソフトウェアプロセスの設計と記述

■つぎの4項目について設計し，記述する

1) 活動(activity)
　　各活動の内容と順序

2) プロダクト(product)
　　各活動結果の生成物（文書，図表，プログラムなど）

3) 役割(role)
　　担当する人々の職種（プロジェクト管理者，プログラマなど）

4) 事前条件，事後条件(pre- /post-condition)
　　各活動の前と後で成り立つ条件

プロセスが含むべき活動

1) 要求(requirement)
ソフトウェアの仕様（機能，運用上の制約）を定義

関連する言葉
要求定義(requirements definition)
要求分析(requirements analysis)
要求獲得(requirements elicitation)
仕様記述(specification)

2) 設計(design)
ソフトウェアアーキテクチャ，データモデル，クラス，インタフェースを記述

3) 実装(implementation)
プログラミングにより実行可能なシステムを構築

4) 確認(validation)
顧客の要求を満たすことをテスト等により確認

関連する言葉
検証(verification)
テスト(testing)

5) 進化(evolution)
顧客の要求の変化に対応して修正・改訂

関連する言葉
保守(maintenance)
発展(evolution)

ソフトウェアプロセスモデル

■ソフトウェアプロセスモデル

- 良く見かけるソフトウェアプロセスを単純化して表現したもの
- これを拡張／詳細化して特定のプロセスを生成するのに使用

■今回はつぎの4つのモデルを学ぶ

1) ウォーターフォール開発(waterfall model)

要求→設計→実装→確認→進化の一連の活動を分離して実施.

2) インクリメンタル開発(incremental development)

要求，設計，実装，確認をインターリーブ(interleave)して実施.
短期間でバージョンアップする一連の版(versions)としてシステムを開発.
過去の版に対する増分(increments)を開発.

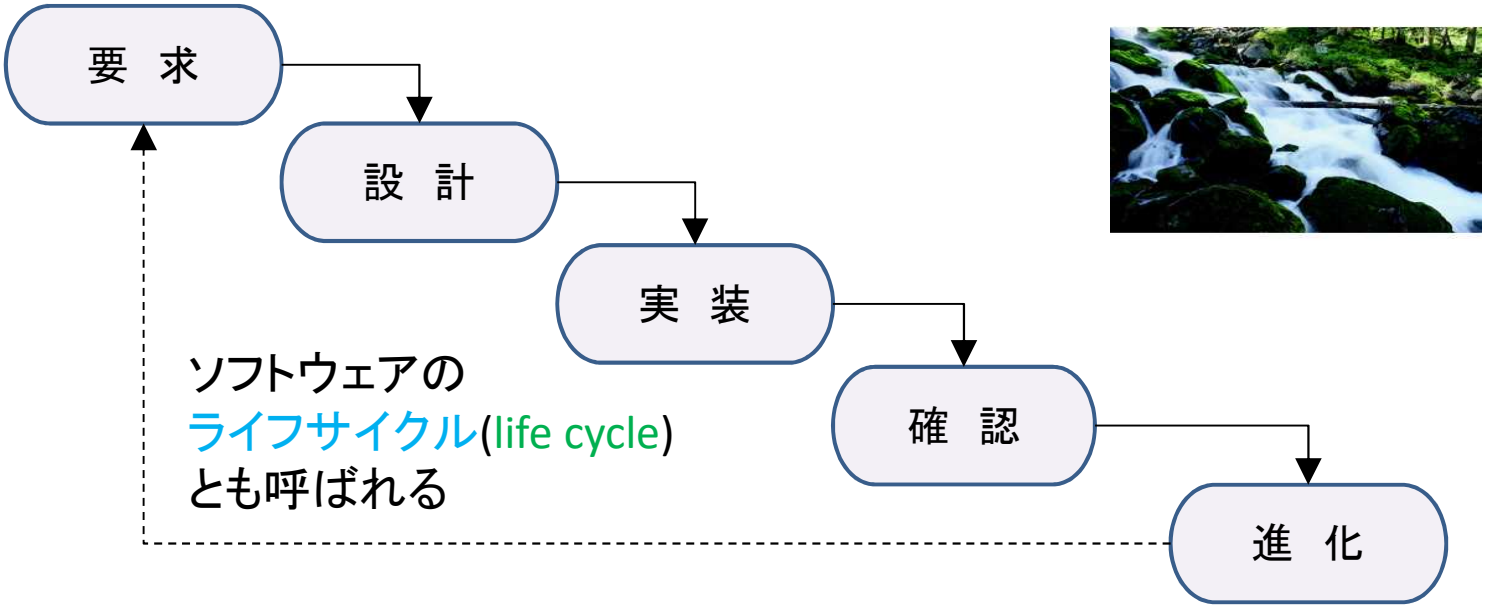
3) アジャイル開発(agile development)

バージョンアップ間隔が超短期の素早いインクリメンタル開発.

4) COTSインテグレーション(COTS integration)

入手可能なソフトウェアを集積(integrate)してシステムを構築.
システムを1から開発することはない.

ウォーターフォール開発(1/2)



- 要求→設計→実装→確認→進化の一連の活動を分離して実施
- 滝(waterfall)のように直列的にフェーズ(phase)がつながる
- 計画駆動(plan-driven):開発の開始前に全活動を計画・スケジューリング
- 原則としてフェーズの重なりや繰り返しを行わない

■長　所

1) フェーズが把握しやすく管理しやすい
2) 各フェーズの終了時にしっかりとしたドキュメントが生成される

ウォーターフォール開発(2/2)

■短　所

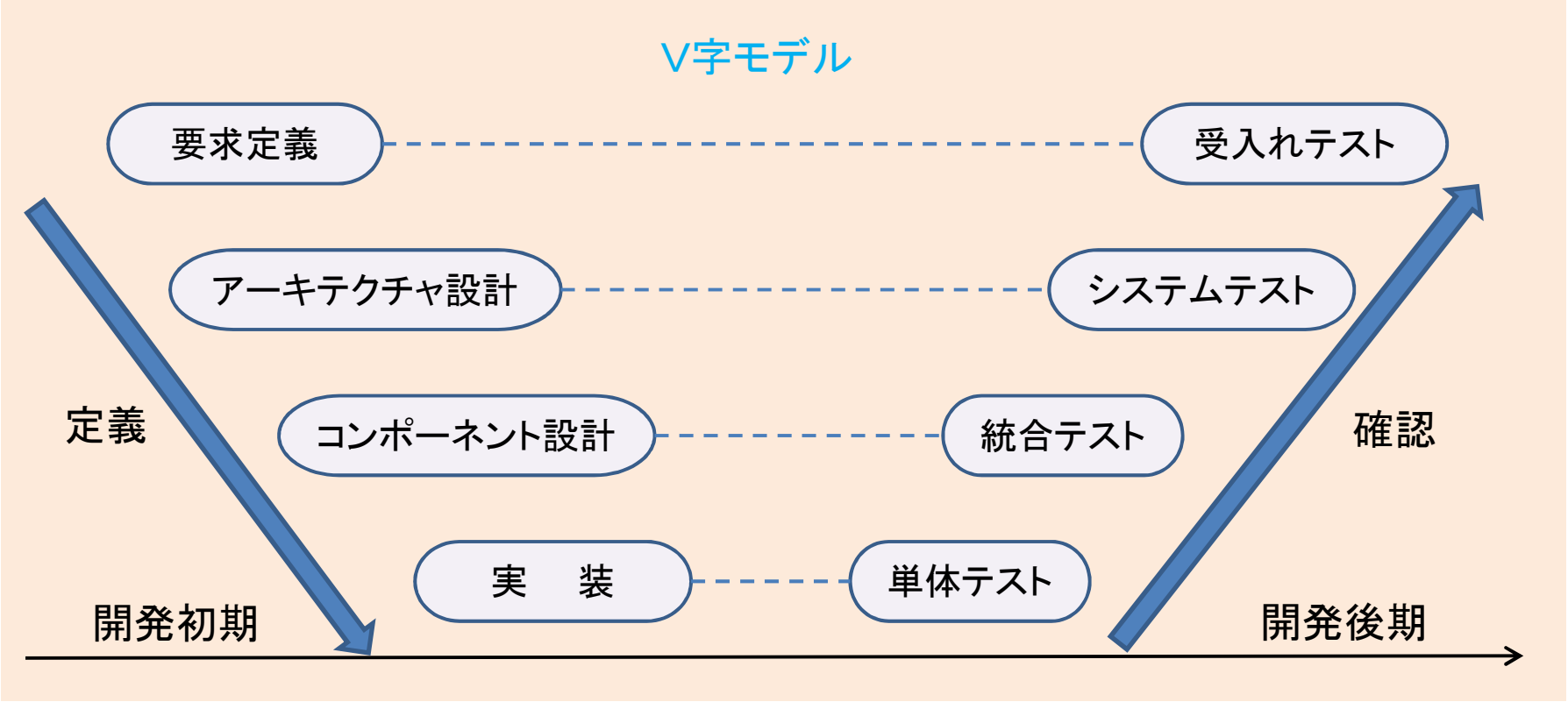
1) 要求の変化への対応が難しい

2) 開発上のリスクが大きい

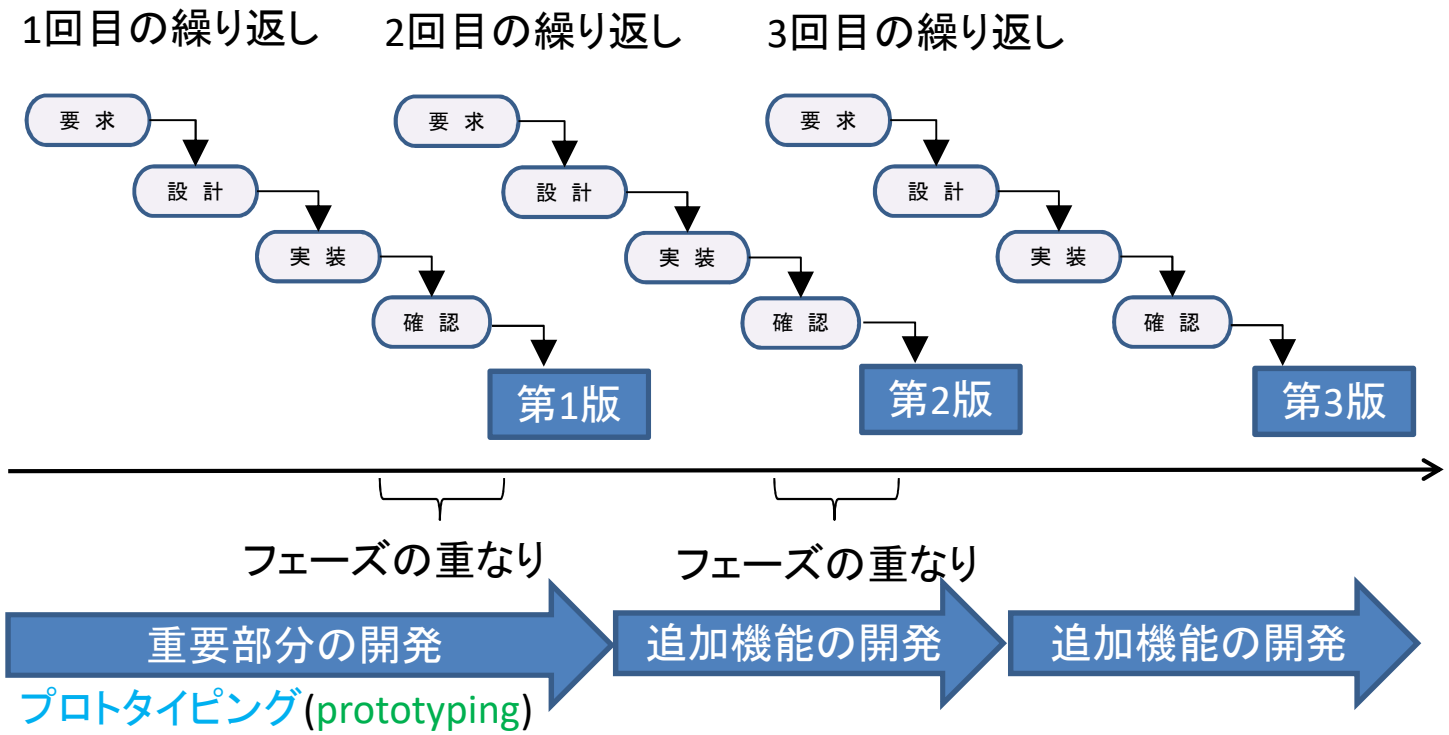
開発初期に定義したものほど，開発後期にならなければ確認できない

→ 大きな手戻りが生じ得る

→ 納期までに開発が終わらない／品質が悪い



インクリメンタル開発(1/2)



開発初期段階で顧客がシステムの重要部分の実装を確認できる

- 要求，設計，実装，確認をインターリーブ(interleave)
- 短い間隔で複数の版(version)を次々と開発し，顧客が迅速に確認
- 顧客からのフィードバックを得て次版のための増分(increments)を開発
- 最終版の確認が得られたら終了

インクリメンタル開発(2/2)

■長　所（ウォーターフォール開発との比較）
1) 要求の変化に対応するためのコストは小さい
　（変化の分析やドキュメントの量は，ウォーターフォール開発より小）
2) 開発の途中で顧客からのフィードバックを得られやすい
3) システムにすべての機能が実装されていない早い段階で運用可能

■短　所
1) ドキュメントの量が少ないので，管理者からプロセスが見えにくい．
　（すべての版を反映させた説明文書を作るのはコストが大）
2) 新しい増分を追加するにしたがって，システム構造が劣化しがち．
　（要求の変化に対応するのがだんだん困難となる）
3) 大規模・複雑で長寿命のシステムでは上記の短所が顕著

アジャイル開発(1/8)

■計画駆動(plan-driven)手法の特徴
- 1980年代～1990年代前半の一般的な「重い」手法
- 大規模・長寿命のソフトウェアシステム向き（航空宇宙，政府関係など）
  →多地域の異なる複数の企業からなる多人数のチームで開発
  →初期の要求仕様から配備まで，長期の開発期間を要するシステム
- 短所：計画・設計・文書化のオーバーヘッドが大きい
- ○オーバーヘッドが正当化される場合：
  →複数のチームが協力して開発をおこなうとき
  →システムが生命や財産を守るクリティカル(critical)システムのとき
  →ライフタイムにわたる長期間の保守に多くの人が関わるとき
- ●オーバーヘッドが正当化されない場合：
  →中小規模のソフトに適用されるとオーバーヘッドが大
  →プログラミングやテストよりも，計画や文書化に多大な時間を使用
  →要求の変化により仕様・設計・実装を変化させるとき

1990年代後半

agile:「素早い」という意味

アジャイル(agile)手法の提案
- 設計や文書化よりもソフトウェア自身の開発に注力
- 非常に短い間隔（2～3週間）でインクリメンタル開発を行う非常に「軽い」手法

アジャイル開発(2/8)

■アジャイル開発の哲学

プロセスとツール　よりも　個人と対話　を重視

包括的な文書化　よりも　動くソフトウェア　を重視

契約交渉　よりも　顧客との協働　を重視

計画への追従　よりも　変化への対応　を重視

■アジャイル開発の具体的な手法

- エクストリームプログラミング(extreme programming)
- スクラム開発(Scrum)

} 後に説明

アジャイル開発(3/8)

■アジャイル開発の原理

原　理	説　明
顧客との協働	- 顧客は開発プロセス全体に密接に関与 - 顧客の役割：システム要求の提供と優先度の提示，増分の評価
増分の提供	- 一連の増分によりシステム開発 - 次回の増分の要求仕様は顧客が指示
人間重視	- 開発チームのスキルを把握・尊重 - チームメンバーは自分なりの方法で開発
変化の許容	- 要求の変化があるのは当たり前 - 要求の変化に対応しやすくシステムを設計
単純性の維持	- ソフトウェアとソフトウェアプロセスの単純さを重視 - 可能な限り，システムから積極的に複雑性を排除

アジャイル開発(4/8)

■原理を実践する上での問題点

原　理		問題点
顧客との協働	-	顧客は，必ずしも開発チームと一体に作業を行う時間がない
増分の提供	-	大企業等では，長年続けてきたプロセスを簡単には変えられない
人間重視	-	チームメンバーは，他のメンバーと必ずしもうまく対話できない
変化の許容	-	多くの関係者間で変化に関する合意が困難
単純性の維持	-	単純性の維持には，余分な作業が必要（納期のプレシャー）

■その他の問題点

- インクリメンタル開発に対する契約書を書くのが困難
- 問題が生じたときはだれの責任か
- システムの進化（保守）にうまく対応できるか
  →ドキュメントが少ない
  →チームはすでに解散
  →顧客の協働意欲は小

アジャイル開発(5/8)

■エクストリームプログラミング(XP: extreme programming)

原理／実践	説　明
インクリメンタルな計画 Incremental planning	要求はストーリーカードに書かれ，開発時間と優先度に基づき，開発するストーリーを次期リリース計画に含める
短期リリース Small releases	ビジネス価値をもつ最小機能を初期リリースに含め，その後は2週間に1回程度インクリメンタルに機能を追加
シンプル設計 Simple design	要求を満たすのに十分な設計のみ実施（将来の変化に対応するための設計は無し．実際に変化したときに設計）
テスト第一主義 Test-first development	新機能の開発前にテスト設計．ツールを用いて単体テストのプログラムを作り，実装後に自動テストを実施
リファクタリング Refactoring	単純性維持のため，可能なら常に設計のリファクタリング（クラスの再設計など）を実施

アジャイル開発(6/8)

■エクストリームプログラミング(続き)

原理／実践	説　明
ペアプログラミング Pair programming	２人１組で開発．１人はコード作成，もう１人は妥当性やシンプルさのチェックと技術支援
共同所有 Collective ownership	コードは全員が所有し，責任をもつ．誰もが任意のコードを修正可
継続的インテグレーション Continuous integration	修正後にすぐ全体システムをインテグレートし，過去のすべての単体テストを実施
サステナブルペース Sustainable pace	コードの品質を保ち，生産性を高めるため，労働時間を適正に保つ
オンサイトカスタマー On-site customer	開発チームに顧客の代表をフルタイムで入れる

アジャイル開発(7/8)

■スクラム開発(Scrum):アジャイル開発のための管理手法

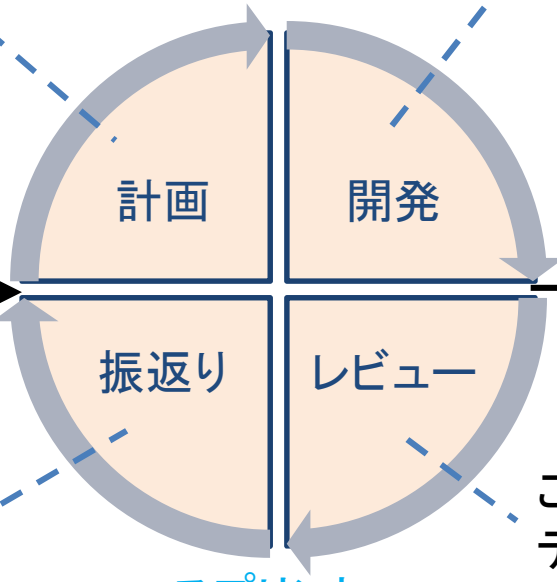
バックログ(backlog):今後行うべき仕事のリスト

バックログの内容を確認し,
今回のスプリントで開発すべき
機能をバックログから選定

毎日15分程度のミーティングで
進捗確認(昨日やったこと,今日
やること,障害)しつつ増分を開発

優先度や開発工数
を考慮して

概要計画
アーキテクチャ設計



プロジェクト終了
ドキュメントの作成

達成度,発生した問題,
その問題に対する改善策
などについて話し合う

このスプリントの開発成果を
デモなどで確認

スプリント
(sprint)

チーム主体の意思決定
(3〜10人=1チーム)

2〜4週間で1サイクルして
増分を開発

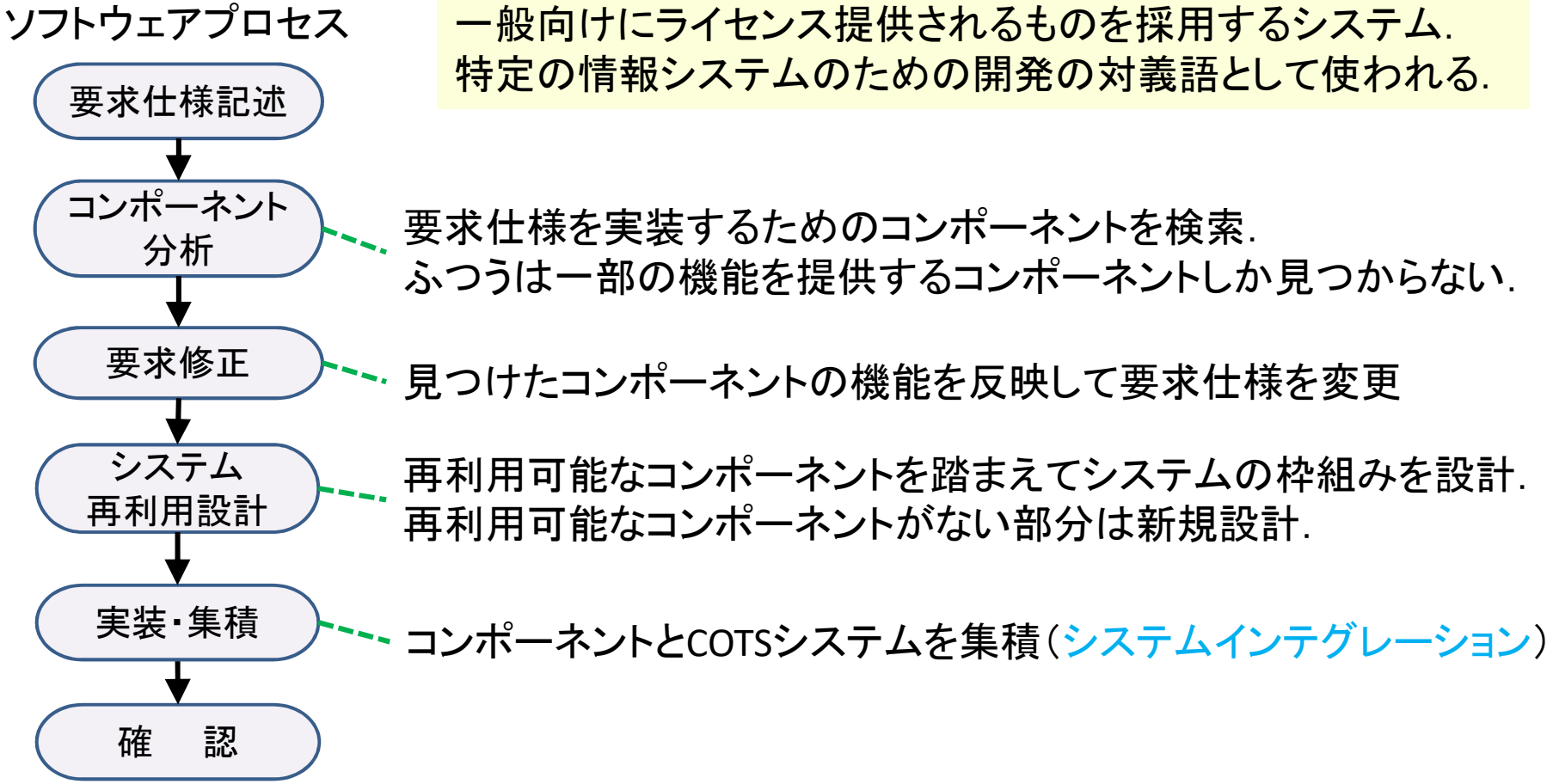
アジャイル開発(8/8)

■スクラム開発の長所
- ソフトウェアの機能を理解しやすい断片に分解可能
- 優先度と工数見積もりにしたがって項目を選択
- 顧客は定期的に増分を入手し，フィードバック可能
- チームメンバー間のコミュニケーションが促進
- 顧客と開発者の信頼関係が構築される

COTSインテグレーション(1/2)

再利用可能なCOTSシステムを集積(インテグレート)し,
なるべく独自開発をせずにシステムを構築

商用オフザシェルフ(COTS: Commercial Off-The-Shelf):
販売やリースをされている既製品のソフトウェア製品または
一般向けにライセンス提供されるものを採用するシステム.
特定の情報システムのための開発の対義語として使われる.



COTSインテグレーション(2/2)

■システムの例
- Webサービスシステム
  →標準的なプロトコルによる標準的なサービス
- 統合型業務ソフトウェア (ERP: Enterprise Resource Management)システム
  →製造計画，注文管理，顧客管理など，企業の典型的な業務を支援
- 大学向け業務ソフトウェア
  →履修管理，成績管理など，大学の典型的な業務を支援

■長所
- システム開発のコストとリスクの削減
- 開発期間の短縮

■短所
- システム要求の妥協が必要
- COTSソフトウェアの新規バージョンは使用者の制御下にない
  →予期しない仕様変更がある場合も
- サードパーティーの製品への依存関係の問題

演習問題 13

ウォーターフォール開発とアジャイル開発を比較し，
それぞれの長所と短所を簡単に説明しなさい．

レポート提出と筆記試験について

（詳しくは，ガイダンスで配布した文書を確認のこと）

2018年2月2日(金)　14:45集合（A21)

■レポート

- 演習問題を8問以上解答
- 欠席した回の分は解答できない

■筆記試験

- 手書きで直接書かれたノートのみ持込み可