Tools and Behavioral Abstraction: A Direction for Software Engineering （K. Rustan M. Leino　著）

南山大学大学院　数理情報研究科数理情報専攻

青山研究室所属

M2012MM002

朝倉　知也

シナリオ

1.はじめに

2.Composing programs

3.振舞いの抽象化のための開発環境

4.挑戦

5.終わりに

6.参考文献

1. はじめに

- ソフトウェア工学(Software Engineering):1968年に初めて公式に使用
  - 数十年で大きな進歩を遂げる
    - ✓ 構造化プログラミングの台頭
    - ✓ 様々なテスト方法の開発（単体試験，システム試験，ホワイトボックス，等）
  - 現在では様々な開発支援ツールが提供されているが，未だ問題が残る
    - ✓ ユーザの要求と開発者の想定の差
    - ✓ ソフトウェアの使いやすさ
    - ✓ 故意または偶然の誤用に対する対策
    - ✓ ソフトウェア欠陥の減少
    - ✓ 機能追加や修正の容易さ
  - ソフトウェア工学は今後，これらの問題を解決することが期待される

2. Composing programs

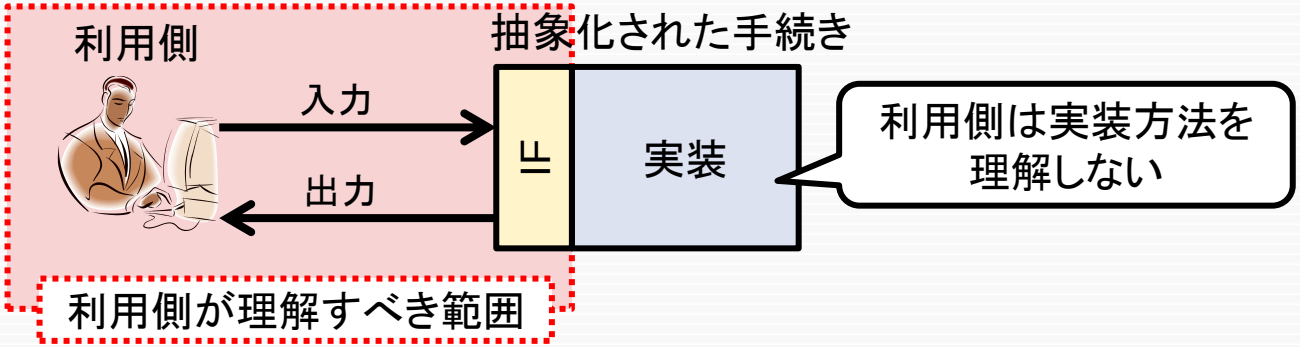
- “Composing programs”
  - プログラム要素を組み合わせて全体のプログラムを構成
  - 新しいコードを記述する際に行うこと
    - ✓ 新たなデータ構造やアルゴリズムの設計
    - ✓ 既存または新たなプログラムと結合
    - ✓ ライブラリ呼び出し，データ構造の再利用

    ⇒プログラムの複雑化

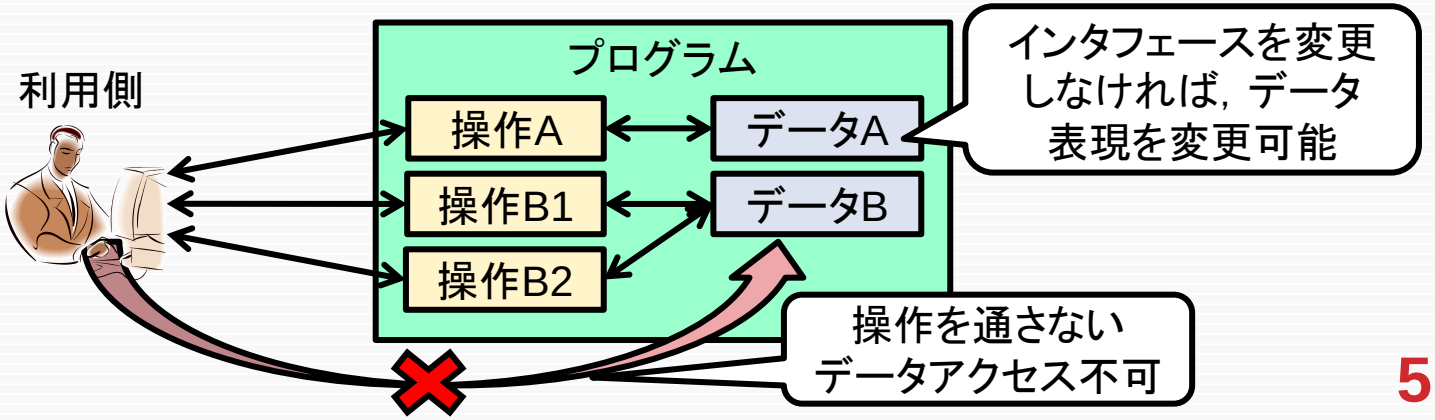
複雑化を防ぐツール：抽象化

2. Composing programs

- 一般的な抽象化
  - 手続きの抽象化(Procedural abstraction)
    - ✓ 利用方法(インタフェース)と実装を分離

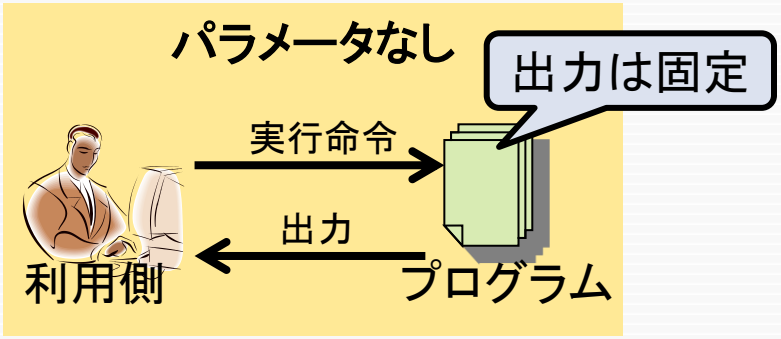


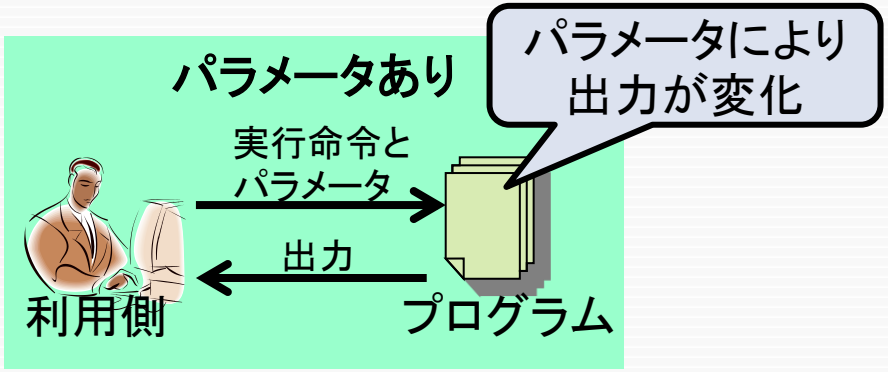
  - データの抽象化(Data abstraction)
    - ✓ データに対する操作と実際のデータ表現を分離



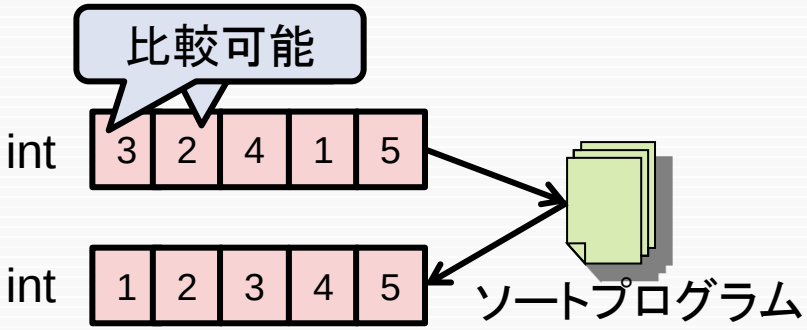
2. Composing programs

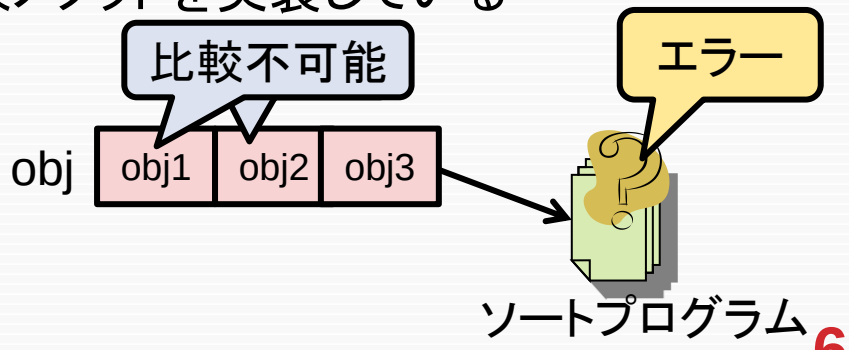
- 抽象化のパラメータと制限
  - ひとつの利用方法に依存しない必要
    ⇒パラメータ(引数)により設定





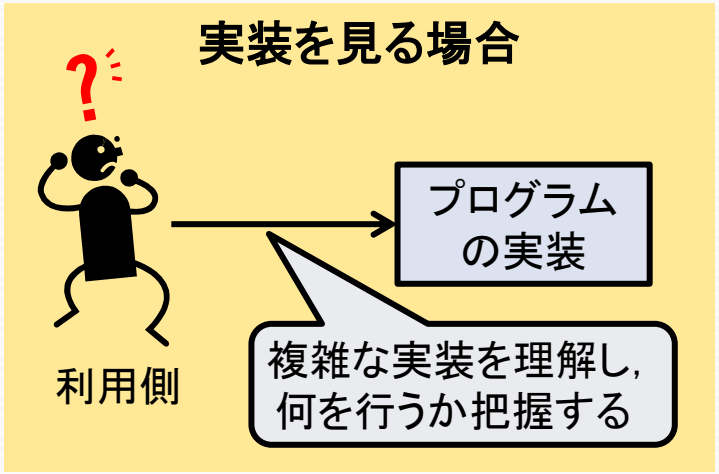
  - パラメータの制限
    例）ソートプログラム
    ⇒制限：対象のデータ型が比較メソッドを実装している

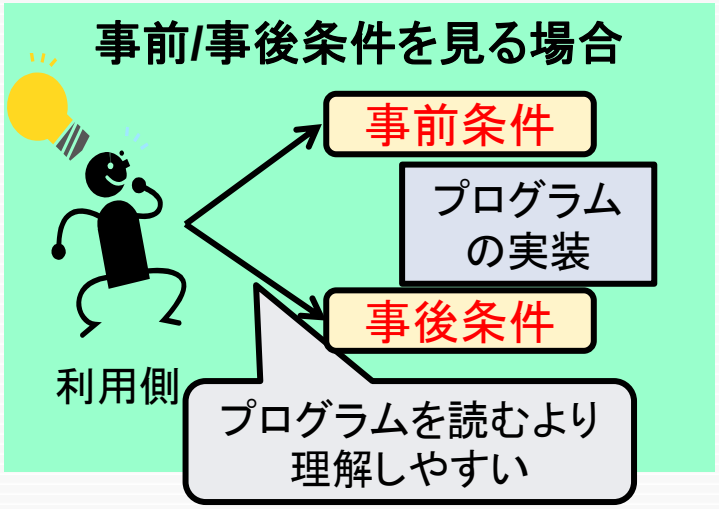




2. Composing programs

- 振舞いの抽象化
  - プログラムが「何をするか」を説明
    - ✓ 事前条件(precondition)と事後条件(postcondition)の明確化

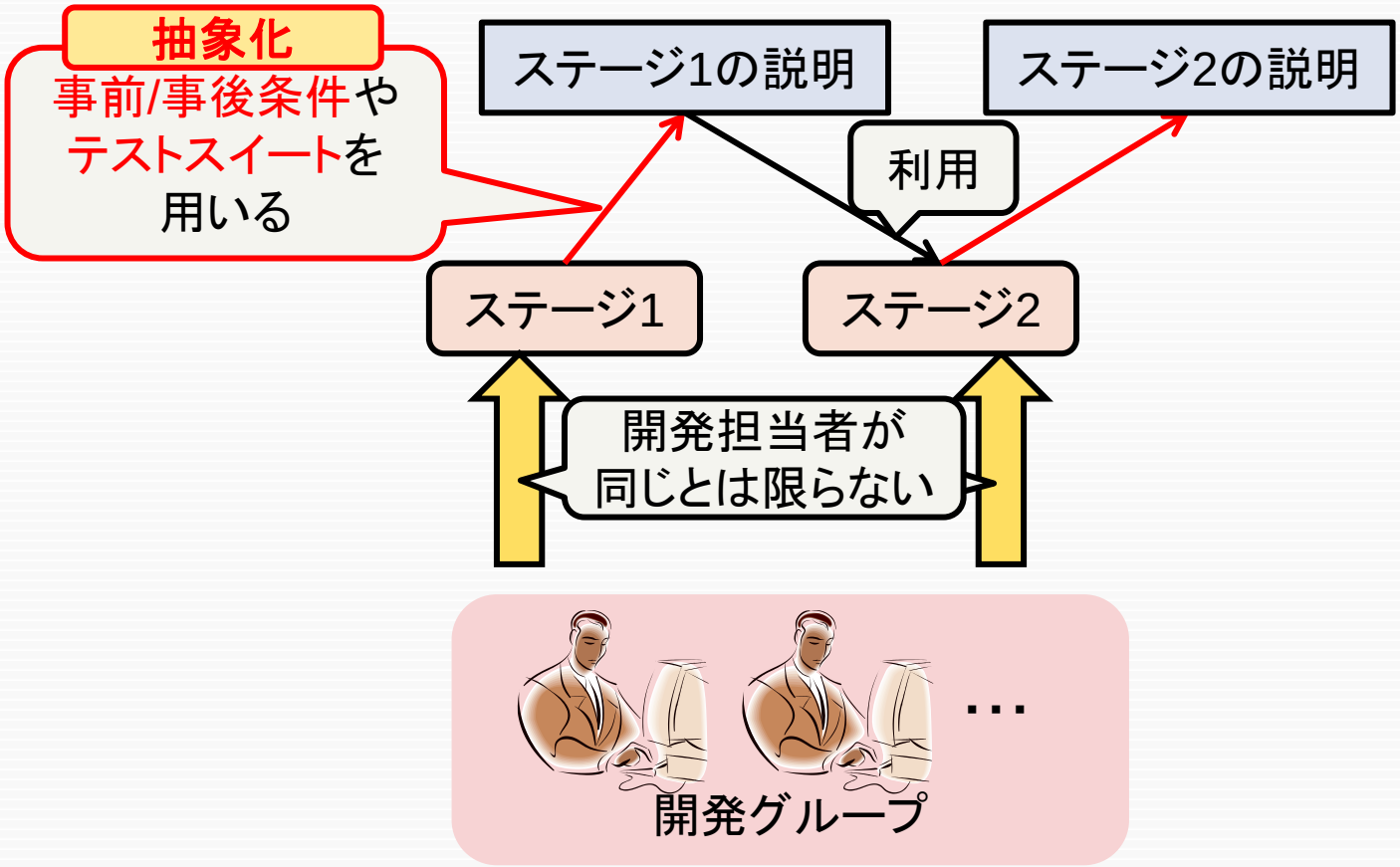




  - テストスイート(複数のテストケースをまとめたもの)
    - ✓ ひとつのユースケースセット
    - ✓ 事前/事後条件とテストスイートを合わせ，プログラムを説明

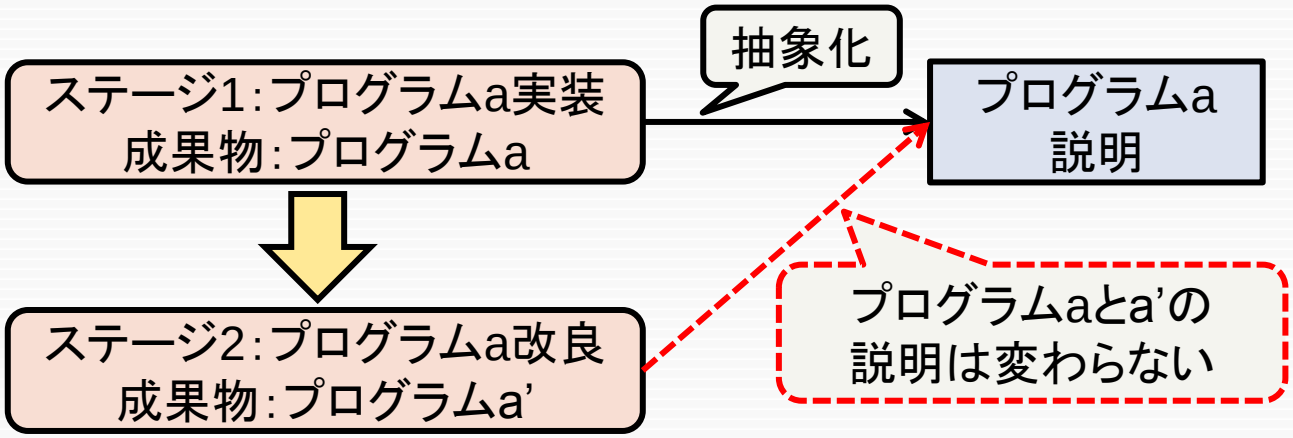
3. 振舞いの抽象化のための開発環境

- 振舞いの抽象化のために開発環境が行うべきこと
  - 複数のステージの説明
    - ✓ 前ステージの振舞いを抽象化し利用



3. 振舞いの抽象化のための開発環境

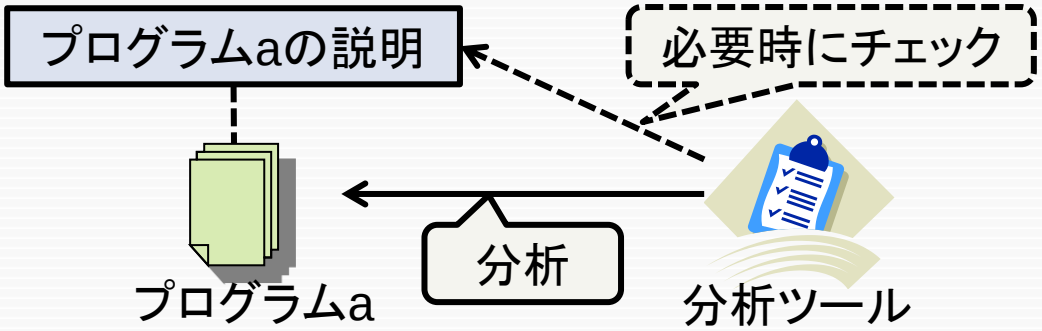
- 表現の変更
  - ✓ 最適化を別ステージとし，早い段階でステージの概要(プログラムの説明)を明らかにする



3. 振舞いの抽象化のための開発環境

- 絶え間ない分析
  - ✓ ステージごとに分析を行う
    (テストスイート実行，タイプチェック，前ステージのプロパティ維持，など)
  - ✓ 開発者に警告後分析を中断せず，更なる分析を防がない

- 分析の自動化
  - ✓ 基本的に分析は自動プロセスだが，手動入力が必要な場合がある
    ⇒ 近いレベルの説明から，自動的に入力すべき値を発見する

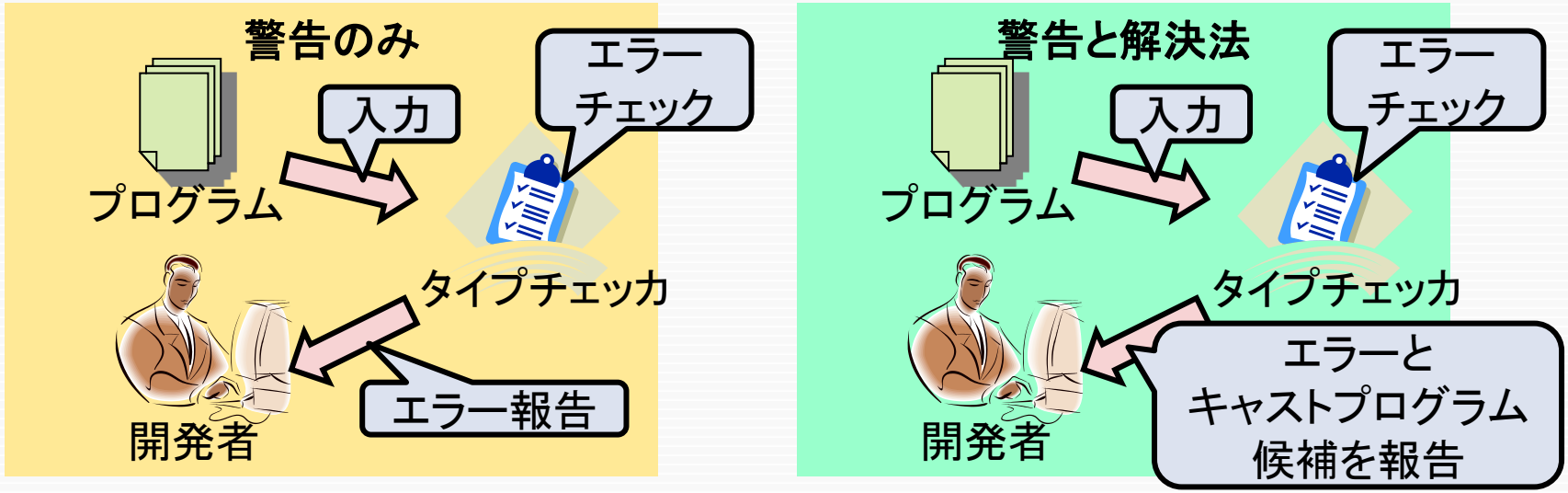


4. 挑戦

- 振舞いの抽象化をサポートする開発環境に必要なこと
  - ユーザインタフェース
    - ✓ 現在：ファイルベース統合開発環境
      ⇒ ステージ間の違いを確認できない
    - ✓ ファイルベースとは異なるインタフェースを利用
  - 早期のシミュレーション
    - ✓ 抽象度の低いステージほど，テストやシミュレーションが容易
    - ✓ 抽象度の高い初期ステージをどのようにシミュレーションするか？
      ⇒ 初期ステージに誤りがある場合，大きな手戻りのコスト
    - ✓ 初期ステージのシミュレーション方法を考慮
      ex) the spirit of Alloy (Alloy Analyzer)

4. 挑戦

- 分析の優先度
  - ✓ 分析の結果からエラーや警告を知らせるだけでなく，解決法を提案する
    例）タイプチェッカ



  - ✓ 開発者が行うことに必要な情報をもたらす分析を優先する

- 分析の続行
  - ✓ エラー発見時にエラー部分を補い，テスト/シミュレーションを続行する
  - ✓ どのようにエラー部分を補うのか？

5. 終わりに

- ソフトウェア工学の改善のために
  - 振舞いの抽象化を中心とする開発環境について議論
    - ✓ プログラムはモジュール，タイプ，プロセスに応じて分割されるのではなく，抽象化のレベルによっても分割される

- 考察

振舞いの抽象化をサポートすることで、現在よりも効率的に開発を行えるのではないかと考えた．しかし，具体的に開発環境でどのようなユーザインタフェースを利用すべきか記述がないため，振舞いの抽象化をサポートするユーザインタフェースを考慮する必要がある．

6. 参考文献

- Tools and Behavioral Abstraction : A Direction for Software Engineering / K. Rustan M. Leino