コード分析技術に基づくソフトウェア品質の評価と改善
～コードクローンの検出とその応用～

大阪大学大学院情報科学研究科
井上　克郎

ソフトウェア依存社会のリスク

航空管制システム障害
欠航215便、遅延1500便以上、
足止め客30万人以上
2003年3月1日

ATCのプログラムミス
300系新幹線計100編成で機能停止
2005年3月22日

東証システムダウン
東証1部、2部、マザーズ
など全2520銘柄が停止
2005年11月1日

ABSの制御プログラム不具合でハイブリッド3車種21万台リコール
2010年2月10日

証券誤発注取り消し不能
損失400億円以上
2005年12月8日

ひかり電話トラブル
7日間にわたって通話不安定
2006年9月19日

日本国内の社会的損失額約3兆円／年（国内総生産の0.6%）†

†日米でソフトウェア開発プロジェクトの成功率はほぼ同じとし、米国商務省国立標準技術研究所（NIST）調査による2002年の米国の社会的損失595億ドルを、日本のGDP515.9兆円（2007年、名目）にあてはめて計算

2

自動改札機のトラブル（2007年10月12日朝）

首都圏のJR、私鉄、地下鉄各社局などの駅において、始発からある会社の自動改札機のみ作動しなくなるトラブルが発生

662駅の4378台が起動せず、260万人に影響

おわびプレスリリースの概要

・トラブルの原因は、ICカード判定部を搭載した自動改札機において、中央のコンピューターから送信されたデータをICカード判定部の記憶部に読み込むプログラムの一部に不具合
・データを正常に読み込むことができず機器異常となり改札機がダウン
・このデータ数がある件数以上、且つある条件下で発生する仕組みであった為、品質保証の過程で発見されず、当日朝、中央から配信されたこのデータ数が条件範囲にヒットした事により、顕在化
・従来このような場合の検証は、最小値や最大値または中間といった節目での検証を行っていた
・今後は無差別検証を更に増加し、事前・事後の設計審査を充実させ、再発の徹底防止を図る
・このため膨大なソフト検証の為の自動ツール化を更に推進する
・現納入機についても、事業所内の同機種にて今後数ヶ月に亘る再々検証を予定
・今回や前回（2006 年12 月1 日）の事態は、「安全と信頼」の企業理念を深く傷つけたものであり、今後は一日も早い信頼の回復に全社員が総力をあげて取組む

3

自動改札機のトラブル（2007年10月18日朝）

首都圏のJR、私鉄、地下鉄各社局などの駅において、始発から同じ会社製の窓口処理機でカード処理ができなくなるトラブルが発生

65駅の101台が不具合、400人に影響

おわびプレスリリースNo.2の概要

・自動改札機と同様の不具合
・12日には発生せず、18日発生したのは、受信データの形式が異なり、不具合条件を満たすデータ件数が異なってたため
・12日のトラブルのあと、社内で同様の不具合がないか、検証したが、プログラム内容に関する判断ミスにより、不具合は発生しないと考えた
・今後、特定のデータの配信を取りやめる
・同様な誤りがないか、全機器で総点検をする

4

教訓多し

- テストの考え方が？
  - テスト戦略は妥当か？
  - 数をこなせば十分か？
- 設計は大丈夫？
  - ある境界を越えた場合に発生する特殊な条件とは？
  - チェック失敗しても、縮退運転しないの？
- 管理体制は大丈夫？
  - コードの流用管理不十分
    - 類似したコードを検出する技術があれば…
  - 危機管理は機能しているか
    - 担当者全員に関連コードが安全か確認すれば2回目の事故は防げた？

5

トヨタ車のアクセルの加速問題について

- 2011年に話題になった゛意図しない"加速問題
  - ソフトウェアかハードウェアのバグ？（政治家やジャーナリスト）
  - フロアマットの設置不良か運転者のミス(トヨタ)
- 全米高速道路交通安全委員会(NHTSA) がNASA工学・安全センター(NESC)に調査を依頼
  - ハードウェア
  - ソフトウェア

6

スロットル制御システムETCS-iのソフトの分析

問題を引き起こす可能性を4つに分類

1. コードの欠陥
2. アルゴリズムの欠陥
3. タスク間の干渉
4. 不十分な欠陥からの回復

それぞれの分類について、問題を引き起こす可能性がある欠陥を列挙

例えば、配列の範囲外アクセス、Nullポインターアクセス、ループの非停止、…

それぞれの欠陥が発生し、かつ当該の問題を引き起こす可能性をチェック

7

各種ツールの利用

- 静的解析ツール
  - gcc
  - Converty
  - CodeSonar
  - Uno
- コード規約違反検出ツール
  - MISRA-C
  - NASA/JPL Rule
- モデル検査ツール
  - Spin
  - Swarm
- アルゴリズム設計
  - Matlab
  - Simulink, Stateflow, SystemTest
  - aiT

ツールが報告する警告
→ 問題につながるかをチェック

ソースコードからモデルや設計図を作成
→ 問題を起こす可能性のチェック

8

2011年1月18日付けNESC報告書

- 意図しない加速につながるような設計や実装の欠陥は発見できなかった
- ソフトウェアの信頼性の評価法のケーススタディ

9

外部機関によるソフトウェアの品質の評価

一般的に難しい（開発者自身でも難しいのに…）

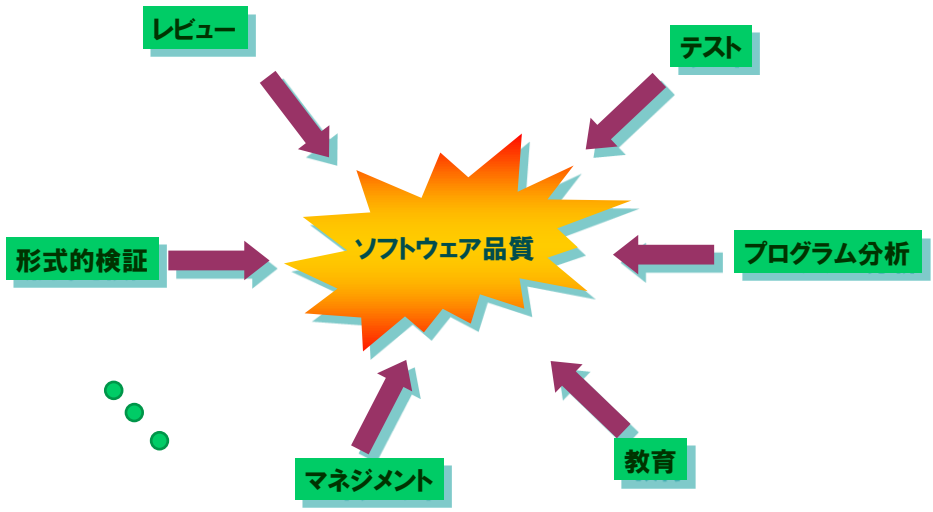
- 設計、開発プロセス、テストなど、開発に関わる情報が少ない
- 残されたプロダクト（ソースコードが主）から品質の推定する技術は未成熟
  - ソースコードの静的解析ツールによる分析・評価
  - 設計やテスト情報の復元・評価

しかし、今後この種のニーズは増加するのは確実

IPA/SEC ソフトウェアの品質説明力強化のための制度フレームワークに関する提案（中間報告）、2011年8月5日.

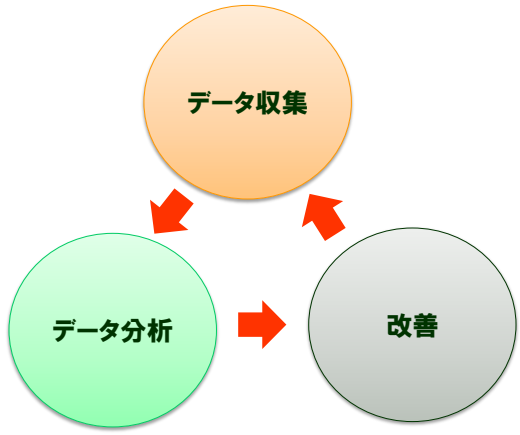
10

ソフトウェアの品質向上に対するアプローチ



11

品質改善への実証的なアプローチ



12

コードクローン

コードクローンとは?

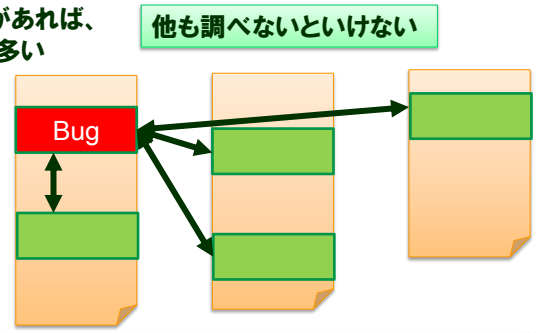
- 同型のコード断片を(同一または他の)ファイルに持つコード断片

```
AFG::AFG(JaObject* obj) {
  objname = "afg";
  object = obj;
}
AFG::~AFG() {
  for(unsigned int i = 0; i < children.size(); i++)
    if(children[i] != NULL)
      delete children[i];

 ...

  for(unsigned int i = 0;
                  i < nodes.size(); i++)
     if(nodes[i] != NULL)
     delete nodes[i];
}
```

コードクローンのできる理由

- コピー＆ペースト
- 定型のコーディング
  ex. file open, DB connect, …
- 効率向上のためにわざと

- プログラムの保守をより難しくする要因
  - ロー箇所のコード断片に問題があれば、他の部分も問題ある場合が多い
    - よく見逃しがち



15

コードクローンの研究と応用の流れ

- 1990年代、コードクローン分析手法についてのいくつかの研究
- 2002年で効率のよいコードクローン検出ツール CCFinderを開発、無償で提供、論文発表
- 多くの研究用ツールの開発
- 企業での試用
- 商用ツールの発売
- オープンソースツール

16

コードクローン検出ツール
CCFinder

CCFinder開発の流れ

- 企業からのニーズが開発のヒント
  - 巨大システムの保守（1000万行、20年以上、横並びリストの限界）
- CCFinderを開発（数百万行のコードを5～30分で分析）
- CCFinderを核とした種々の支援ツールの開発
  - Gemini　可視化・分析支援ツール
  - Aries　リファクタリング支援ツール
  - Libra　類似コード片検索支援ツール
  - Ikka　統合支援ツール
  - CloneInspector　コピー＆ペースト修正間違い検出ツール
- 産学連携活動の強化
  - セミナー活動、共同研究
- ソフトウェアクローン国際会議IWSCの立上
- CCFinderXの開発、オープンソース化

18

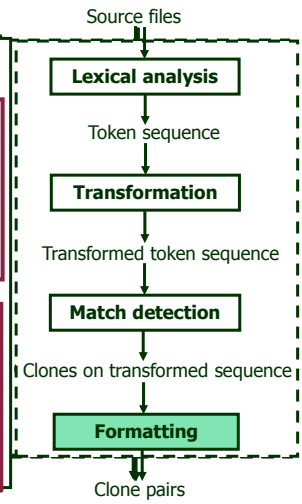
コードクローンの定義

- 簡単な統一的な定義はない（いろいろな研究者が定義）
- しかし、おおまかな統一認識
  - Type 1 clone: コメント、空白等を除き文字列的に同一
  - Type 2 clone: 固有名を正規化して同一（parameterized）
  - Type 3 clone: コードの挿入、削除
  - Type 4 clone: 意味的に同一
- いろいろな検出法
  1. 行ごとの照合 (type 1)
  2. AST (Abstract Syntax Tree)の比較 (type 2, 3)
  3. PDG (Program Dependency Graph)の比較 (type 3)
  4. メトリクス値の比較 (type 3)
  5. トークン列比較 (type 2)

19

CCFinder のコードクローン検出プロセス

```
1. static void foo() throws RESyntaxException {
2.   String a[] = new String [] { "123,400", "abc", "orange 100" };
3.   org.apache.regexp.RE pat = new org.apache.regexp.RE("[0-9,]+");
4.   int sum = 0;
5.   for (int i = 0; i < a.length; ++i)
6.     if (pat.match(a[i]))
7.       sum += Sample.parseNumber(pat.getParen(0));
8.   System.out.println("sum = " + sum);
9. }
10. static void goo(String [] a) throws RESyntaxException {
11.   RE exp = new RE("[0-9,]+");
12.   int sum = 0;
13.   for (int i = 0; i < a.length; ++i)
14.     if (exp.match(a[i]))
15.       sum += parseNumber(exp.getParen(0));
16.   System.out.println("sum = " + sum);
17. }
```

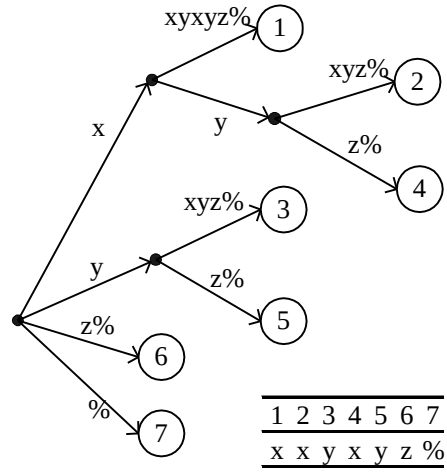


20

接尾木Suffix-tree

接尾木の特性
1.葉は各部分列に対応
2.パスが部分系列表現
3.各頂点から異なるラベルで分岐
→ 共通部分パスがクローン

		1	2	3	4	5	6	7
		x	x	y	x	y	z	%
1	x	*						
2	x	*	*					
3	y			*				
4	x	*	*		*			
5	y			*		*		
6	z						*	
7	%							*

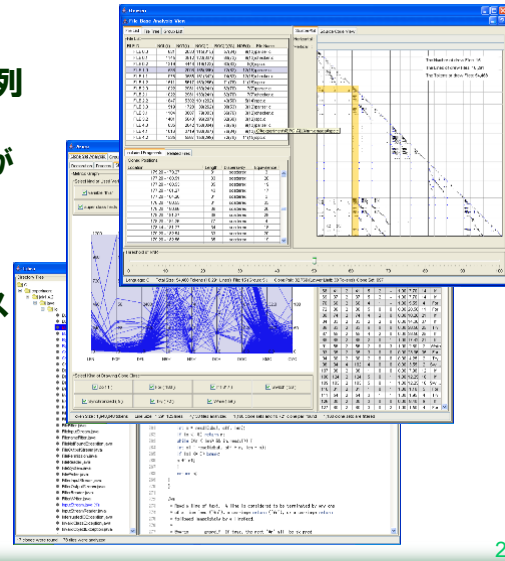


1	2	3	4	5	6	7
x	x	y	x	y	z	%

21

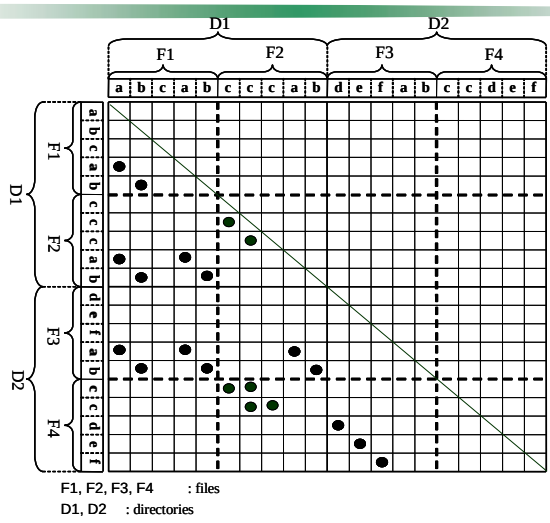
視覚化・分析支援ツールGeminiの概要

- CCFinderの出力を視覚化
  - CCFinderの出力は文字列
- 対話的に分析を進めることができる
  - 散布図
  - クローンのメトリクス
  - ファイルに関するメトリクス
- 不必要な出力のフィルター



22

Geminiの散布図



23

Gemini が計算するメトリクス

- クローンクラスS(同一コード片集合)に対するメトリクス
  - *LEN(S)*: Sの平均長(トークン数)
  - *POP(S)*: Sの要素数
  - *NIF(S)*: Sの各要素を含むファイルの数
  - *RNR(S)*: Sの要素の非繰り返し率(0～1、0に近いとほとんど同一文の繰り返し)
- あるファイルFに対するメトリクス
  - *ROC(F)*: Fに含まれているコードクローンの比率
    - 全部がコードクローン: 1
    - コードクローンを含まない: 0
  - *NOC(F)*: Fに含まれているコードクローンの断片数
  - *NOF(F)*: Fに含まれているコードクローンと同型断片を含むファイル数

24

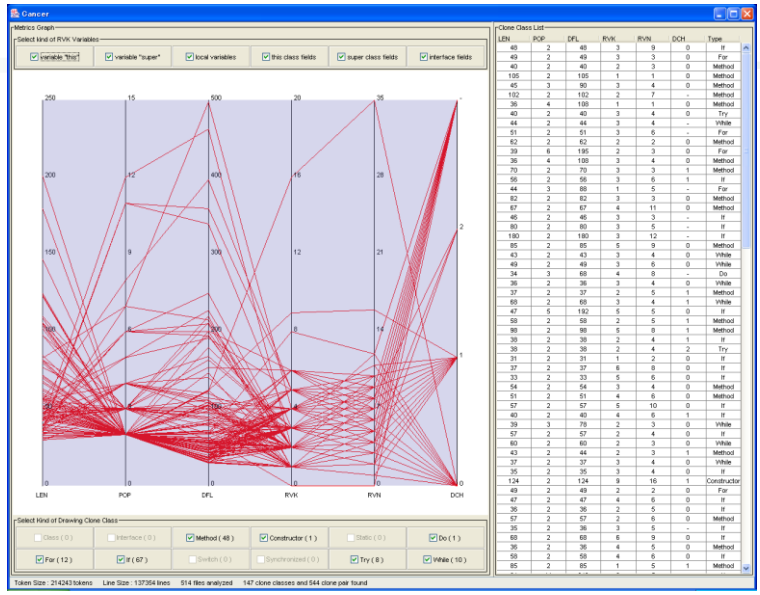
リファクタリング支援ツールAries

- **構造的に分離可能なコードクローンを取り出す**
  1. CCFinderでコードクローン対検出
  2. **クローン集合にまとめる**
  3. **軽い構文解析を行い構造的な切れ目で取り出す**

  - **例**: Java
    - **宣言部**: class declaration, interface declaration
    - **メソド**: method body, constructor, static initializer
    - **文**: do, for, if, switch, synchronized, try, while

25

26



26

27

クローン片1

```
609:    reset();
610:    grammar = g;
611:    // Lookup make-switch threshold in
612:    if (grammar.hasOption("codeGenM
613:        try {
614:            makeSwitchThreshold =
615:            //System.out.println("setti
616:        } catch (NumberFormatExcept
617:            tool.error(
618:                "option 'codeGenMa
619:                grammar.getClassN
620:                grammar.getOption(
621:            );
622:        }
623:    }
624:
625:    // Lookup bitset-test threshold in the
626:    if (grammar.hasOption("codeGenBi
627:        try {
628:            bitsetTestThreshold = gra
```

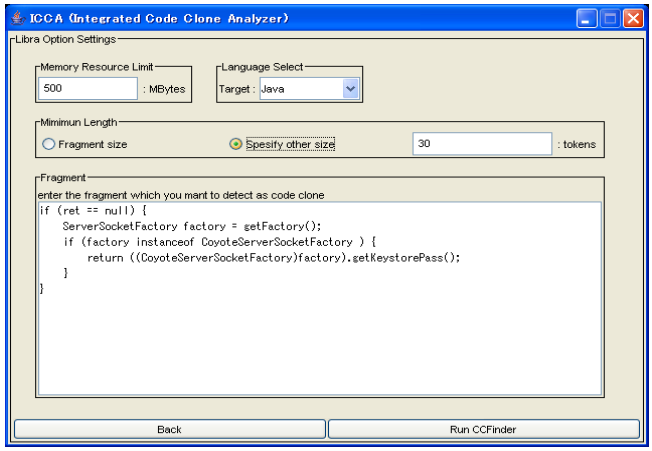
クローン片2

```
623:    }
624:
625:    // Lookup bitset-test threshold in the gr
626:    if (grammar.hasOption("codeGenBitse
627:        try {
628:            bitsetTestThreshold = gramm
629:            //System.out.println("setting
630:        } catch (NumberFormatException
631:            tool.error(
632:                "option 'codeGenBitset'
633:                grammar.getClassNam
634:                grammar.getOption("co
635:            );
636:        }
637:    }
638:
639:    // Lookup debug code-gen in the gram
640:    if (grammar.hasOption("codeGenDebu
641:        Token t = grammar.getOption("co
642:        if (t.getText().equals("true")) {
```

27

類似コード片検索支援ツールLibra

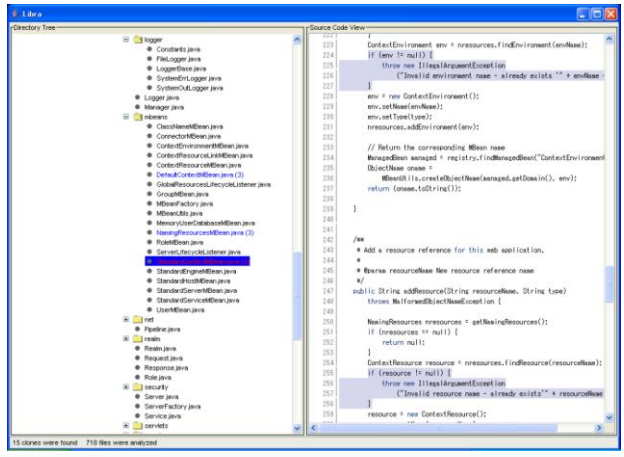
■コード片入力



28

Libraによる検索結果例

- 入力片と対象システムとの間のコードクローンのみ検出



29

適用例

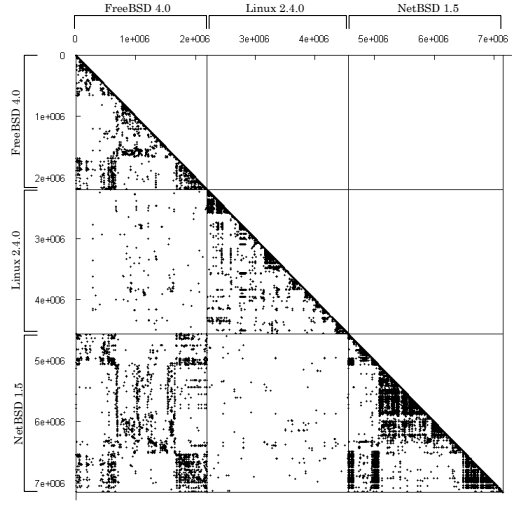
適用事例概要

- Open Source Software
  - FreeBSD, NetBSD, Linux (C, 7MLOC)
  - JDK Libraries (Java 1.8MLOC)
  - Qt (C++, 240KLOC)
- 種々の会社のシステム(100社以上)の分析
  - IPA/SEC, NTTデータ, 日立, 日立GP, 日立SAS, 富士通, NEC, NECソフト, SRA, Samsung, JAXA, etc…
  - 自社ソフトウェアの分析、納入ソフトウェアの分析
- 大学での演習課題の分析
- 裁判の証拠

…

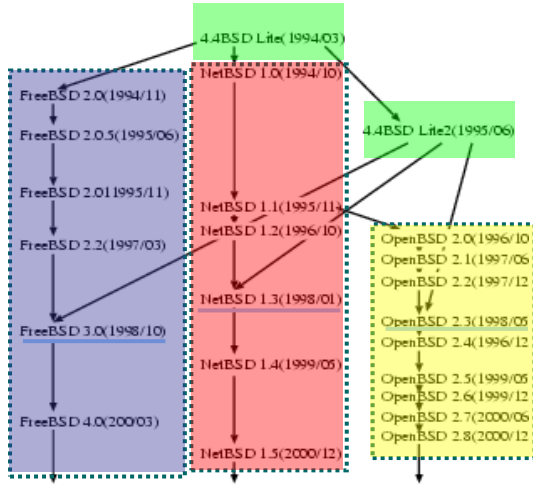
31

FreeBSD, NetBSD, Linux



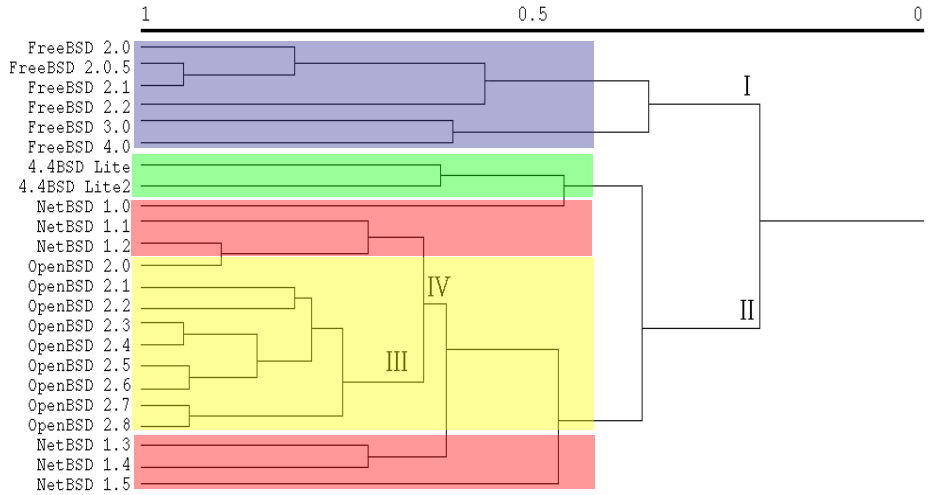
32

BSD Unix OSの歴史



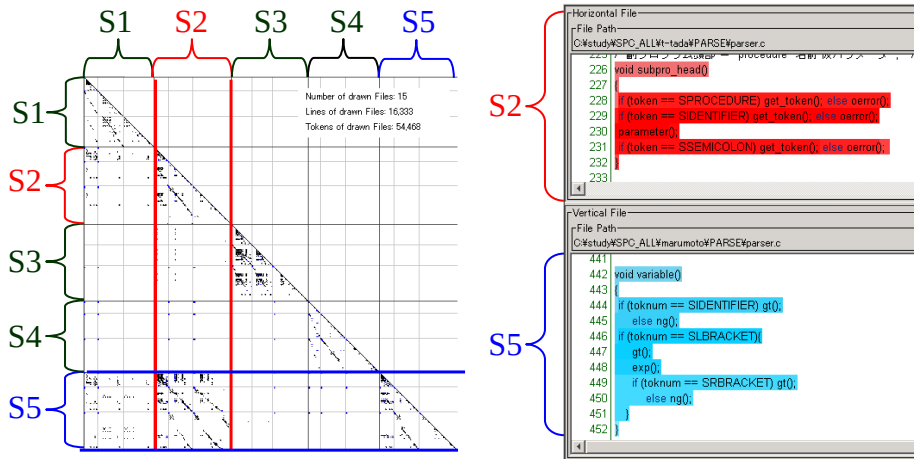
33

クローン率を距離としたクラスター分析による系統木



34

大阪大学の学生演習への適用



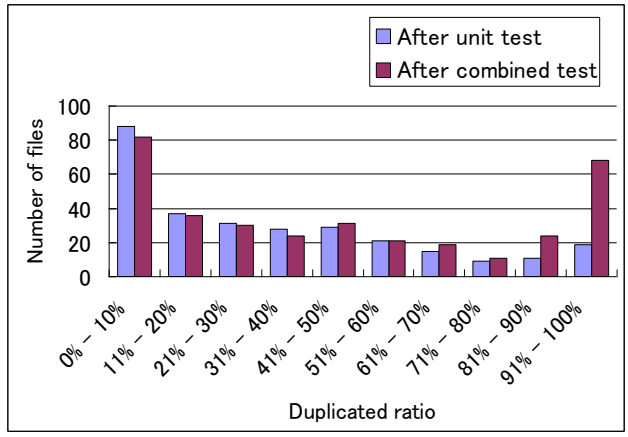
35

IPA/SEC 先進プロジェクト

- 対象
  - 種々のセンサーから情報収集を行う交通情報システム
  - センサーの種類ごとに5社で開発。全体のマネージャーは個別のサブシステムに関する知識はあまりない
- 分析方法
  - 単体テスト前(280,000LOC),結合テスト後 (300,000LOC)の2回分析
  - 最小検出トークン長30

36

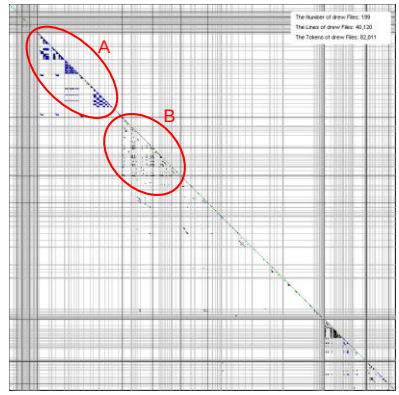
コードクローン率



- 結合テストの直前にシステムの整理
  - ライブラリの整備

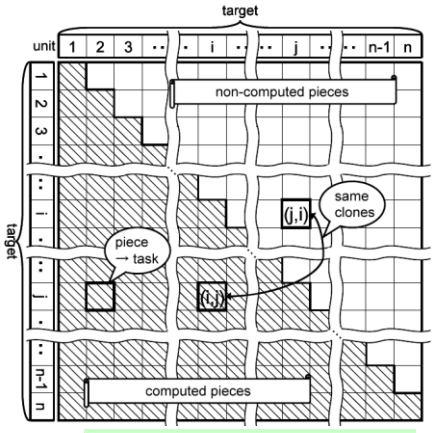
散布図の例

- ある会社のコードの散布図
- Aの部分：自動生成
  - デバッグ用プリント文
  - データの整合性チェック文
  - 連続if文
- Bの部分：ディレクトリにまたがる多くのクローン
  - 車の位置測定部
  - 車の種類ごとにディレクトリ（タクシー、バス、トラック、…）
- 各車種毎の論理構造はほとんど同一



大規模な対象の分析(タスク分割による並列化)

驚異的並列問題(Embarrassingly parallel problem)

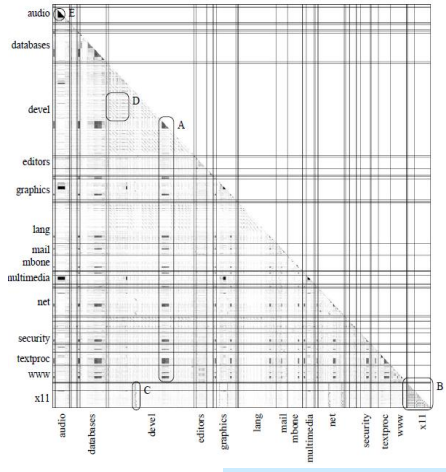


各四角ごとにCCFinder適用

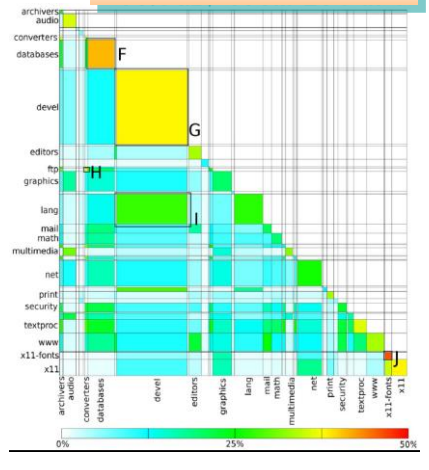
D-CCFinder (分散CCFinder)

仮想PCクラスター(学生演習室のPC80台)

FreeBSDのアプリケーション集(Ports Collection)

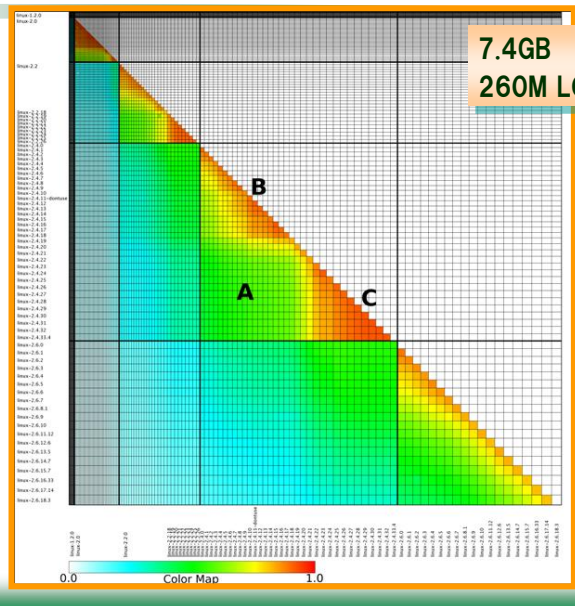


10.8GB/403M LOC in C



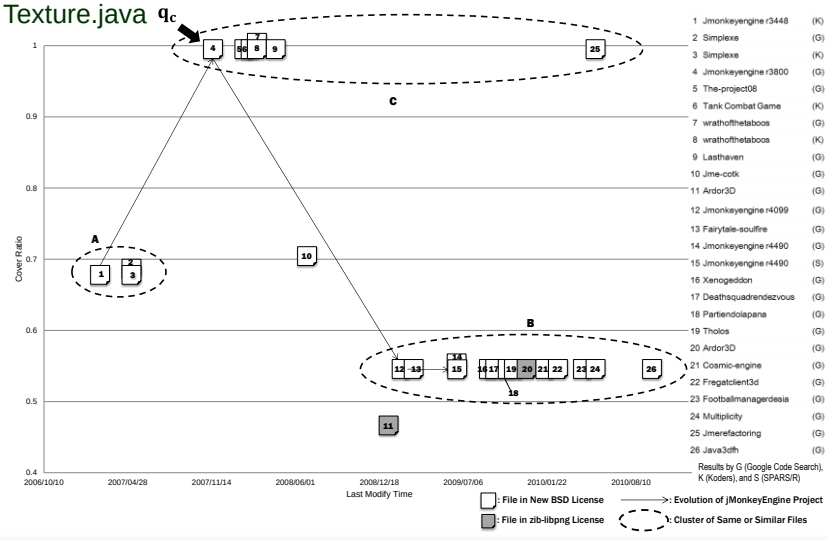
Livieri, S., Higo, Y., Matsushita, M., Inoue, K., "Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder", International Conference on Software Engineering, Minneapolis, MN. (May 2007, to appear)

Linux カーネル136バージョンの間の分析



41

コードの起源や発展の調査



42

まとめ

まとめ

- システムトラブルの事例
  - 自動改札、アクセル
- コードクローン
  - 検出手法
  - 応用例
- ソフトウェア分析の一手法
  - 他の分析方法との併用
- ソフトウェアの品質の改善への道筋

リソース

■論文

神谷年洋, 肥後芳樹,吉田則裕: "コードクローン検出技術の展開", コンピュータソフトウェア, Vol.28, No.3, pp.29-42, 2011.
井上克郎, 神谷年洋, 楠本真二: "コードクローン検出法", コンピュータソフトウェア, Vol.18, No.5, pp.47-54, September 2001.(http://sel.ist.osaka-u.ac.jp/~lab-db/betuzuri/archive/349/349.pdf)
T. Kamiya, S. Kusumoto, and K. Inoue, CCFinder: A multi-linguistic token-based code clone detection system for large scale source code, IEEE Transactions on Software Engineering, vol. 28, no. 7, pp. 654-670, Jul. 2002.

その他いろいろ　HP参照

■ウェブ

- CCFinder:
  http://sel.ist.osaka-u.ac.jp/cdtools/index-e.html
- CCFinderX:
  http://www.ccfinder.net/ccfinderx.html