産学連携による実証的ソフトウェア工学研究事例

大阪大学　大学院情報科学研究科
コンピュータサイエンス専攻
楠本真二

実証的ソフトウェア工学 (Empirical Software Engineering)

- ソフトウェア開発の分野において，他の科学・工学分野と同様に，計測，定量化と評価，そしてフィードバックによる改善という実証的手法 (エンピリカルアプローチ) の実践を中心とした研究分野.

- 5つのステップ
  1. 現状の問題点把握
  2. 問題点を解消する手法の提案
  3. 実験/ケーススタディを通じた手法の評価
  4. 実際の開発プロジェクトへの適用
  5. 手法の改善・改良

主な研究歴

- 1990年～1994年（鳥居研究室）
  - プログラマ・開発チームの性能評価
  - ソフトウェアレビューの評価尺度
- 1994年～2000年（菊野研究室）
  - ソフトウェアプロセス改善
  - レビュー手法・レビュー工数割当法
  - 複雑度メトリクス
- 2000年～2007年（井上研究室，現在に至る）
  - 見積り
  - ソフトウェア保守支援

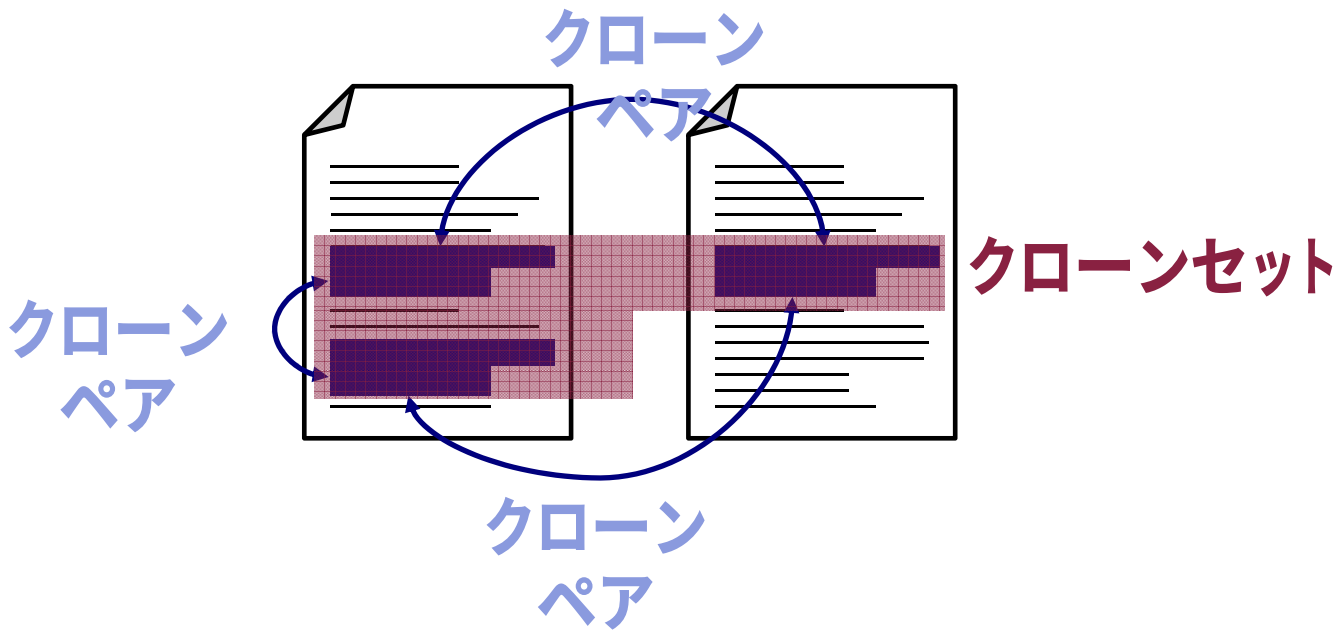
最近のテーマ

- ソフトウェア保守
  - コードクローン
  - リファクタリング
  - 複雑度メトリクス
- 見積り
  - ファンクションポイント計測
  - ユースケースポイント計測
  - 保守ポイント

コードクローン

- ソースファイル中に含まれるコード片で，同一または類似したコード片を持つもの

  (コピー&ペーストプログラミング，実行パフォーマンス向上，コーディングスタイル，偶然等で作り込まれる)



コードクローンの問題

- コードクローンはソフトウェア保守性を阻害する.
  - あるコード片にコードクローンが存在する場合，その全てのクローンについて修正を行うかどうかの検討をしなければならない.
  - 機能追加も同様

- クローン情報のドキュメント化や大規模ソフトウェアシステムからのコードクローンの抽出は，人手で実施するのは難しい.

- 大規模ソフトウェアからの高速なコードクローン検出と分析を支援する手法が必要

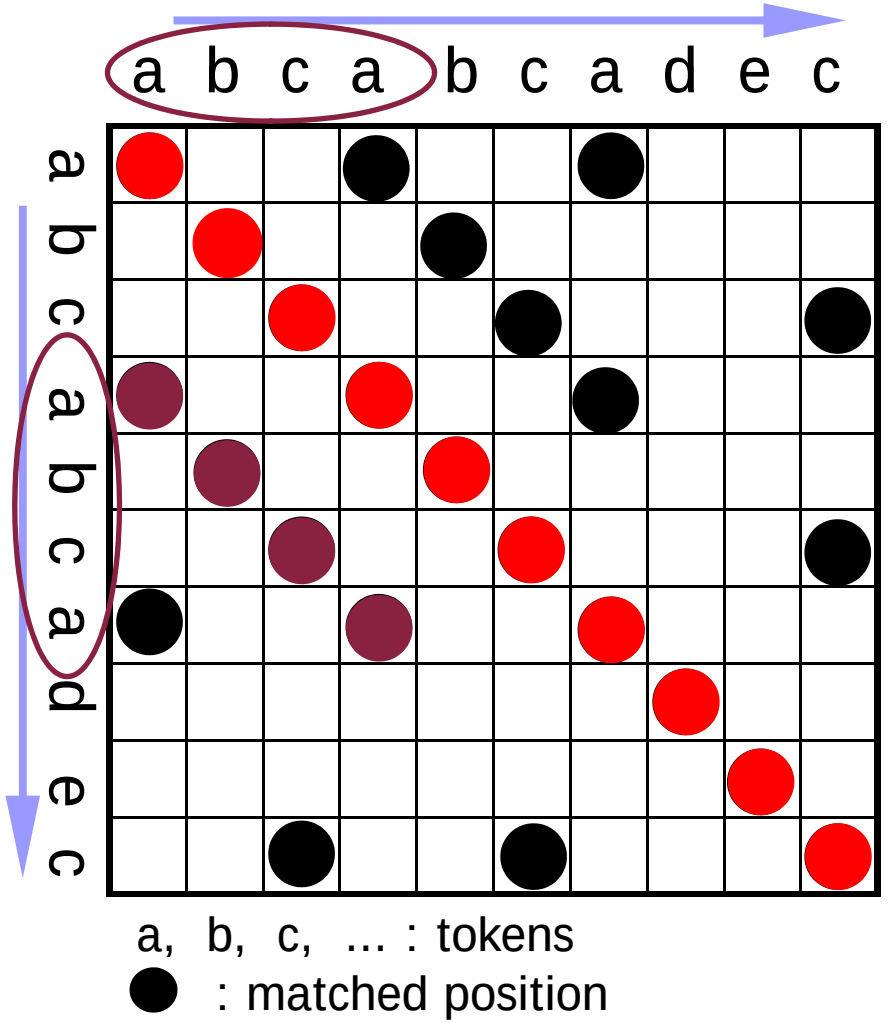
これまでの経緯

- コードクローン検出･分析ツールの開発
  - クローン検出ツールCCFinder　(字句単位のクローン検出)
  - 分析環境ICCA
    - CCFinder をコードクローン検出エンジンとして使用
    - 目的別にサブコンポーネントが存在
      - 可視化, 理解支援 - Gemini コンポーネント
      - リファクタリング支援 - Aries コンポーネント
      - デバッグ支援 - Libra コンポーネント

- ツールの適用
  - フリーソフトウェア, 大学の演習
  - 配布先各社･大学等での利用

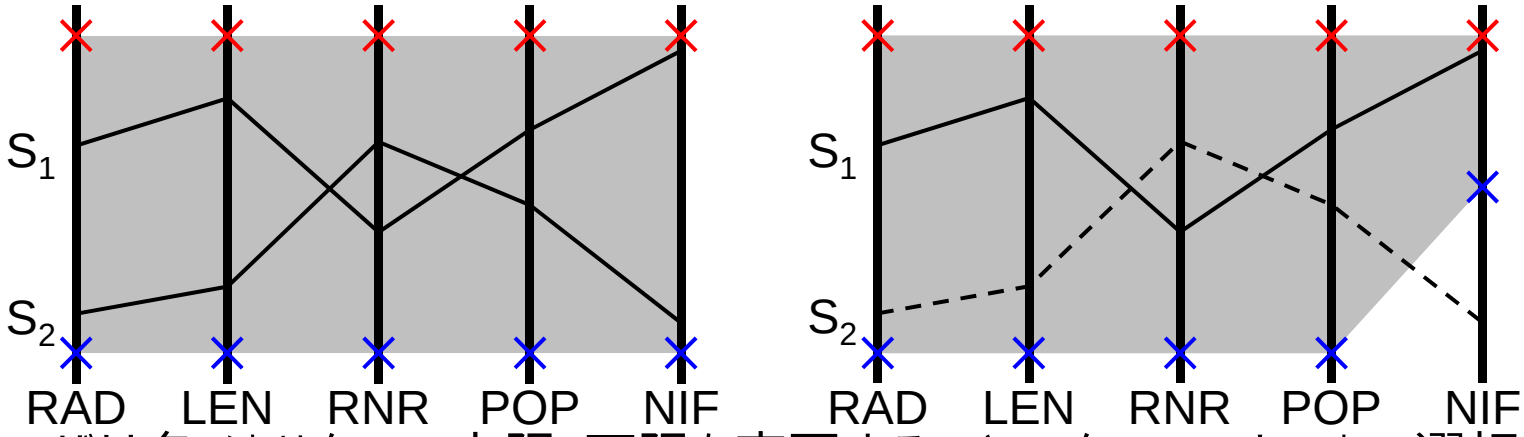
コードクローン分析ツール Gemini
クローン散布図:

- 水平・垂直方向にソースコード中のトークンを出現順に配置
  - 原点は左上隅
- ●はその水平方向のトークンと垂直方向のトークンが等しいことを意味する
  - 常に対角線が引かれる
  - クローンペアは線分として出現する
  - 対角線に対して線対称となる



コードクローン分析ツール Gemini
メトリクスグラフ(概要)

- メトリクスを用いてクローンセットを定量的に特徴づける
- 多次元並行座標表現を用いている
  - 各メトリクスにつき1つの座標軸が存在する
  - 各クローンセットにつき1つの折れ線がメトリクス値に基づいて描画される



- ユーザは各メトリクスの上限・下限を変更することでクローンセットの選択・選択解除を行う
  - 全てのメトリクスが上限と下限の間にあるクローンセットが選択状態になる
  - 選択状態にあるクローンセットのソースコードは簡単に閲覧可能

実プロジェクトデータへの適用

- 目的
  - 適用対象の開発環境でのコードクローン分析結果の有用性評価
- 適用対象
  - 事例1：某ソフトウェア開発会社
  - 事例2：SEC 先進ソフトウェア開発プロジェクト5社

分析手順

- 基本分析
- 散布図やメトリクスグラフを用いて特徴的なコードクローンを抽出
- 個々のコードクローンを確認し，コードクローンに対する処置を提示
- マネージャ，現場の開発者による結果の評価

基本分析

- ファイル数
- ディレクトリの数
- 総行数
- トークン数

- クローンセット数
- クローンペア数
- いずれかのコードクローンに含まれるコード片数
- 重複度
  - いずれかのコードクローンに含まれるトークン数／ファイル全体のトークン数

事例1:適用対象

- 二つのバージョンのJavaソースコードを利用
- 規模
  - V1： ファイル数: 約750, 行数:約180KLines
  - V2： ファイル数: 約850, 行数:約220KLines

  ※行数には、空行・コメント行を含む。
- 6種類のパッケージ(それぞれサブパッケージ含む)から構成
  - スクラッチ
  - 自動生成
  - 半分自動生成

事例2：先進ソフトウェア開発プロジェクト

- 自動車プローブ情報システムプラットフォームの開発
- COSE（ソフトウェアエンジニアリング技術研究組合）のプロジェクトマネージャ
  - ベンダ5社の開発
  - 各社が独立して開発
- 単体テスト，結合テスト終了後に分析

主な意見

- プログラム設計時に共通化可能な処理について検討しておくべきであった.
- ファイル作成者毎にこのような分析が継続して実施可能であれば、事前に障害を防ぐための情報として有用なものになる.
- 指摘された部分の処理内容を考えると，更にクローンを作成する可能性は小さいと予想される.
- 指摘された部分を集約すると、プログラムの構造が複雑になり理解しづらくなる可能性が高い.
- 新規作成時に有用とはない.
- 既設プログラムの改造を行う場合等の改造漏れを防ぐ指針としては有効ではないか.

考察

- 個々のコードクローンに対して，それなりの理由が存在する．
  - 悪いコードクローンばかりでは無い．

- 計測対象のソースコードのみの情報ではコードクローンの特徴について判断できない．
  - 履歴データの利用
    - 繰り返し修正されているコードクローン
    - ほとんど修正されていないコードクローン
  - 品質データを加味した総合的な評価
    - コードクローンに伴うバグの修正漏れの影響

その他の活動

- ソフトウェア工学工房セミナー：コードクローン検出技術とその応用（過去12回開催）
- 超大規模プログラムに対するコードクローン分析
- APIの使用に伴うコードクローンの特徴分析
- 類似バグ検出

見積り

- ファンクションポイント法，ユースケースポイント法
  - ソフトウェアの機能規模を計測する手法
    - 規則にしたがって計測される値
    - 機能仕様にだけ依存（開発環境，プログラミング言語とは独立）

- 実用面での課題
  - 測定者によって誤差が生じる．
  - 基礎データとして用いるために，過去に開発されたソフトウェアの値を計測する必要がある．
  - 計測導入のための初期コスト（教育等）が必要となる．

これまでの経緯

- 計測の自動化
  - IFPUG法，ユースケースポイント法
- 計測対象
  - REQUARIOで作成された要求仕様書
  - UML(Unified Modeling Language)で記述された仕様書
  - Javaプログラム
  - ユースケースモデル (ユースケースポイント)

ユースケースポイント法

- G.Karnerが提案
- プロジェクトの開発工数（人時）を見積る一手法
  - オブジェクト指向開発での要求分析段階で作成される，ユースケースモデルを利用
  - 重み付けしたアクタ，ユースケースのカウント
    - 3種類（単純・平均的・複雑）に分類，重み付け
  - プロジェクトの技術的複雑さ，開発環境を考慮した調整

ユースケースモデル

- システムの動作・機能要求をユーザ視点で表現
- 構成
  - ユースケース図
    - UMLによる視覚化
    - アクタ：システムを使用する人間，関連する別システム
    - ユースケース：ユーザが望むシステムの動作
  - ユースケース記述
    - ユースケースの機能を実現するために必要な詳細情報を記述
    - ユースケースシナリオ：動作の記述部分
    - イベント：ユースケースシナリオの各行

研究の目的

- ユースケースポイントツールの試作
  - アクタ，ユースケース分類（重み付け）の自動化
- 実際のWebアプリケーションのユースケースモデルへの適用および評価
  - 5プロジェクトへの適用
  - 人手による計測結果との比較

アクタの分類

タイプ	説明	重み
単純	定義済みAPIを備える (外部システム)	1
平均的	プロトコル駆動のインタフェース (外部システム) テキストベースのインタフェース (人間)	2
複雑	GUIを介する (人間)	3

- 問題点
  - アクタとシステム間のインタフェースに関する情報をどこから得るか
- 分類手法
  - アクタ名に関するキーワード (ユーザ設定可) により，外部システムと人間とを分類
  - 関連するユースケースのイベントに対する，キーワードマッチング
    - そのアクタが主体となるイベントを抽出
    - インタフェースに関連するようなキーワード群の設定 (複雑：「ボタン」「押す」など)
    - マッチングの割合の高い複雑さを採用

ユースケースの分類

- ユースケースの内容の詳細を記述した，ユースケースシナリオ内のトランザクションの数に基づく．

タイプ	トランザクション数	重み
単純	3個以下	5
平均的	4～7個	10
複雑	8個以上	15

- 問題点
  - 自然言語で書かれたイベントからトランザクションを識別
  - イベントの書き方は明確に定められているわけではない

トランザクション抽出方法

(1) ユースケース記述から事前条件を，各イベントから条件部分をそれぞれ削除する．
   - 事前条件の記述部を削除
   - 条件部分は「～の場合」等と書かれている文節を削除

(2) 各イベントの主語と動詞の対応をとり，主語が省略されている場合は補完する．
   - ツール「Cabocha」の利用．主語と動詞の対応ルールを用いる

(3) トランザクションの対象とならないイベントを削除する．
   - 当該イベントに含まれる動詞リストの利用

(4) トランザクションを構成するイベントをまとめる．
   - アクタが主語のイベントとそのイベントで引き起こされるシステムが主語のイベントを一つにまとめる

適用と評価

- 5つのWebアプリケー書開発プロジェクトに対する適用
- 熟練した経験者による手動での分類結果との比較

プロジェクト	ユースケース数
A	15
B	14
C	20
D	28
E	13

適用と評価 (アクタ)

プロジェクト	手動での分類			ツールでの分類			複雑度の一致した割合
	単純	平均	複雑	単純	平均	複雑	
A	1	0	4	0	1	4	0.8
B	3	0	2	2	1	2	0.8
C	0	0	2	0	0	2	1.0
D	1	0	4	1	0	4	1.0
E	0	0	8	0	0	8	1.0

適用と評価 (トランザクション数)

	手動での合計	ツールでの合計	トランザクション 合計の比率
A	85	85	1.0
B	90	90	1.0
C	130	140	1.08
D	145	145	1.0
E	135	145	1.07

- トランザクション合計の比率
  - 全てのユースケースで検出されたトランザクションの合計比較
  - 手動を理想値として，完全一致の場合1.0
  - 少し手動より多くトランザクションを数えている傾向

適用と評価（3段階複雑度）

プロジェクト	手動での分類			ツールでの分類			複雑度の一致した割合
	単純	平均	複雑	単純	平均	複雑	
A	13	2	0	13	2	0	1.0
B	10	4	0	10	4	0	1.0
C	14	6	0	12	8	0	0.9
D	27	1	0	27	1	0	1.0
E	2	8	3	2	8	3	1.0

- プロジェクトC
  - 単純（3以下）と平均的（4以上）の境界を挟むずれが，2個あった．

まとめ

- 2つの産学連携事例について紹介
  - コードクローン分析
  - ユースケースポイント計測

- 今後の課題
  - 実際の開発プロジェクトへの適用
  - 手法の改善・改良