ソフトウェア工学
6章　実装

発表者：M1 倉地亮介

目次

- 実装と実行環境
  1) 実装とは　2) 実装の課題　3) ソフトウェアと実行環境
- 実装に関わる技術
  1) ビルド作業と依存関係　2) バージョン管理　3)CIとOSSについて
- デバッグ
  1) デバッグとは　2) デバッグの課題　3) デバッグに関する技術
- まとめ

実装と実行環境

1. 実装とは

実装：設計に基づき実行可能なソフトウェアを構築する作業である

インターネットでは

・何らかの機能や仕様を実現するための装備，またはその方法
・システム開発における「プログラミング」の工程のこと　etc…

ソフトウェアの分野では，仕様やアルゴリズムをプログラミング言語で記述すること

つまり，プログラミングすること

実装とは

近年の計算機技術の進化，多様化

ソフトウェアの実行環境も様々になった

- ソフトウェアを実行するためには，プログラムだけでは無理
- 実行環境を設定するための情報が必要

実装とは，実行可能なソフトウェアを構築するとともに，
それが想定する実行環境を設定する作業ともいえる

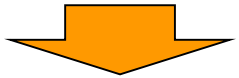
実装の課題(1)

- プログラムの作成
  - ー実行可能なソフトウェアの定義には様々な知識，技術，熟練が必要
- 機械的な作業の繰り返し
  - ーソフトウェアを作る際には，多数の成果物のコンパイルやリンクが必要
- 依存関係の管理
  - ー成果物の複雑な依存関係を正しく管理する必要がある

実装課題(2)

- バージョンの再現
  - バージョンの異なるソフトウェアを作る際，成果物や必要な情報をバージョンごとに管理する必要がある
- 不具合の再現や原因の特定
  - ソフトウェアを作っても正しく動作しない場合には，その原因を見つける必要がある

ソフトウェアや実行環境の構成要素は，人間が定義した機能的な人工物



これらを利用するには，その仕様をしっかりと理解する

ソフトウェアと実行環境(1)

アプリケーションフレームワークの利用

・様々なアプリケーションが共通に利用できる骨格をあらかじめ用意しておくこと

メリット ： ・速く作れる
・コードの書き方を統一できる

デメリット ： ・フレームワーク自体を覚えるのに時間・労力がかかる
・仕組みを理解しなくても書けてしまう

定義部分は必ずしも，プログラミング言語で記述されるとは限らない

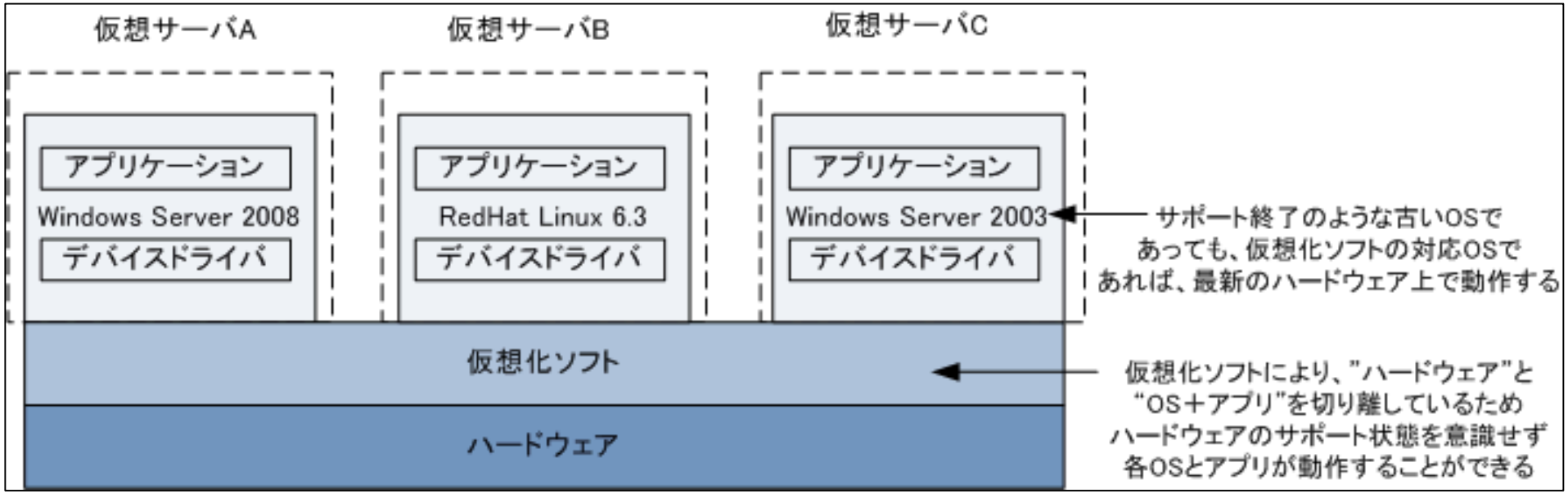
－ e.g. XML，HTML，JSP

ソフトウェアと実行環境(2)

仮想化環境・クラウド

・仮想化：リソースを，それを利用する側に対して隠蔽する技法

仮想化環境を使うことで
- 物理サーバ台数の削減
- ハードウェアリソースの効率利用
- サーバ環境のカプセル化



ソフトウェアと実行環境(2)

仮想化環境・クラウド

・仮想化：リソースを，それを利用する側に対して隠蔽する技法

仮想化環境を使うことで
- 物理サーバ台数の削減
- ハードウェアリソースの効率利用
- サーバ環境のカプセル化

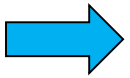
・バーチャルマシンの実現方法

実装に関わる技術

ビルド作業と依存関係(1)

ビルドとは
・実行可能なソフトウェアの定義を，実行可能なソフトウェアへと変更する作業のこと
・コンパイル＋α

ビルド作業
・コンパイル＋ライブラリのリンクだけ？
・テスト実行までも含める？



作業範囲の厳密な境界はない

■ **ビルド作業の自動化**
・ビルド作業を効率化し，人為的な間違えをなくすため

ビルド作業の自動化ツール
・作業手順をあらかじめ定義しておき，それに基づいてツールを適用することで，
実行可能なソフトウェアを生成する

Make，Ant，Scons　など

ビルド作業と依存関係(2)

依存関係の管理

- ビルド作業に関わる成果物には様々な依存関係がある
  - 依存関係：依存先の成果物が修正された場合に依存元の成果物に関わるビルド作業を再度行う必要があるという関係

- インクリメンタルなコンパイル
  - 修正に応じて最小限の再コンパイル
  - 依存関係が適切に管理できていないと無理

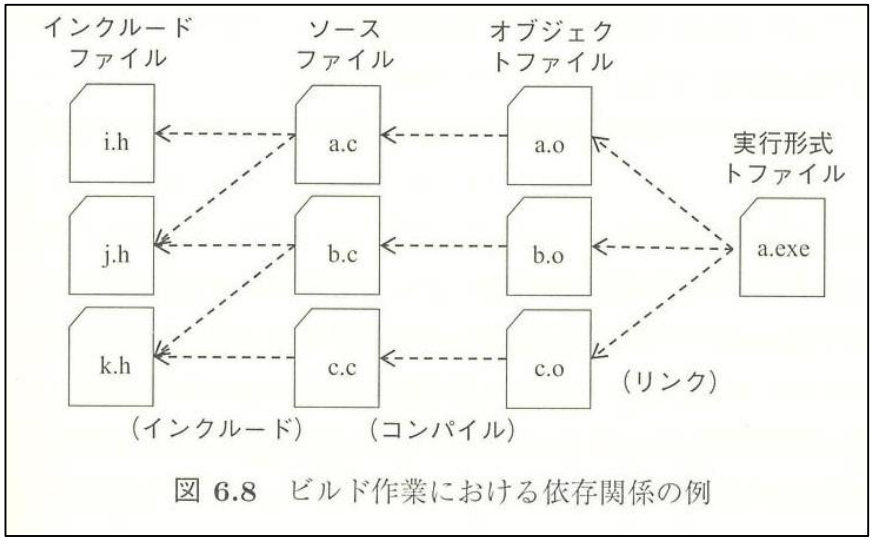


図 6.8　ビルド作業における依存関係の例

依存関係を抽出し，修正に応じて最小限の再コンパイルする作業を自動化することが大切

バージョン管理(1)

■ バージョン：実行可能なソフトウェアをリリースあるいは，再リリースすること

- リビジョン：不具合など相対的に小さなリリース
  ーバージョンとほぼ同じ

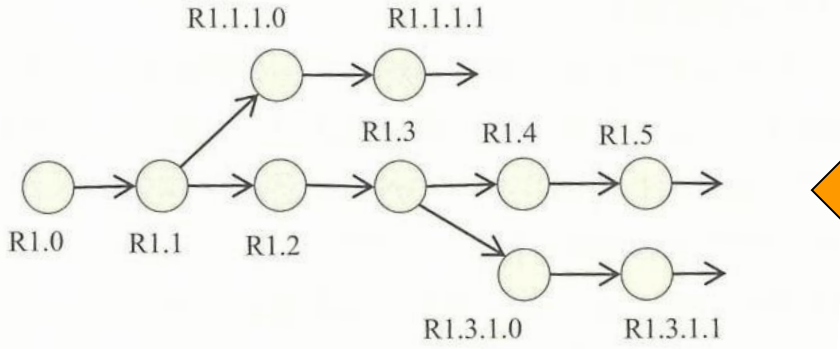
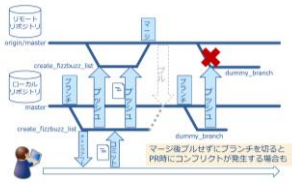


図 6.9　バージョン木の例

・バージョンを時系列的に順序づけれる
・バージョンは分岐することもある

バージョン管理の目的
バージョンを再現するために必要な成果物や情報を記録し，必要に応じて再現すること



バージョン管理(2)

コンフリクト

同じ個所を別々のブランチで,別々の変更をかけてしまっているのにマージすると発生

- ロック方式

チェックアウト(pull)されるとリポジトリ中の成果物をロック

・コンフリクトによるトラブルを回避

・ある人の修正作業中が長い時その間，他の人が成果物に対する作業を一切できない

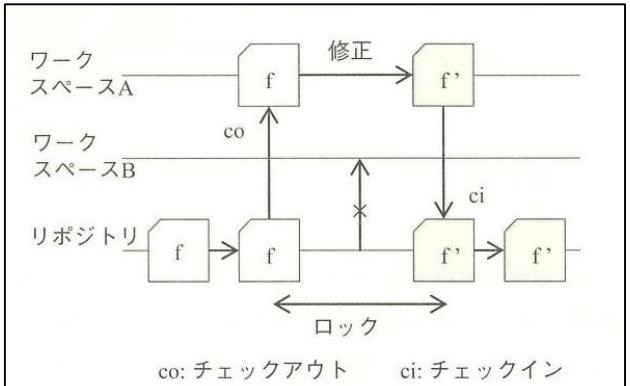


図 6.11　ロック方式

- コピーマージ方式

複数の作業者が同時にチェックアウトして作業することを認める方式

・同じ個所を異なった修正を行った場合⇒警告

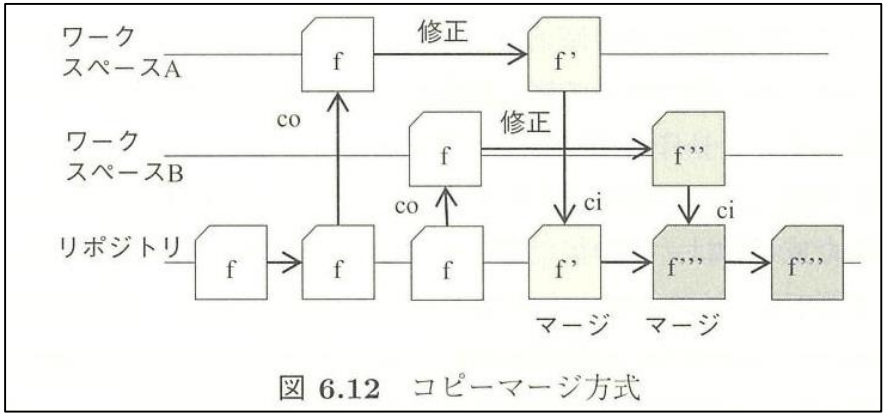


図 6.12　コピーマージ方式

CIとOSSについて

- 継続的インテグレーション(CI)
- プログラムを作成し，ビルド作業を行い配置してテストをするという作業を繰り返し頻繁に行う方法

  - 問題の早期発見，解決できる
  - 頻繁に行うので自動化(CIツール)

- オープンソースソフトウェア(OSS)
- 誰でも自由に利用できるという条件で公開されたソフトウェア

特徴：
- ボランティアでの参加が基本 ⇒ 安価あるいは無償
- 開発者が自由に参加/離脱することが可能 ⇒ 参加/離脱の流動性が激しい
- 緊急時の技術的なサポートが得にくいこと ⇒　利用者は信頼性が不安

デバッグ

デバッグとは

- プログラム中の欠陥を検出し，その原因を特定し，それを修正する作業

デバッグの方法

1. 不具合の発見： 期待した動作をしない
2. 不具合の再現： 再現できない不具合の解明は困難
3. 原因の特定 ： 不具合はあくまで現象，その現象だけを回避してもダメ．真の原因を特定する
4. 原因の解決 ： 修正方法を検討
5. 確認 ： 不具合が解決されたかだけでなく，他で副作用が出てないかも確認

デバッグの課題

デバッグは特有の難しさを持っている

- 不具合を見つけることの難しさ
  ー そもそも欠陥の不在を示すことは不可能
- 不具合の再現の難しさ
  ー 1度は観測された不具合を再現できないことも多い
- 原因特定の難しさ
  ー 不具合を再現できても，なぜ起こるか分からない
- 修正方法が自明とは限らない
  ー 修正によって他の機能に影響がでるかもしれない

デバッグの技法

どこにどのような欠陥が潜んでいるか分からないので，思いつくままにデバッグしてもうまくいかない，基本的な技法や方針に従うべき

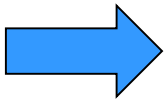
- 状況の的確な把握
  ーバグが見つかった時、when,where,howを記録する
- 観測の容易化
  ー必要な情報を観測しやすくする
- 問題の簡略化
  ー不具合が起こる状態をできるだけシンプルにする
- 原因の追究
  ー不具合を見つけたところからさかのぼって原因を追究する

デバッグに関する技術

- プリントと表明
  - プリント命令をプログラムの適当な箇所に挿入してデータの値などを表示してチェックする
- デバッガ
  - ブレークポイントでプログラムを中断させ，変数の値やメモリの状況をチェックする
- バグ管理システム
  - 発見された不具合を登録し，修正される状況を追跡する

まとめ

- 実装と実行環境
  - ー実装とは，実行可能なソフトウェアを構築するとともに，
    それが想定する実行環境を設定する作業である
- 実装に関わる技術
  - ビルド
  - バージョン管理
  - 継続的インテグレーション



自動化が重要

- デバッグ
  - ーデバッグ作業は難しく多くの課題がある
  - ーデバッグに関する技術