Software Design
12. Software Processes

Natsuko Noda
nnoda@shibaura-it.ac.jp

ソフトウェア設計論
12. 開発プロセス

野田 夏子
nnoda@shibaura-it.ac.jp

Today's Topics

- Software development process
  - Waterfall model
  - Prototyping
  - Agile development

本日のお題

- ソフトウェア開発プロセス
  - ウォーターフォールモデル
  - プロトタイピング
  - アジャイル開発

Development Process models

開発プロセスモデル

Software Development process

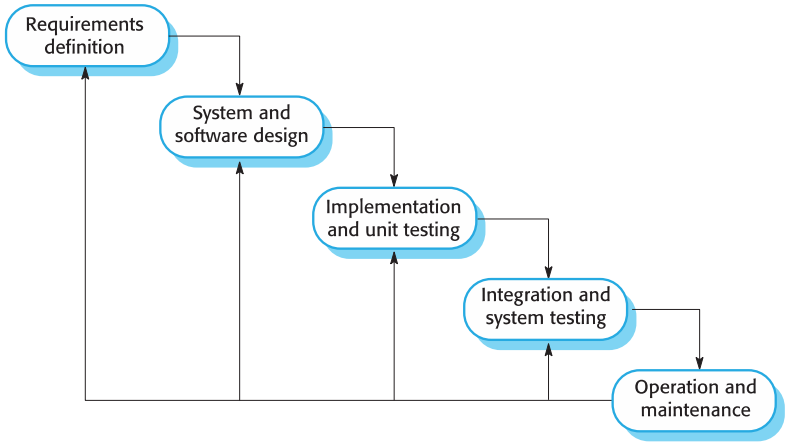
- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Requirements definition– defining what the system should do;
  - Design and implementation – defining the structure of the system and implementing the system;
  - Validation – checking that it meets the customer's wants;
  - (Evolution – changing the system in response to changing customer needs.)
- A development process model is a model that shows how the development process is thought to be related. It is represented by the top-level processes and their relations.

ソフトウェア開発プロセス

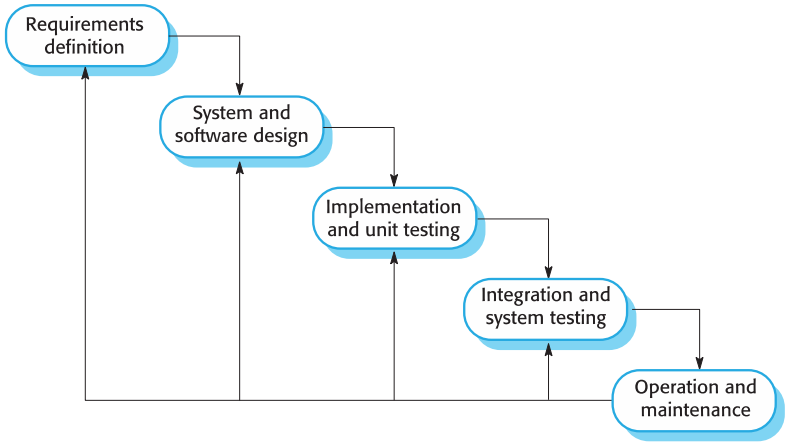
- ソフトウェアシステムを開発する際に必要となる関連した活動の体系的な集合
- 様々なプロセスがあるが大きな単位としては以下を含む：
  - 要求定義：システムが何をすべきであるかの定義
  - 設計・実装：システム構造を定義、システムを実装
  - 妥当性確認：ユーザの希望を満たすものかを確認
  - (進化：顧客のニーズの変化に対応してシステムを変更)
- 開発プロセスモデルは、開発プロセスがどのような考え方で関連づけられるかを示したモデル。最上位のプロセスとその関連づけによって表現される。

cf. SE-J, 8.1

The waterfall model



ウォーターフォールモデル



cf. SE-J, 8.2.1

Waterfall model phases

- There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- In principle, a phase has to be complete before moving onto the next phase.
  - Phases are done in sequence and, as a rule, do not overlap or repeat phases.

ウォーターフォールモデルのフェーズ

- ウォーターフォールモデルは、それぞれが分離された以下のようなフェーズから成る
  - 要求の分析と定義
  - システム設計・ソフトウェア設計
  - 実装と単体テスト
  - 統合テスト・システムテスト
  - 運用・保守
- 原則として、各フェーズは次のフェーズに移る前に完了しなければならない
  - フェーズを順序に沿って行い、原則的にフェーズの重なりや繰返しを行わない

Waterfall model problems

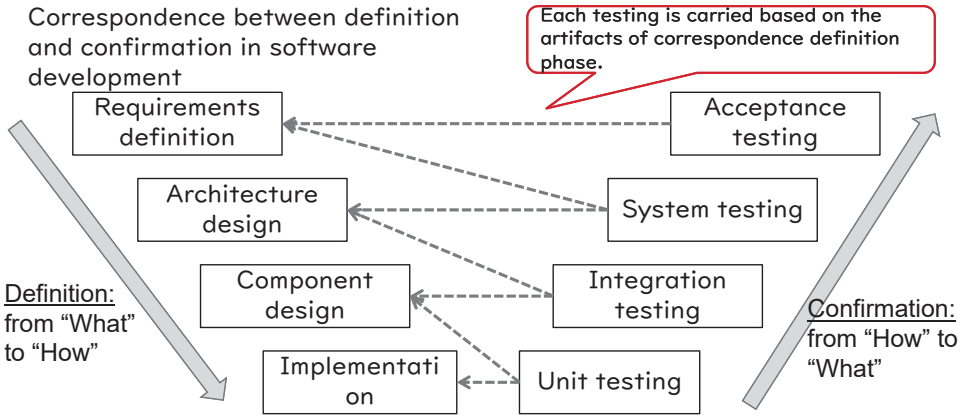
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway.
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - few business systems have stable requirements.

ウォーターフォールモデルの問題

- ウォーターフォールモデルの最大の欠点は、プロセスが進行した後の変化に対応することの難しさ
- プロジェクトを柔軟性のない個別のステージに分けてしまうと、顧客の要求の変化に対応することが難しくなる
  - したがって、このモデルは、要求が十分に理解されており、設計プロセス中の変更がかなり制限される場合にのみ適切
  - しかし、ビジネスシステムの中には、安定した要求を持つものはほとんどない！

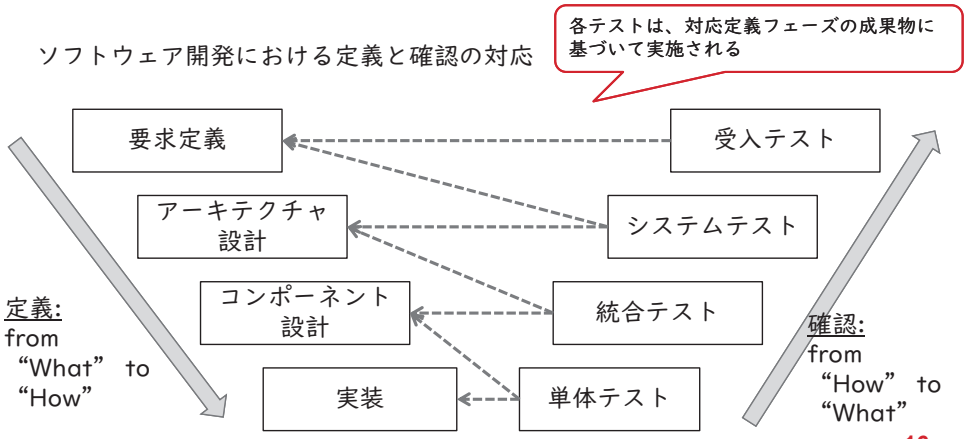
V-model

- A model in which each phase of the waterfall model is divided into a phase related to definition and a phase related to confirmation, and these phases are arranged in a V shape.



V字モデル

- ウォーターフォールモデルの各フェーズを定義に関わるフェーズと確認に関わるフェーズに分け、それをV字型に配置したモデル



Risk in Waterfall Model

- As the V-shaped model shows, what is defined in the earlier stages of development, cannot be confirmed until later in development.
- In the waterfall model, the sequence of phases is linear and not repetitive, so there is a high risk that if system and acceptance tests show that the requirements are not met, there will be no time left for response near the end of development

ウォーターフォールモデルのリスク

- V字モデルが示すように、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
- ウォーターフォールモデルでは、一連のフェーズを直線的に繰返しなく行うので、もしもシステムテストや受入れテストで要求に合致しないことが判明しても、開発終了間際で対応の時間が残されていない危険性が高い

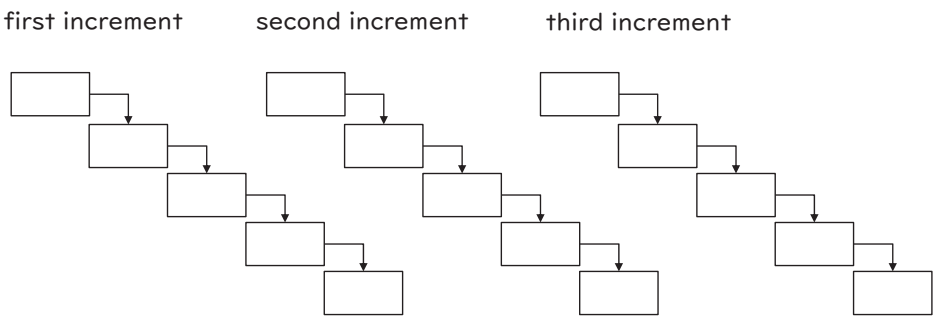
Waterfall model in practice

- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - in those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

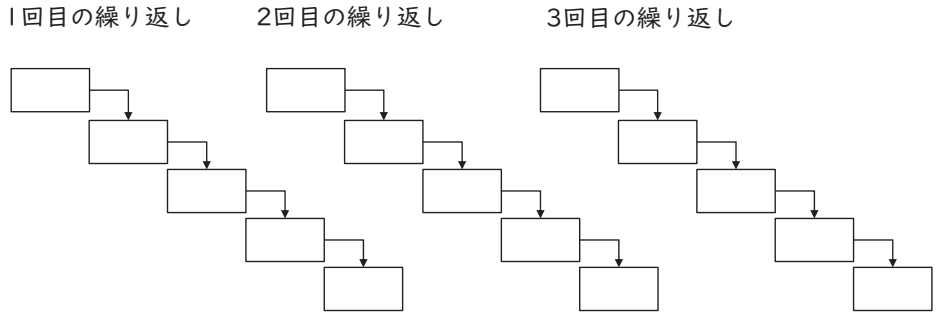
ウォーターフォールモデルの実際

- ウォーターフォールモデルは、システムが複数のサイトで開発される大規模なシステムエンジニアリングプロジェクトで主に使用される
  - このような状況では、ウォーターフォールモデルの計画主導的な性質が作業の調整に役立つ

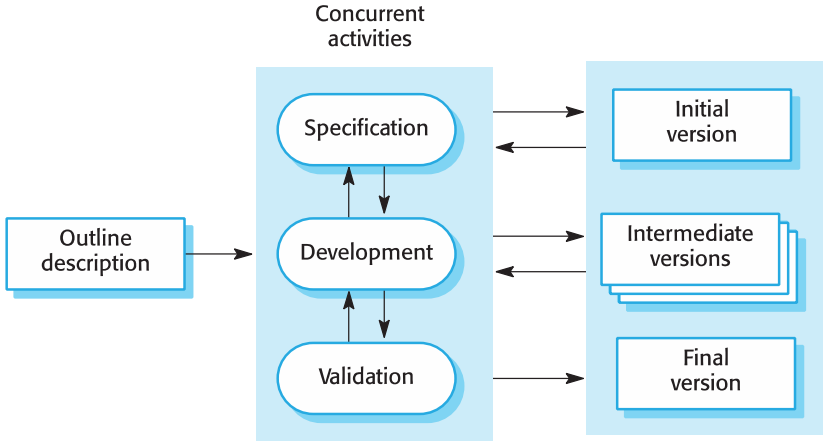
Incremental process model



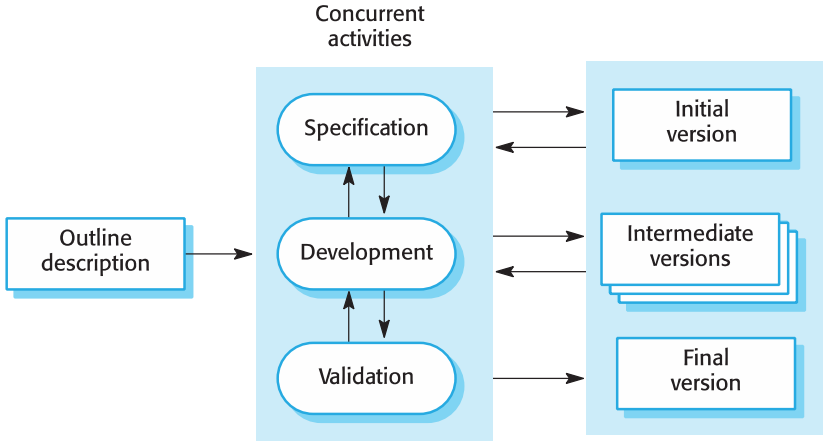
インクリメンタルプロセスモデル



Incremental process model



インクリメンタルプロセスモデル



cf. SE-J, 8.2.3

Incremental development benefits

- The cost of accommodating changing customer requirements is reduced.
  - the amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
  - customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - customers are able to use and gain value from the software earlier than would be possible with a waterfall process.

インクリメンタルな開発の利点

- 顧客要求の変化に対応するためのコストが削減される
  - 再度やり直す必要のある分析とドキュメントの量は、ウォーターフォールモデルで必要とされる量よりもはるかに少ない
- 開発作業に対する顧客のフィードバックを得やすい
  - 顧客はソフトウェアのデモにコメントできるし、どの程度の実装が行われたかを確認することができる
- 有用なソフトウェアをより迅速に顧客に届け、配置することが可能
  - 顧客は、ウォーターフォールプロセスで可能になるよりも早くソフトウェアを使用して価値を得ることができる

Incremental development problems

- The process is not visible.
  - managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

インクリメンタルな開発の問題

- プロセスが見えない
  - 管理者は、進捗状況を測定するために定期的な成果物を必要とする。システムの開発が早く行われている場合には、システムのすべてのバージョンを反映したドキュメントを作成するのは費用対効果が高くない
- 新たな繰り返しを追加すると、システム構造が劣化する傾向がある
  - ソフトウェアを改善するためのリファクタリングに時間と費用が費やされない限り、定期的な変更はその構造を破壊する傾向がある。ソフトウェアの変更をさらに取り入れることは、ますます困難になり、コストがかかる

Advanced Development Process Models

発展的な開発プロセスモデル

Coping with change

- Change is inevitable in all large software projects.
  - business changes lead to new and changed system requirements
  - new technologies open up new possibilities for improving implementations
  - changing platforms require application changes
- Change leads to rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality.
- Process models that anticipate and tolerate changes are needed.

変化への対応

- 大規模なソフトウェアプロジェクトでは、変更は避けられない.
  - ビジネスの変化に伴い、システム要求が変更されたり追加されたりする
  - 新技術により実装の改善が必要になる可能性がある
  - プラットフォームの変更により、アプリケーションの変更が必要になる
- 変更は手戻りにつながるので、変更の費用には、やり直し(例えば、要求の再分析)だけでなく新しい機能を実装するための費用も含まれることになる
- 変化を先取りして許容するプロセスモデルが必要

Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- A prototype can be used in:
  - the requirements engineering process to help with requirements elicitation and validation;
  - in design processes to explore options and develop a UI design.

プロトタイピング

- プロトタイプは、コンセプトを実証し、設計のオプションを試すために使用されるシステムの初期バージョン
- プロトタイプは以下で使われ得る:
  - 要求の抽出と検証を支援するための要求工学プロセスにおいて
  - 設計プロセスの中で、選択肢を探し、UI設計をするために

cf. SE-J, 8.2.4

Benefits of prototyping

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

プロトタイピングの利点

- ユーザビリティの改良
- ユーザの本当の要求に近づける
- 設計の質の改善
- 保守性の改良
- 開発コストの削減

Prototype development

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality
  - prototype should focus on areas of the product that are not well-understood;
  - error checking and recovery may not be included in the prototype;
  - focus on functional rather than non-functional requirements such as reliability and security

プロトタイプ開発

- ラピッドプロトタイピング用の言語やツールが必要になるかもしれない
- 機能を削減して開発
  - よく理解されていない点がある部分にフォーカスして作られるべき
  - エラー処理等は含まなくて良い
  - 信頼性やセキュリティ等の非機能要求よりも機能要求に注力して開発

Types of prototypes

- Throw-away prototypes
  - Prototypes are build only to confirm specific aspects of a system, and they are discarded after the confirmation.
- Evolutionary prototypes
  - Prototypes are build as a very robust ones in a structured manner and they are constantly refined.
  - They form the heart of the systems.

プロトタイプの種類

- 使い捨て型
  - プロトタイプは、システムの特定の側面を確認するためだけに構築され、確認後は破棄
- 進化型
  - プロトタイプは非常に堅牢なものとしてきちんと構造化して作られ、それが常に改良されていく
  - システムのコアになっていく

Boehm's spiral model

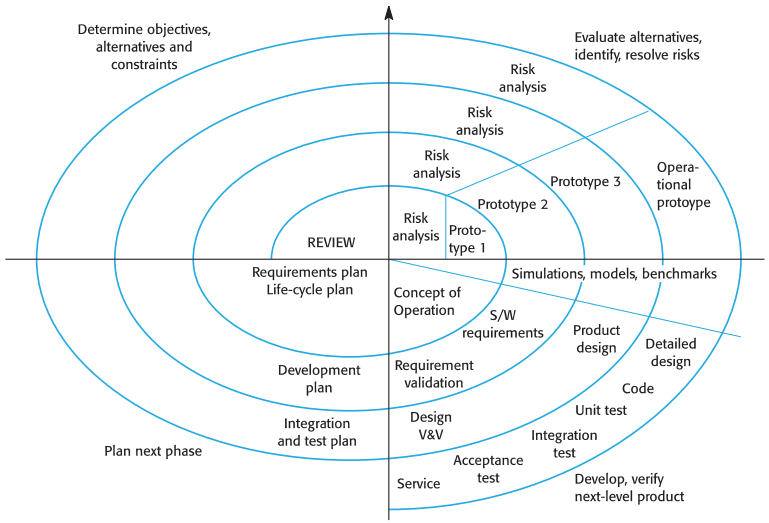
- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- Risks are explicitly assessed and resolved throughout the process.

ベームのスパイラルモデル

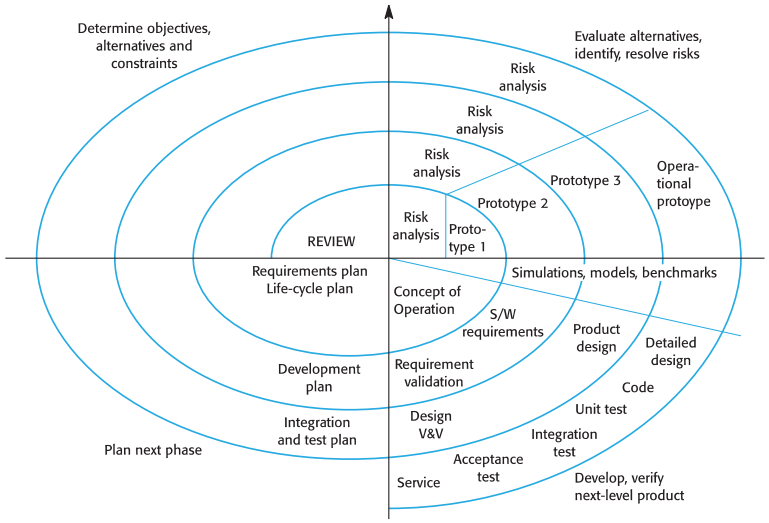
- プロセスは、バックトラックを伴う一連の活動としてではなく、スパイラルとして表現される
- スパイラルのそれぞれのループはプロセスにおけるフェーズに相当
- リスクをプロセス全体を通して明確に評価し解決

cf. SE-J, 8.2.4 (2)

Boehm's spiral model: Overview



スパイラルモデル：全体像



Agile Development

アジャイル開発

cf. SE-J, 8.2.5

Agile development

- Agile development is a general term for light development methods that aim to respond quickly to changes.
- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - focus on working software rather than documentation
  - are based on an iterative approach to software development
  - are intended to deliver working software quickly and evolve this software quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

アジャイル開発

- アジャイル開発は、変化に迅速に対応することを目指した軽い開発手法の総称
- 1980年代と1990年代のソフトウェア設計手法に関わるオーバーヘッドへの不満がアジャイルな手法の創造へとつながる。これらの方法は：
  - 文書よりも動くソフトウェアに焦点を当てる
  - ソフトウェア開発の反復的なアプローチに基づく
  - 動くソフトウェアを迅速に提供し、変化する要求を満たすためにこれを迅速に進化させることを目的とする
- アジャイル手法の目的は、ソフトウェア開発プロセスのオーバーヘッドを削減し（ドキュメントを制限するなど）、過度な手戻りをすることなく、変化する要求に迅速に対応できるようにすること

The principles of agile methods

Informative

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is to provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

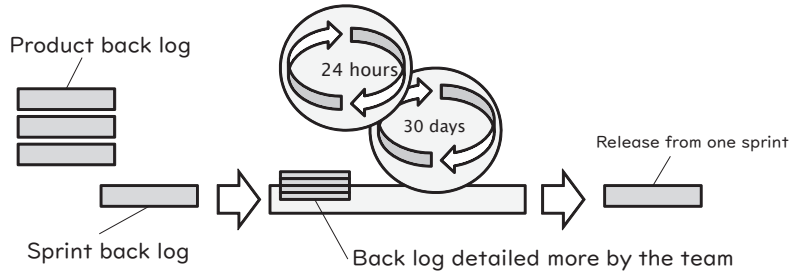
アジャイル開発の原理

参考

Principle	Description
顧客の参加	Customers should be closely involved throughout the development process. Their role is to provide and prioritize new system requirements and to evaluate the iterations of the system.
繰返して進化していくの成果物の提供	The software is developed in increments with the customer specifying the requirements to be included in each increment.
プロセスより人	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
変化の許容	Expect the system requirements to change and so design the system to accommodate these changes.
単純さの維持	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

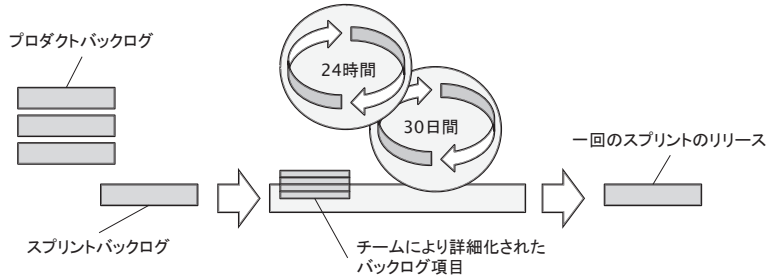
Scrum

- Three roles in the process
  - Product owner： understands product requirements and priorities from a business perspective
  - Scrum master： helps the development method to deliver the requirements in line with the priorities
  - Software development team：develops software in practice



スクラム

- 以下の3つの役割によりこの手法を実現
  - プロダクトオーナー：ビジネス的な観点からプロダクトへの要求や優先度を理解
  - スクラムマスター：要求を優先度に沿って実現するために手法を支援
  - ソフトウェア開発チーム：実際に開発



Extreme programming

- Perhaps the best-known and most widely used agile method.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - new versions may be built several times per day;
  - increments are delivered to customers every 2 weeks;
  - all tests must be run for every build and the build is only accepted if tests run successfully.

エクストリームプログラミング

- アジャイル開発でもっともよく知られ広く使われている手法
- エクストリームプログラミング(XP)は、反復型の開発において極端な(extremeな)アプローチを取る
  - 新しいバージョンは一日に数回ビルドされる
  - 2週間に1回のペースで顧客に前回からの増分が届けられる
  - すべてのテストはすべてのビルドに対して実行されなければならず、テストが正常に実行された場合にのみビルドが受け入れられる

Testing in XP

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- XP testing features:
  - test-first development.
  - incremental test development from scenarios.
  - user involvement in test development and validation.
  - automated test harnesses are used to run all component tests each time that a new release is built.

XPにおけるテスト

- テストはXPの中心であり、XPはすべての変更が行われた後にプログラムをテストするというアプローチを開発
- XPにおけるテストの特徴:
  - テストファーストな開発
  - シナリオを使ったインクリメンタルなテスト
  - テスト開発と妥当性確認へのユーザの参加
  - 自動化されたテストハーネス(テスト実行用のソフトウェア)は、新しいリリースがビルドされる度にすべてのコンポーネントテストを実行するために使用される

Test-first development

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - usually relies on a testing framework such as Junit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.
- Nowadays it is often separated from Agile development. This has been evolved to "test-driven development" (TDD).

テストファースト開発

- コーディングの前にテストを書くことで、実装すべき要求が明確になる
- テストは自動的に実行できるように、データではなくプログラムとして書かれる。テストには、それが正しく実行されたかどうかのチェックが含まれる
- Junitなどの支援ツールを使うことが一般的
- 新しい機能が追加されたときに、これまでのすべてのテストと新しいテストが自動的に実行され、新しい機能がエラーを発生させていないことをチェック
- 現在ではアジャイル開発と切り離されることが多く、「テスト駆動開発」（TDD）へと進化

Manifesto for Agile Software Development

- introduced by K. Beck et al. in 2001 as the statement of the main purpose of the Agile method.

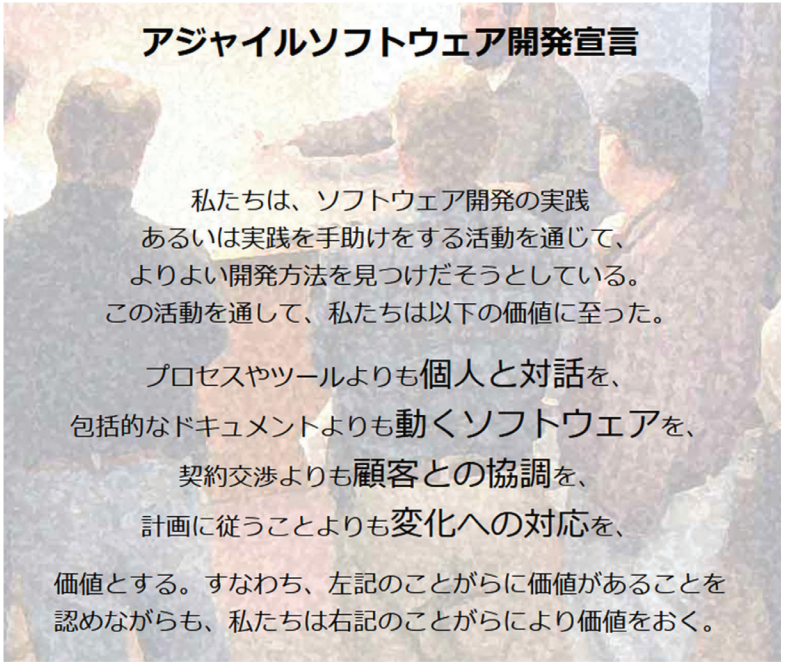
https://agilemanifesto.org/iso/en/manifesto.html

アジャイルソフトウェア開発宣言

- 2001年K. Beckらが、アジャイル開発の主旨を表すものとして表明

https://agilemanifesto.org/iso/ja/manifesto.html

https://agilemanifesto.org/iso/en/manifesto.html



https://agilemanifesto.org/iso/ja/manifesto.html

Principles behind the Agile Manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

https://agilemanifesto.org/iso/en/principles.html

アジャイル宣言の背後にある原則

- 顧客満足を最優先し、価値のあるソフトウェアを早く継続的に提供します。
- 要求の変更はたとえ開発の後期であっても歓迎します。変化を味方につけることによって、お客様の競争力を引き上げます。
- 動くソフトウェアを、2-3週間から2-3ヶ月というできるだけ短い時間間隔でリリースします。
- ビジネス側の人と開発者は、プロジェクトを通して日々一緒に働かなければなりません。
- 意欲に満ちた人々を集めてプロジェクトを構成します。環境と支援を与え仕事が無事終わるまで彼らを信頼します。
- 情報を伝えるもっとも効率的で効果的な方法はフェイス・トゥ・フェイスで話をすることです。
- 動くソフトウェアこそが進捗の最も重要な尺度です。
- アジャイル・プロセスは持続可能な開発を促進します。一定のペースを継続的に維持できるようにしなければなりません。
- 技術的卓越性と優れた設計に対する不断の注意が機敏さを高めます。
- シンプルさ（ムダなく作れる量を最大限にすること）が本質です。
- 最良のアーキテクチャ・要求・設計は、自己組織的なチームから生み出されます。
- チームがもっと効率を高めることができるかを定期的に振り返り、それに基づいて自分たちのやり方を最適に調整します。

https://agilemanifesto.org/iso/ja/principles.html

For your review

1. What is the waterfall model?
2. What are problems of the waterfall model?
3. What is a prototype?
4. What is incremental development?
5. What are the agile methods?
6. Read the principles behind the Agile Manifesto, and consider the importance of the software design in the Agile methods.

Extreme programming practices (a)

Informative

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent, and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices (b)

Informative

Famous approach in XP

Pair programming	Developers work in pairs, checking each other's work, and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

エクストリームプログラミングのプラクティス

- (参考情報なので、詳細は気にしなくて良い)
- (日本語でのまとめは、参考書「ソフトウェアエ学」のp.188、表8.1を参照。なお、前頁の英語と多少項目が異なるが、これは文献やバージョンで若干の違いがあるため)