Javaソースのコードクローンと品質改善活動
～NTTコムウェアにおける取り組み事例～

NTTコムウェア
堂山真一

doyama.shinichi@nttcom.co.jp

ＣＣ分析例

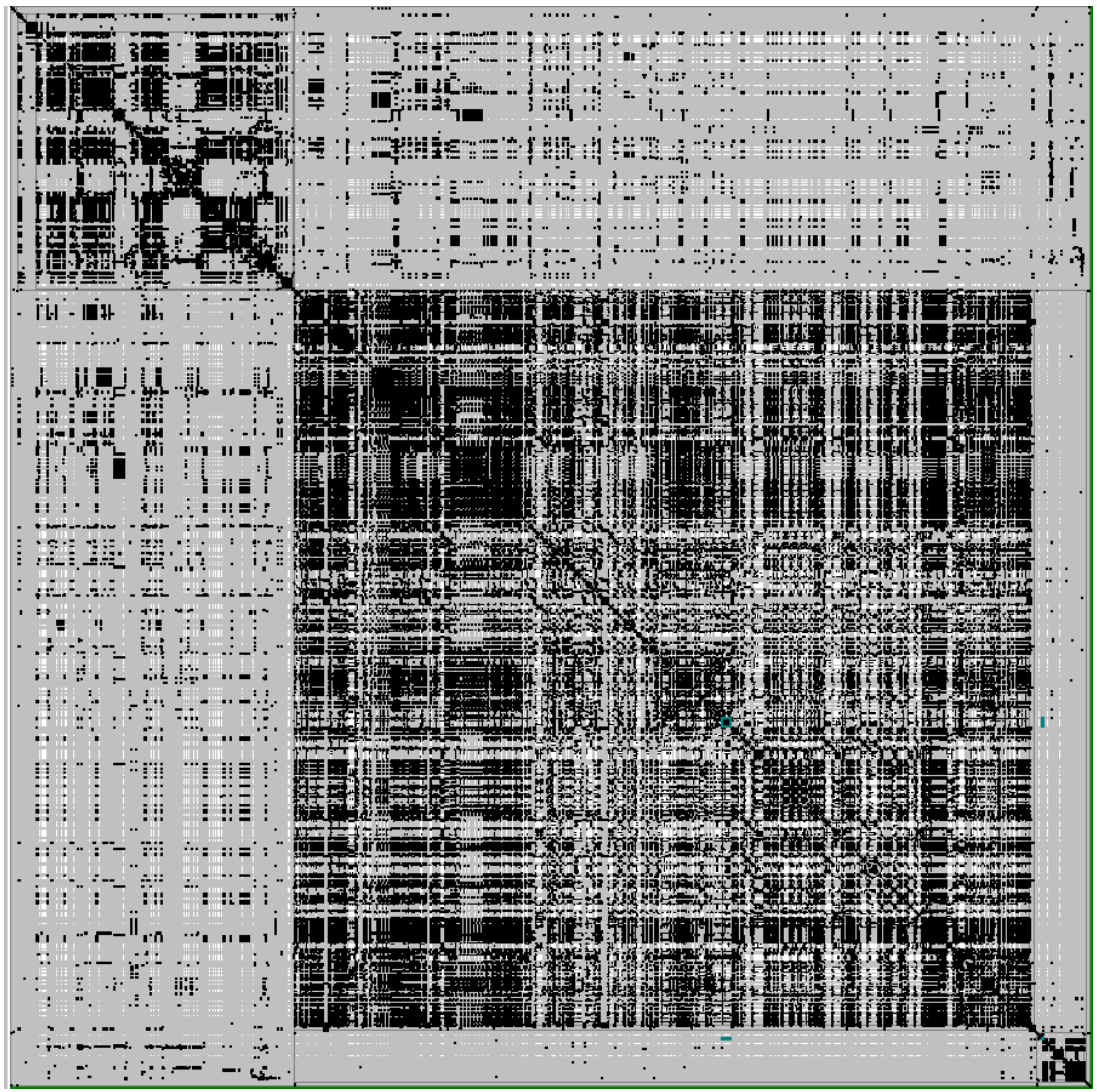
「ナスカの地上絵」

「ソースの鳥瞰図」

サブシステムに分かれている。

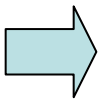
機能追加が繰り返された。

ツールを利用している。



doyama.shinichi@nttcom.co.jp

Agenda



1. コードクローン（ＣＣ）分析の位置づけ
2. ＣＣは、なぜ生まれるか
3. ＣＣは、すべて悪いのか
4. ＣＣと、どうつき合うか

doyama.shinichi@nttcom.co.jp

品質改善活動の全体像

1. 要求仕様完成度の向上
2. 見積もり精度の改善
3. フレームワークの標準化
4. 早期プロトタイプによる検証
5. マイルストーンによる進捗管理
6. ソース検査による品質チェック 
7. 負荷試験による品質チェック

ＣＣ分析例

C	2000ファイル	3ML	600万トークン
C++	5000ファイル	4ML	600万トークン
Java	4500ファイル	3ML	400万トークン

巨大なソフトウェアであるほど、全体像を捉えることが難しい。

巨大なソフトウェアほど、高い信頼性が求められ、

ライフサイクルが長い傾向がある。

doyama.shinichi@nttcom.co.jp

Agenda

1. コードクローン（ＣＣ）分析の位置づけ
2. ＣＣは、なぜ生まれるか
3. ＣＣは、すべて悪いのか
4. ＣＣと、どうつき合うか

ＣＣは、なぜ生まれるか

1. たまたま似たルーチンをそれぞれコーディング。
2. サブルーチンコールせずに、コピー＆ペースト。
3. 全体を見ずに機能追加を積み重ねる。（古い旅館）
4. 余りに短期開発なので、他社ＡＰＩをコピー＆ペースト。
5. 単体試験の残骸を間違って盛り込む。
6. 先輩のソースを真似たためのＣＣ。
7. そもそも似たような画面・帳票があることが原因。

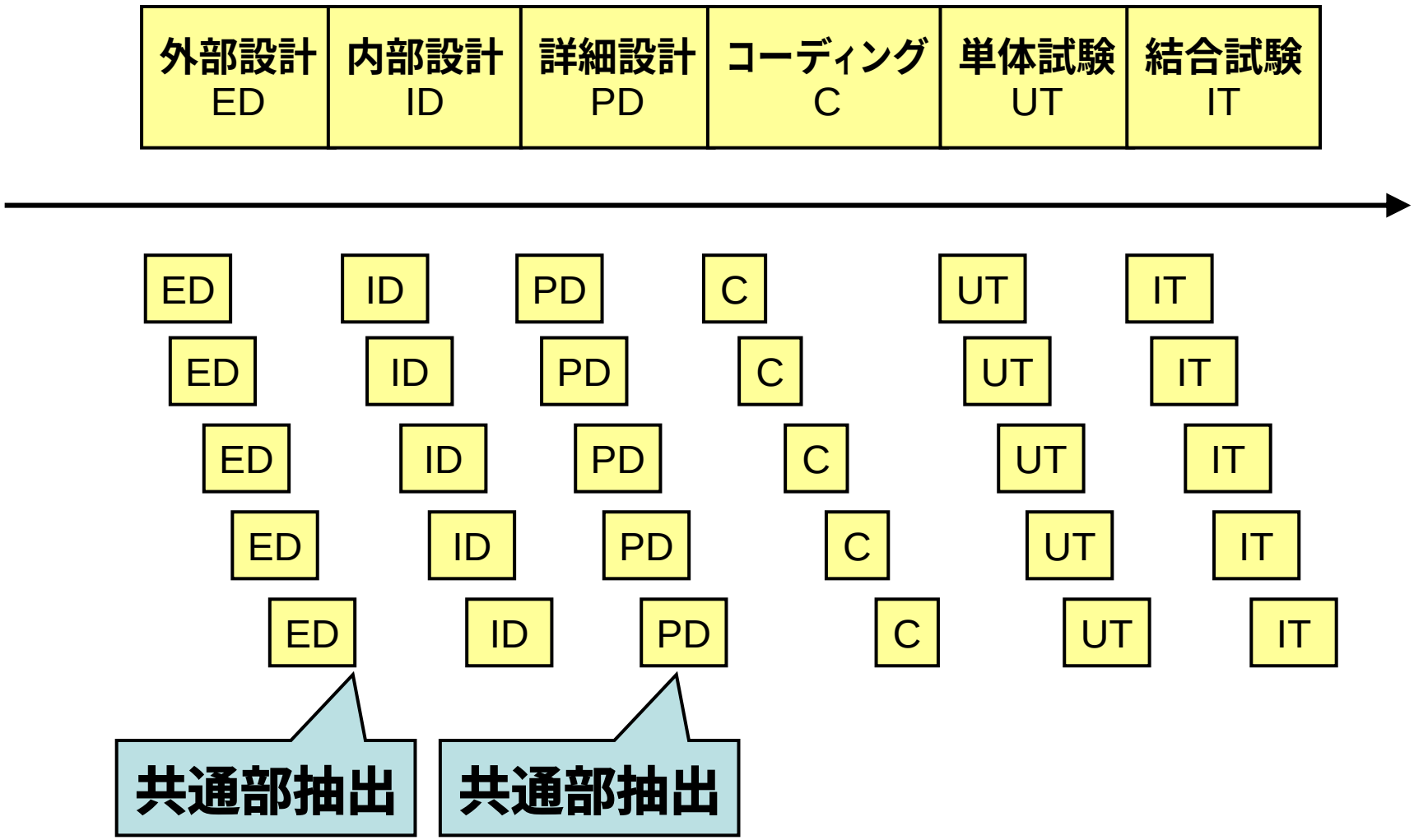
doyama.shinichi@nttcom.co.jp

特に Java に注目する理由

1. 外注率が高い。
2. 短い開発期間が要求され、複数社外注の傾向がある。
3. 新米プログラマが多い。
4. フレームワーク利用はCCを生じやすい。
5. 既存システムを Java で更改する案件が多い。
6. 既存 Java を書き直す案件も出始めている。
7. アジャイル開発でＣＣが生成される？
8. 分析ツールが出回っている。

昔の実装工程

ウォーターフォール

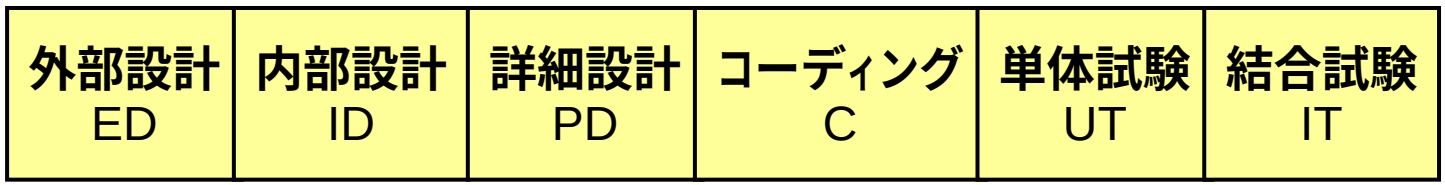


doyama.shinichi@nttcom.co.jp

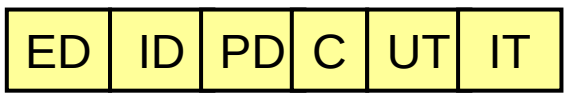
いまどきの実装

参考
http://www.bcm.co.jp/itxp/cat05/

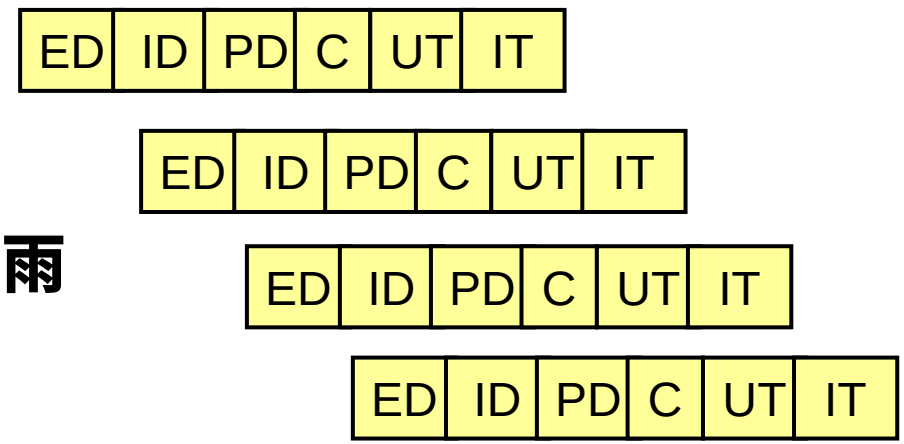
ウォーターフォール（昔の教科書）



プロトタイピング



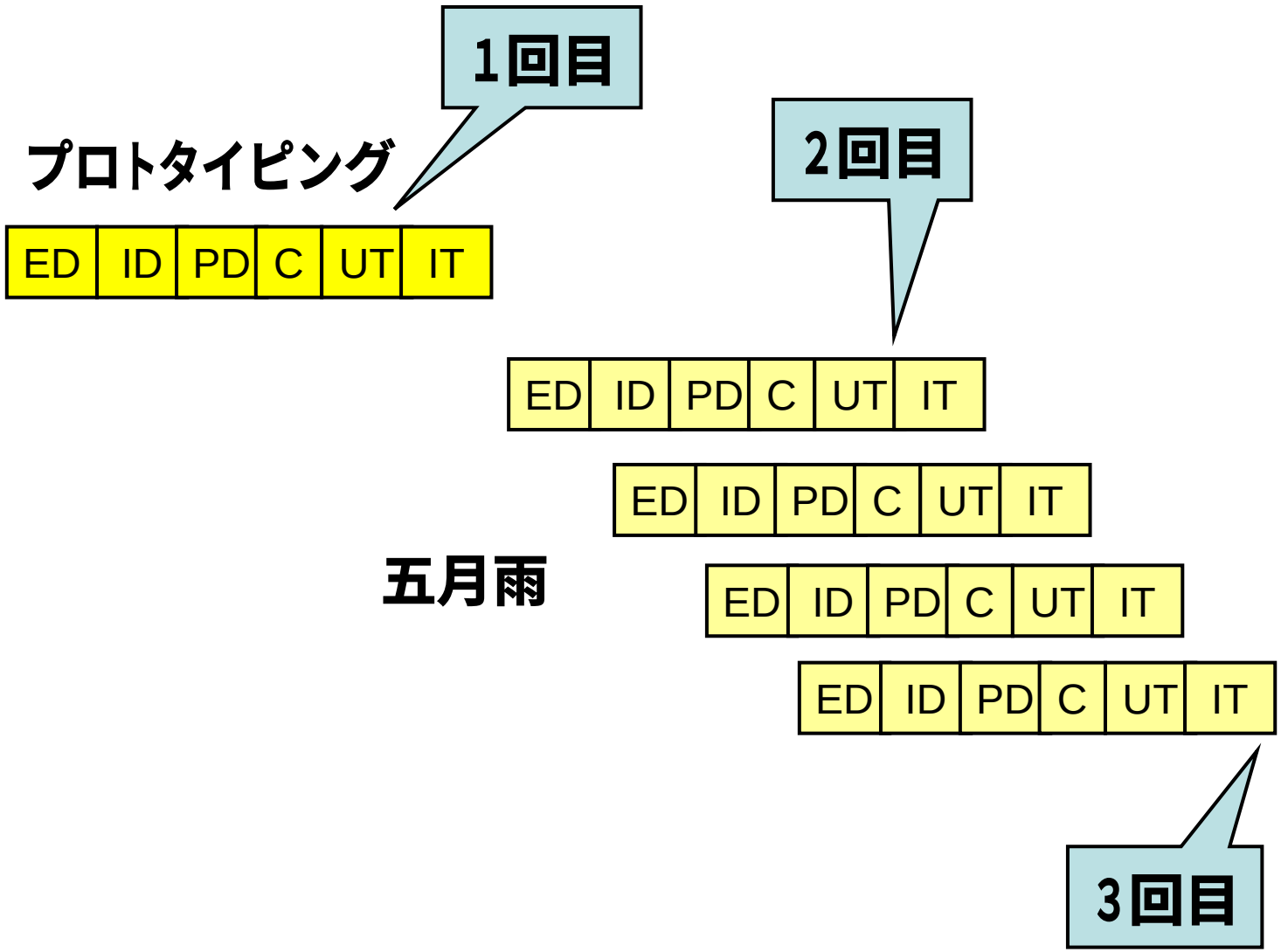
反復



五月雨

doyama.shinichi@nttcom.co.jp

ソース検査のタイミング



doyama.shinichi@nttcom.co.jp

ソース検査（1回目）

プロトタイプの主な目的

1. ＤＢ仕様の大枠は確定しているか検証。
2. 画面仕様の大枠は確定しているか検証。
3. フレームワークは確定しているか検証。
4. 性能目標をクリアできるアーキテクチャか検証。

ソース検査の主な目的

1. フレームワークは確定しているか検証。
2. コーディング規約を守っているかチェック。
3. スキルは十分かチェック。

doyama.shinichi@nttcom.co.jp

ソース検査（2回目）

プロトタイプの展開

各チームの代表が作ったプロトタイプを持ち帰り、
プロトタイプを参考にして、
各チームが担当する部分を実装してみる。

ソース検査の主な目的

1. プロトタイプの展開ができているかチェック。
2. 1回目の指摘を反映できているかチェック。

doyama.shinichi@nttcom.co.jp

ソース検査（3回目）

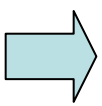
すべての実装を完了して最終試験

1. 実装漏れがないかチェック。
2. 1回目、2回目の指摘を反映できているかチェック。
3. 他にデグレードがないかチェック。

ＣＣは、すべて悪いのか

ソースを個別に見ていくと、

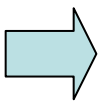
1. 簡単にはリファクタリングできない。
2. 画面や帳票がそもそもクローンになっている。
3. 短期実装にはコピーペーストは有効。
4. 各人が独自に考えて実装する方がかえって危険・・・



ＣＣと、どうつき合うか

ＣＣと、どうつき合うか

1. クローンペアをどう考えるか。
2. クローンペアをどう管理するか。



機能追加の追跡は、通常、構成管理で。

リファクタリングの追跡をＣＣだけでやるのは非効率。

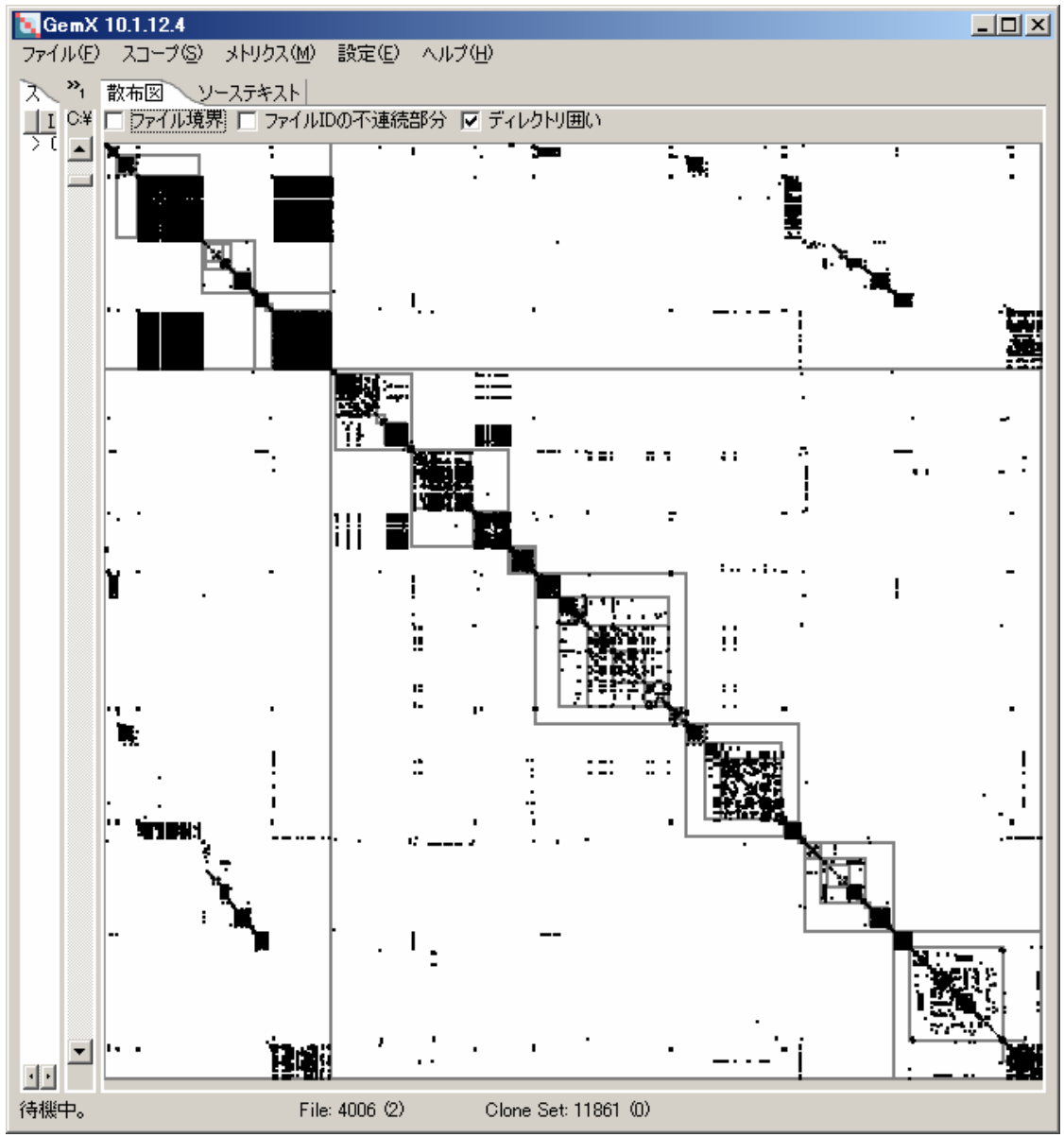
ライフサイクルの追跡を構成管理と統合すべき。

doyama.shinichi@nttcom.co.jp

5年間の変遷

クローンとなっているのは、

データ関連
帳票関連
共通ルーチンなど

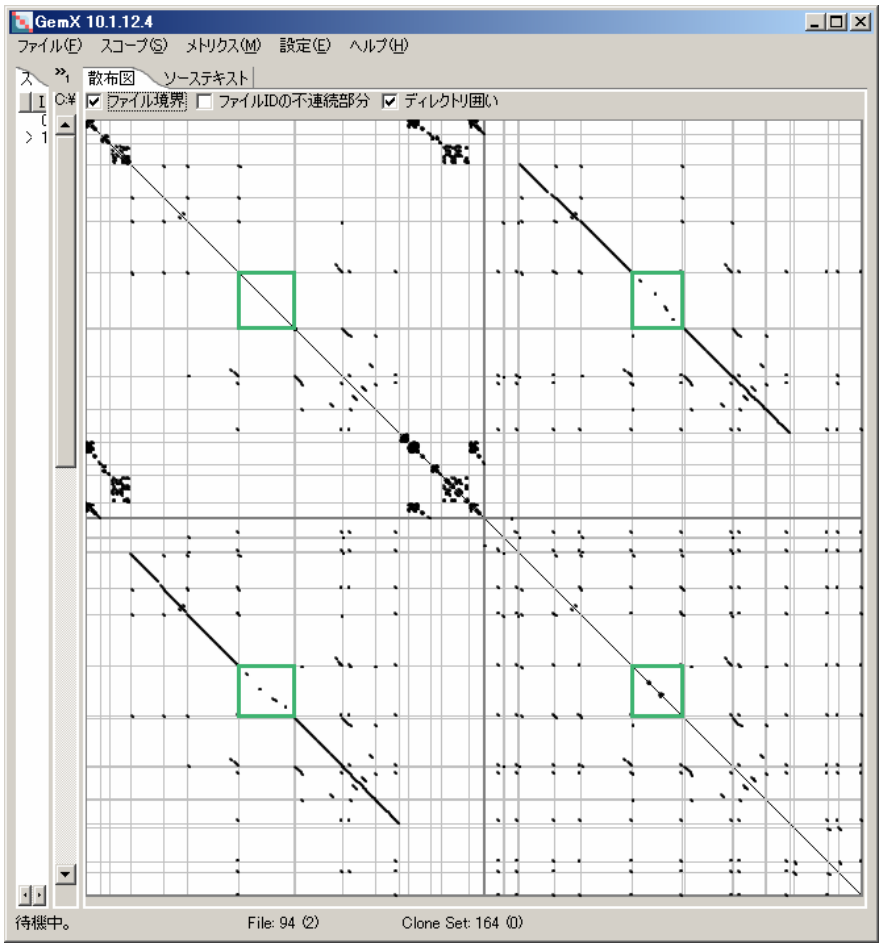


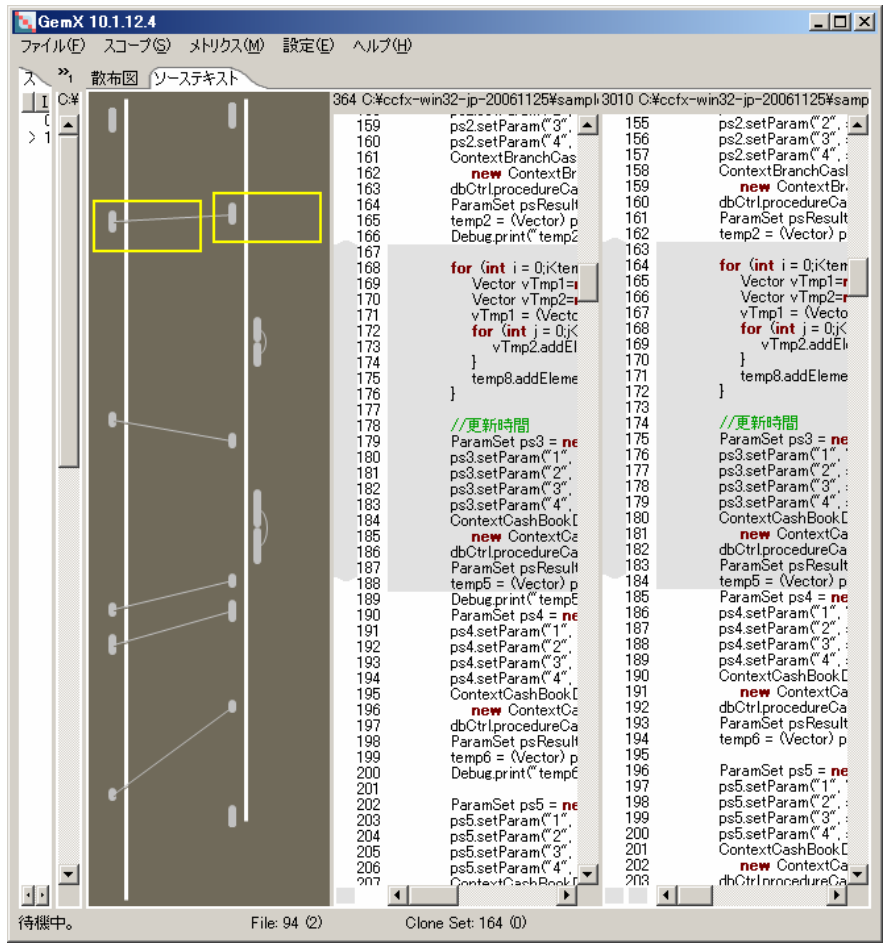
最小クローン長80トークン
最小トークンセットサイズ20
P-match利用

doyama.shinichi@nttcom.co.jp

5年間の変遷

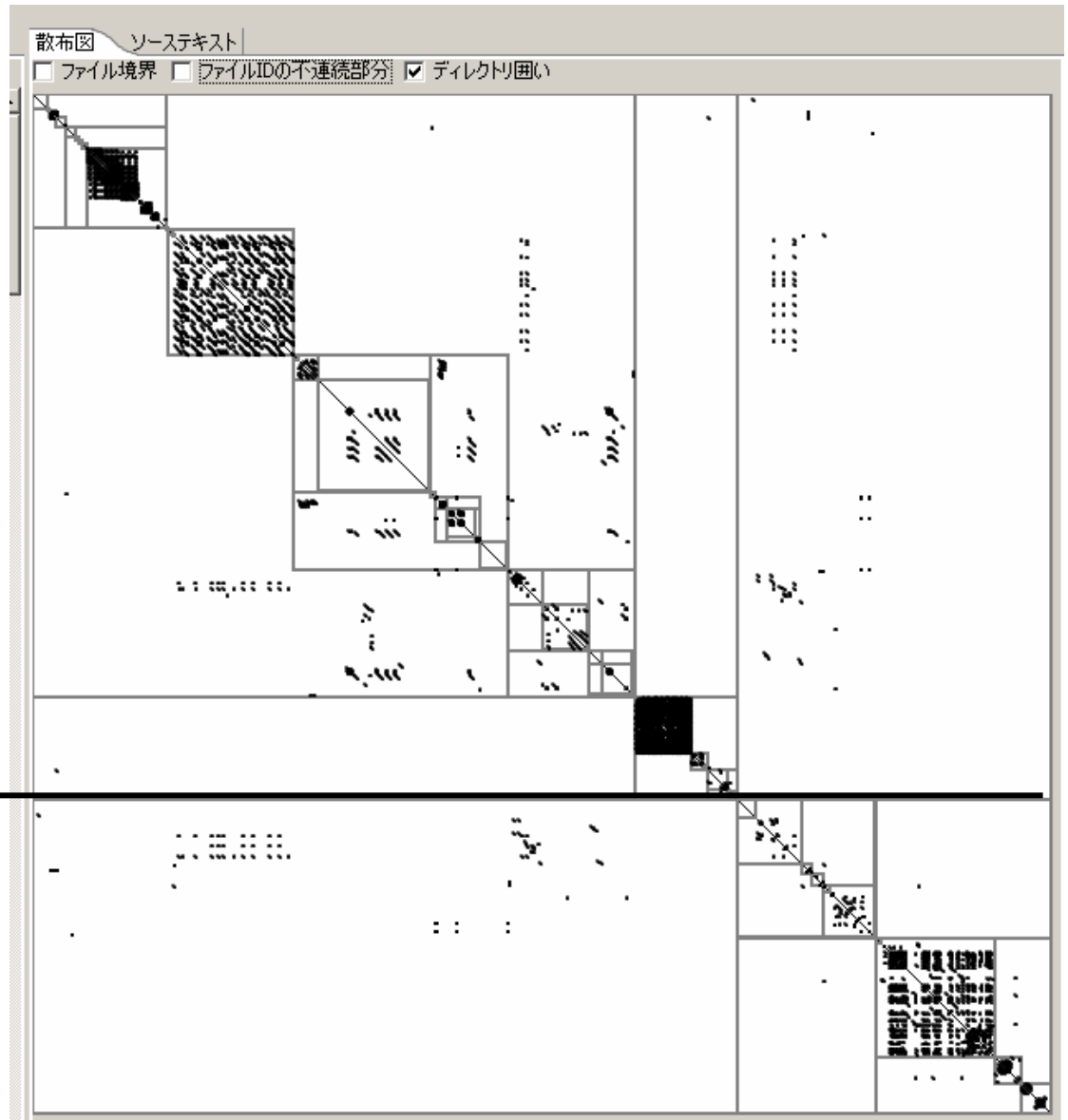
機能追加が各ファイルに霜降りに入っていく様子を、CCから見ると





doyama.shinichi@nttcom.co.jp

機能追加＋リファクタリング＋ディレクトリ整理



after

before

バージョン間比較

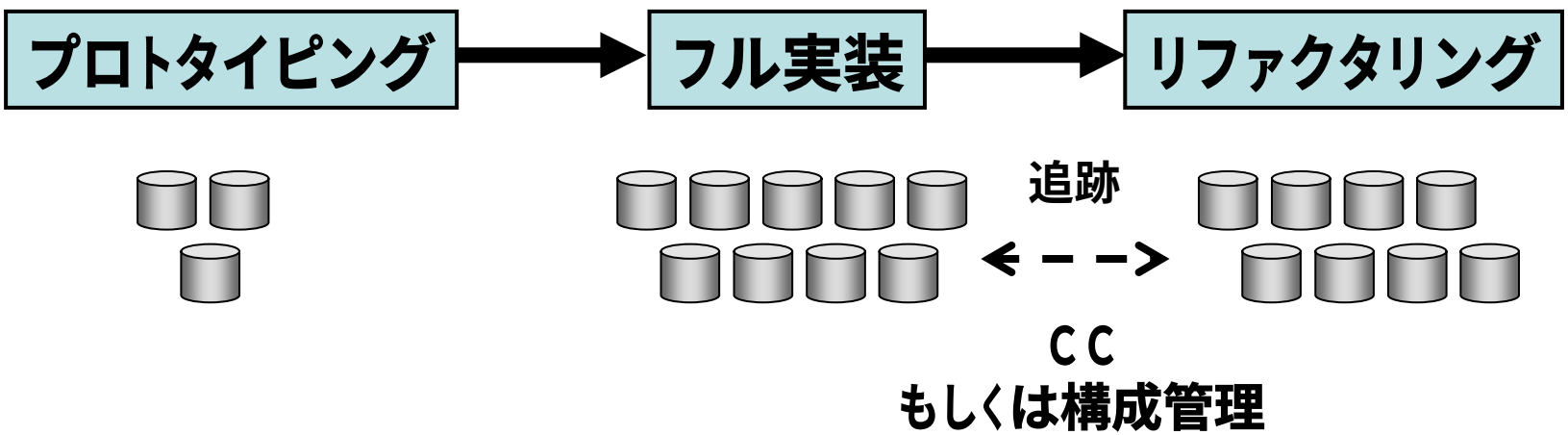
追跡!?

doyama.shinichi@nttcom.co.jp

クローンペア管理では限界

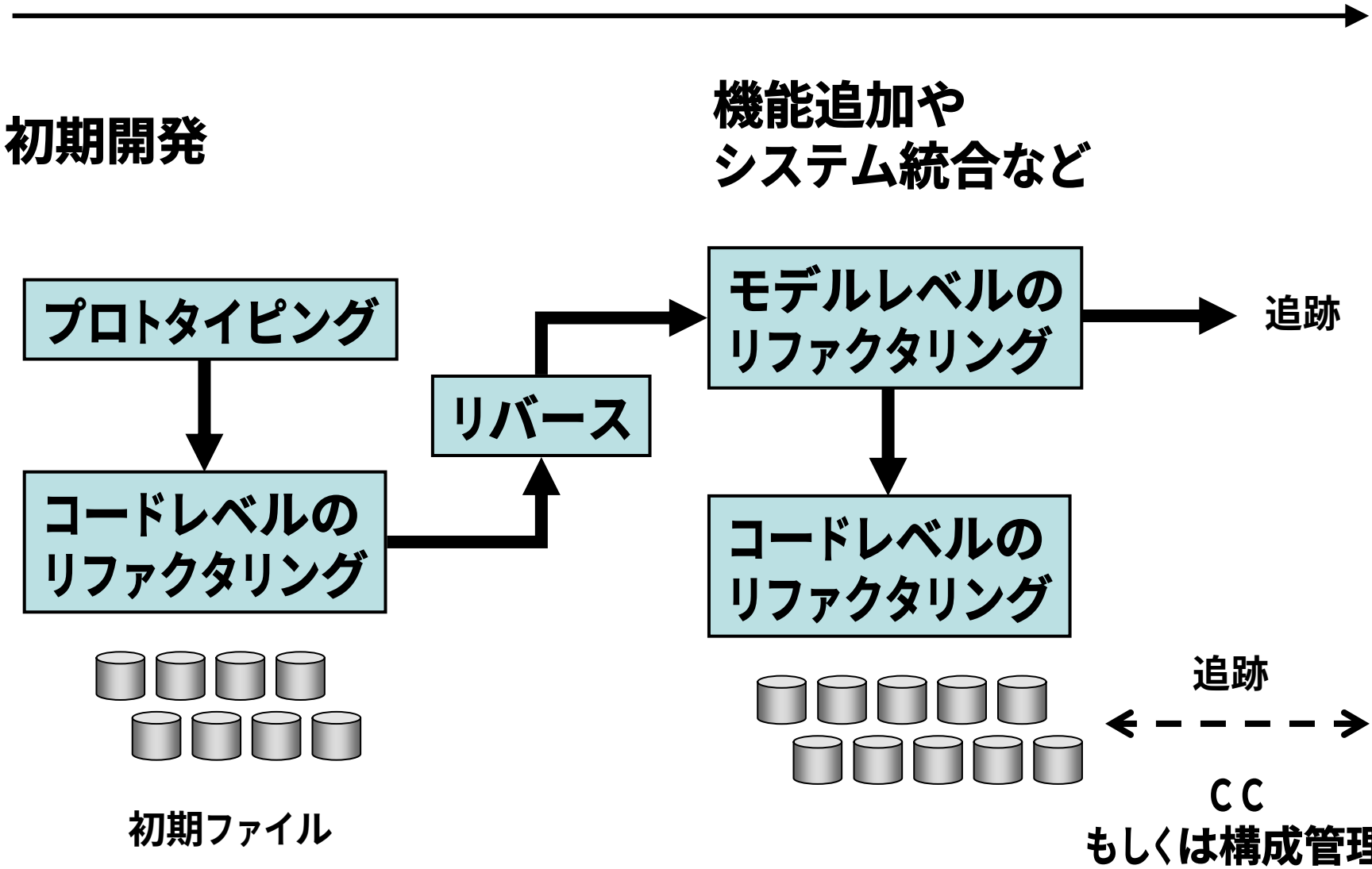
初期開発

ファイル名、クラス名、メソッド名も
変わるため、一般的な構成管理では
追跡できない。



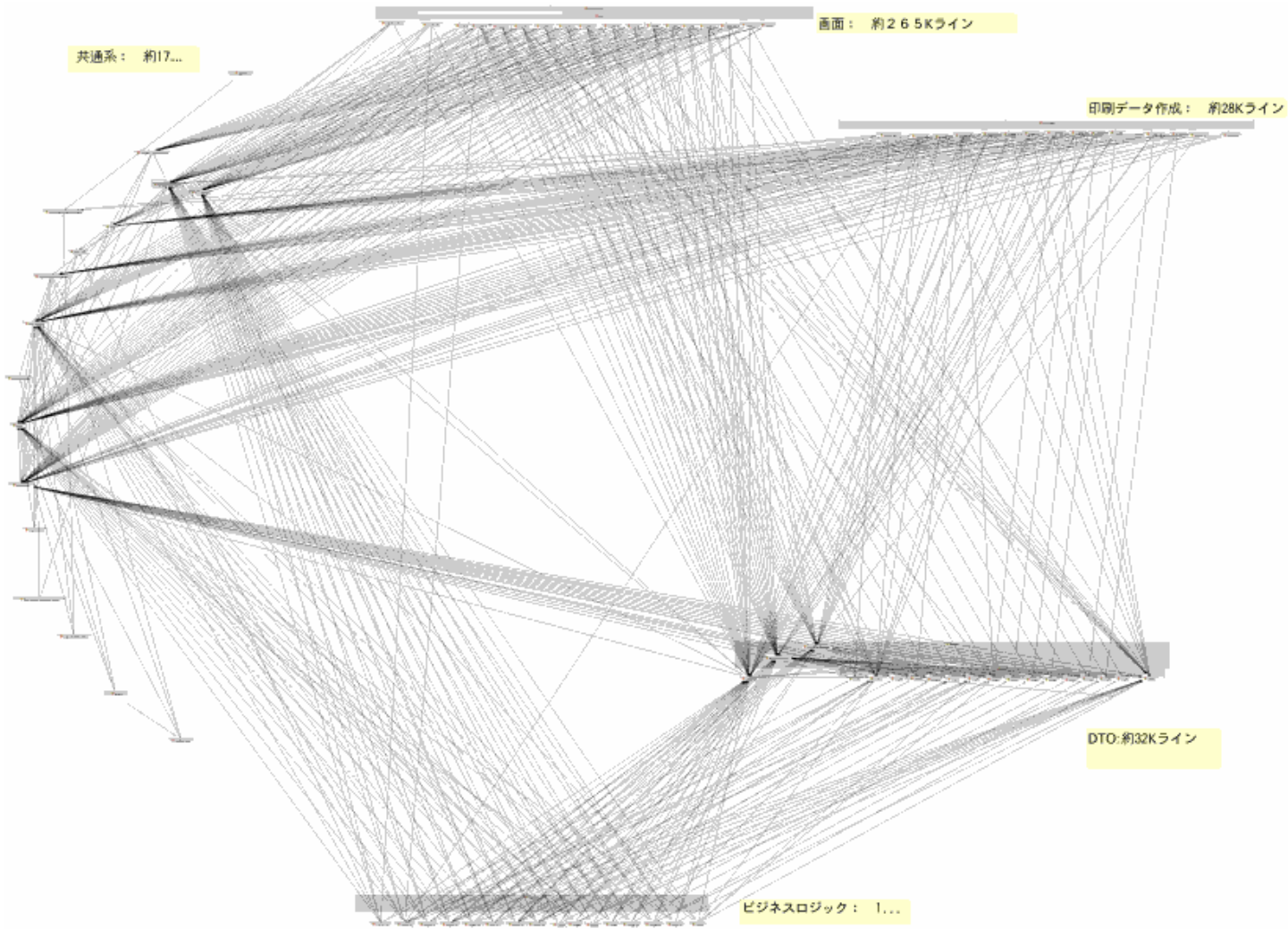
doyama.shinichi@nttcom.co.jp

ライフサイクルでの設計情報管理



doyama.shinichi@nttcom.co.jp

パッケージ間参照の例



doyama.shinichi@nttcom.co.jp

課題

1. 現状は、ミクロな品質、ミクロなドキュメント、ミクロな保守

   クローンペアのデグレード試験への応用

2. 全体像の把握（ＣＣ、モデル）
3. 機能追加の方針（どういう追加はどこにどうやって）
4. ライフサイクル・アーキテクチャのドキュメンテーション
5. Eclipse との統合で、さらに高度な操作性を。
6. 構成管理との統合