

テスト自動化を促進する 周辺技術とその運用法

コベリティジャパン株式会社



動的解析と静的解析

- 動的解析

- 利点

- 見つかった問題はすべて修正対象（誤検知はない）
 - パフォーマンスや発生するタイミングの解析が可能

- 欠点：

- 実際に動作させる必要がある = 解析自体に人的リソースが必要 = 自動化は困難
 - 問題が見つかってその原因の特定が困難な場合がある



- 静的解析

- 利点

- 全自動、実行環境は必要ない
 - 人的には困難な全制御パスのシミュレーションが可能
 - 人間が介在しないため客観的な品質データを取得可能

- 欠点

- 誤検知が発生する = ソースコードが存在しないライブラリなど
 - パフォーマンス、タイミングは解析できない

静的解析の種類

- コーディング規約チェック系
 - ベスト・プラクティスに基づいたコードの書き方
 - 標準規約など予め定義したコーディングルールに対し違反がないかを確認



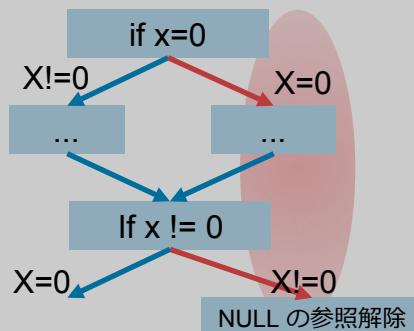
ランタイムエラー検出系

- バグ自体を検出
- 従来、実行時でないとは確認できなかった、メモリリーク、データ破壊、データ競合などを静的に検出
- コベリティの強さ
 - スピード（差分ビルド、差分解析）
 - 精度、根本原因の特定のし易さ
 - 検出した問題の管理（派生開発サポート、フローに基づくバグ管理）

静的解析 – 手戻りを激減させるテクノロジー

SAT ソルバ

不具合が発生しない
パスを排除し、
誤検知を減らす



全パス & プロシージャ間 解析

関数またはファイルに
跨る不具合の検出

```
void bar() {  
  foo(p);  
  if(p != 0) { ... }  
}
```



```
void foo(int *p) {  
  *p = 42;  
}
```

統計的な解析

プログラムの目的を
推測し、予期しない
動作を報告する

```
int *p = malloc(sizeof(int));  
if(p != 0) *p = 42;  
...  
int *p = malloc(sizeof(int));  
if(p != 0) *p = 42;
```

多分正しい

```
int *p =  
  malloc(sizeof(int));  
*p = 42;
```

多分間違い

ホワイトボックス ファザー

汚染データを投入し、
無害化されているかを
自動的に確認する

a b c d ... < > & ' ...



htmlEncode()



< > &
" '

例：バッファオーバーラン

3. 分岐条件 "`wc == 1`" は、true に分岐しました。

```
646         if (wc == 1) {
647             float fv;
648             TIFFGetField(tif, fip->field_tag, &fv);
```

シングルトン
の変数

Address渡し

4. address_of: "&fv" のアドレスはシングルトンポインタです。

❖ CID 12319 (#1/2): 範囲外のアクセス (ARRAY_VS_SINGLETON)

5. callee_ptr_arith: "&fv" がそれを配列として操作する関数 "TIFFWriteRationalArray(TIFF *, TIFFDirEntry *, float *)" に渡されています。これは、メモリの位置調整のミスや破壊の原因となる場合があります。[詳細を隠す]

```
649         if (!TIFFWriteRationalArray(tif, dir, &fv))
```

× /mnt/hgfs/VMShare/Coverity/samples/ids/6.5/tiff-3.9.1/libtiff/tif_dirwrite.c

```
989 TIFFWriteRationalArray(TIFF* tif, TIFFDirEntry* dir, float* v)
990 {
991     uint32 i;
992     uint32* t;
993     int status;
994
995     t = (uint32*) _TIFFmalloc(2 * dir->tdir_count * sizeof (uint32));
```

1. 分岐条件 "`t == NULL`" は、false に分岐しました。

```
996     if (t == NULL) {
997         TIFFErrorExt(tif->tif_clientdata, tif->tif_name,
998                     "No space to write RATIONAL array");
999         return (0);
1000     }
```

2. 分岐条件 "`i < dir->tdir_count`" は、true に分岐しました。

```
1001     for (i = 0; i < dir->tdir_count; i++) {
1002         float fv = v[i];
1003         int sign = 1;
```

3. ptr_arith: 式 "`v + i`" で "`v`" に対してポインタ演算が行われています。

呼出し先の
関数内表示

渡された変数
を配列として
アクセス

例：リソースリーク

呼出し先の関数内でアロケートされた領域を検知

```
5257 static int
5258 TIFFFetchStripThing(TIFF* tif, TIFFDirEntry* dir, uint32 nstrips, uint64** lpp)
5259 {
5260     static const char module[] = "TIFFFetchStripThing";
5261     enum TIFFReadDirEntryErr err;
5262     uint64* data;
5263     err=TIFFReadDirEntryLong8Array(tif, dir,&data);
5264     if (err!=TIFFReadDirEntryErrOk)
5265     {
5266         TIFFReadDirEntryOutputErr(tif, err, module, TIFFFieldWithTag(tif, dir->tdir_tag)->field_name, 0);
5267         return(0);
5268     }
5269     if (dir->tdir_count!=(uint64)nstrips)
5270     {
5271         uint64* resizeddata;
5272         resizeddata=(uint64*)_TIFFCheckMalloc(tif, nstrips, sizeof(uint64), "for strip array");
5273         if (resizeddata==0)
5274         {
5275             return(0);
5276             if (dir->tdir_count<(uint64)nstrips)
5277             {
5278                 _TIFFmemcpy(resizeddata, data, (uint32)dir->tdir_count*sizeof(uint64));
5279                 _TIFFmemset(resizeddata+(uint32)dir->tdir_count, 0, (nstrips-(uint32)dir->tdir_count)*sizeof(uint64));
5280             }
5281             else
5282                 _TIFFmemcpy(resizeddata, data, nstrips*sizeof(uint64));
5283             _TIFFfree(data);
5284             data=resizeddata;
5285         }
5286     }
5287     *lpp=data;
5288 }
```

1. alloc_arg: "TIFFReadDirEntryLong8Array(TIFF *, TIFFDirEntry *, uint64 **)" は、"data" に格納したメモリをアロケートしています。[詳細の表示]

2. 分岐条件 "err != TIFFReadDirEntryErrOk" は、false に分岐しました。

3. 分岐条件 "dir->tdir_count != (uint64)nstrips" は、true に分岐しました。

4. 分岐条件 "resizeddata == NULL" は、true に分岐しました。

❖ CID 12334 (#1/1): リソースリーク (RESOURCE_LEAK)
5. leaked_storage: 変数 "data" がスコープを外れるため、リソースがリークしています。

領域をを解放せずにローカル変数を破棄

例：コピペミス

変数名の書き換えミス

関数外でも、検出可能

original: "personAType" はコピー元のように見えます。

◆ CID 10064 (#1/1): コピーペースト エラー (COPY_PASTE_ERROR)

copy_paste_error: "personAType" の "personAType" はコピーアンドペーストのミスのように見えます。"personBType" にすべきではありませんか？

```
58
59     Person p = r.getPersonA();
60     personA = p.getPersonName().toString();
61     personAId = p.getPersonId();
62     try {
63         personAType = p.isPatient() ? "Patient" : "User";
64     }
65     catch (Exception ex) {
66         personAType = "User";
67     }
68
69     p = r.getPersonB();
70     personB = p.getPersonName().toString();
71     personBId = p.getPersonId();
72     try {
73         personBType = p.isPatient() ? "Patient" : "User";
74     }
75     catch (Exception ex) {
76         personAType = "User";
77     }
78
```

例：クロスサイトスクリプティング

71 @RequestMapping("/admin/locations/locationTagAdd")

フレームワークのエントリーポイントとしてこの関数に入ります。サーブレットリクエストからのデータであるため、パラメータ "name" は汚染されています。

パラメータ "name" は汚染されたデータを受取ります。

72 ① public String add(@RequestParam("name") String name, @Request

73

74 if (!StringUtils.hasText(name)) {

75 request.setAttribute(WebConstants.OPENMRS_

76

発生 1/4 ~ openmrs-trunk

汚染された入力源から、不十分にエスケープされた汚染データを使用する文字列作成、次に出力へのデータの流れ:

◆ tainted_source LocationTagController.java:72

◆ tainted_path_param LocationTagController.java:72

◆ tainted_path_arg LocationTagController.java:81

入入口 (source)

パラメータ "name" は汚染されたデータを受取ります。

46 ② public LocationTag(String name, String description) {

org.openmrs.BaseOpenmrsMetadata.setName(java.lang.String) に汚染されたデータが渡されています: "name"

47 setName(name);

48 setDescription(description);

49 }

50

◆ tainted_path_param LocationTagController.java:72

◆ tainted_path_arg LocationTagController.java:81

◆ tainted_path_param LocationTag.java:46

◆ tainted_path_arg LocationTag.java:47

◆ tainted_path_param BaseOpenmrsMetadata.java:71

◆ tainted_path BaseOpenmrsMetadata.java:72

経路

CID 11758 (#1-2/4): クロスサイト スクリプト (XSS) インジェクション (XSS) コンテキスト HTML_ATTR_VAL_DQ のために適切にサニタイズされていないため、"\${msgArgs}" の HTML ページへの追加はクロスサイトスクリプティングを招く恐れがあります。

EL におけるクロスサイトスクリプティングへの対処策: 検出されたコンテキストのためにエスケープする必要があります。例えば、

fn:escapeXml(msgArgs)

ここで fn:escapeXml は汚染データをエスケープする関数 (HTML attribute 用)。
[詳細情報]

91 <div id="openmrs_msg"><or

92 </c:if>

93 <c:if test="\${err != null}">

94 <div id="openmrs_error">

適用して次へ 適用 元に戻す

サニタイズすべき箇所

◆ tainted_path_call headerMinimal.jsp:8

◆ tainted_path_call headerMinimal.jsp:8

◆ tainted_path_call headerMinimal.jsp:9

◆ tainted_path_call headerMinimal.jsp:10

◆ xss_injection_site headerMinimal.jsp:91

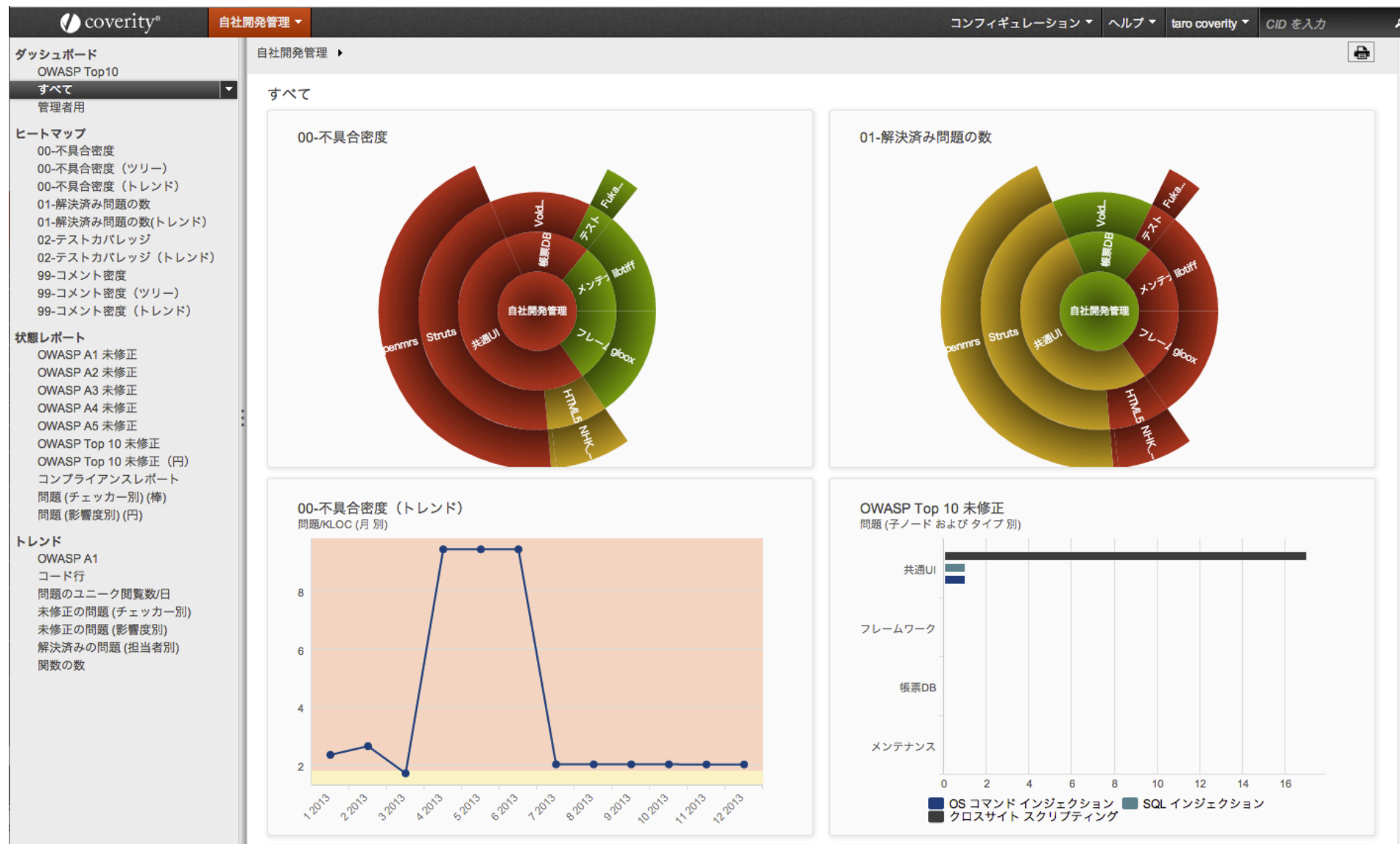
発生 3/4 ~ openmrs-1.2

汚染された入力源から、不十分にエスケープされた汚染データを使用する文字列作成、次に出力への

修正方法の提示

検出した問題をトラッキングする

開発組織、プロジェクト、サプライヤ



Coverity を取り入れた開発者ワークフロー



SCAN: オープン・ソースの状況



[Scan Home](#) [FAQ](#) [Scan News](#) [Projects Using Scan](#) [About](#)

[Sign Up](#)

[Sign In](#)

COVERITY SCAN STATIC ANALYSIS

Find and fix defects in your C/C++ or
Java open source project for *free*.

Test every line of code and potential execution path.

The root cause of each defect is clearly explained,
making it easy to fix bugs

Integrated with  

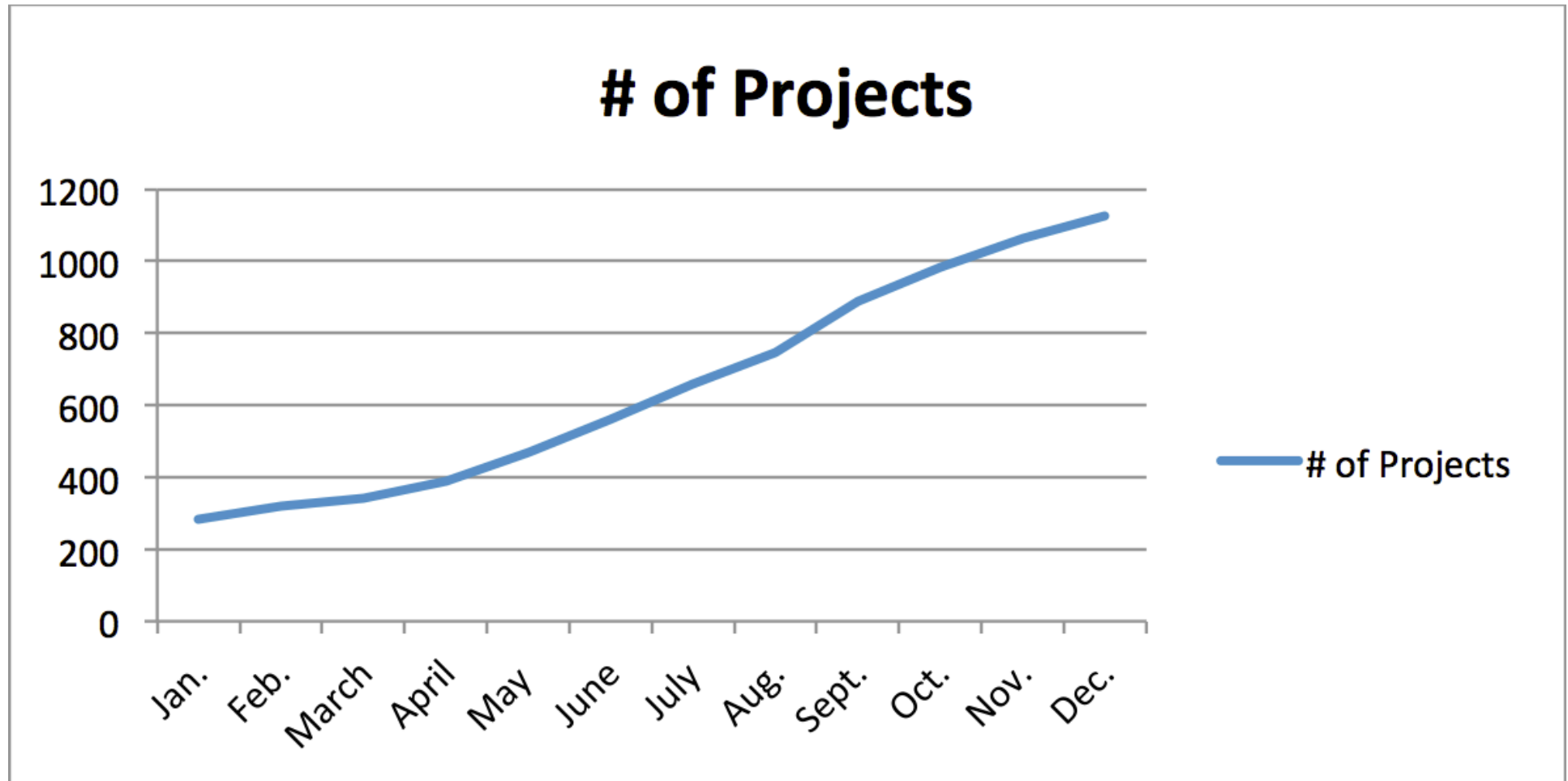
[Sign Up For Free](#)

```
188
2. alloc_fn: Storage is returned from allocation function "vmalloc(unsigned long)".
3. var_assign: Assigning: "sysram" = storage returned from "vmalloc(size)".
189     sysram = vmalloc(size);
4. Condition "!sysram", taking false branch
190     if (!sysram)
191         return -ENOMEM;
192
193     info = framebuffer_alloc(0, device);
5. Condition "info == NULL", taking true branch
194     if (info == NULL)
195         return -ENOMEM;
```

◆ CID 703417 (#1 of 1): Resource leak (RESOURCE_LEAK)
6. leaked_storage: Variable "sysram" going out of scope leaks the storage it points to.

Feedback

2013年に利用者が急増



オープン・ソースの規模感

Table 1: 2013 OSS コードの規模感: C/C++			
サイズ (行数)	2013 プロジェクト 数	2013 プロジェクトの 割合	2012 プロジェクトの 割合
100,000 以下	358	48%	38%
100,000- 499,999	299	40%	44%
500,000- 100 万行	42	6%	7%
100 万行以上	42	6%	11%
計	741		

検出されたバグの密度平均（年別）

Table 2: 2008-2013 平均不具合検出率	
Coverity Scan Report 年	平均不具合密度 C/C++ （件/KLOC）
2008	.30
2009	.25
2010	.81
2011	.45
2012	.69
2013	.59

Coverity によって自動検出されたバグのうち、影響度が高・中レベルのもの

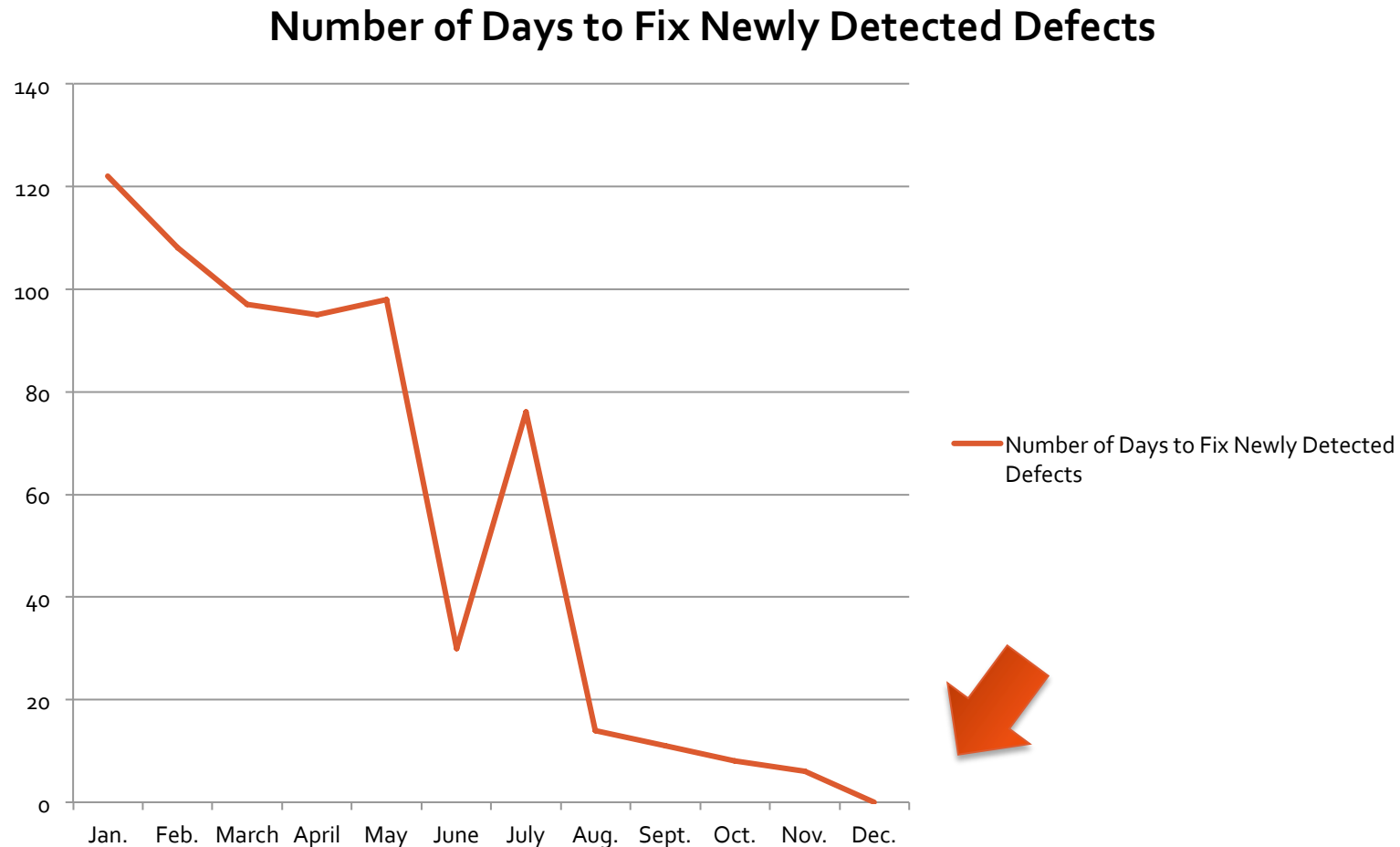
Linux の現在

Table 8: LINUX ANALYSIS: 2006-2013				
年	バージョン	解析行数	新規不具合数	修正数
2006	2.6.16	3,451,730	1,264	435
2007	2.6.16	3,458,369	425	217
2008	2.6.27	4,202,209	596	365
2009	2.6.32	4,862,567	527	417
2010	2.6.33, 2.6.34, 2.6.35	5,504,780	3,858	462
2011	2.6.38, 2.6.39, 3.0, 3.1	6,849,378	2,331	1,283
2012	3.2, 3.3, 3.4, 3.5, 3.6, 3.7	7,387,908	5,803	5,170
2013	3.12	8,578,254	3,299	3,346

Linuxコンポーネント別不具合密度

TABLE 9: 2013 LINUX 不具合密度 (コンポーネント別)	
コンポーネント	Linux 2013
Drivers	0.58
Drivers-Media	0.47
Drivers-SCSI	0.67
Staging-lustre	0.85
Drivers-Staging	0.77
Networking	0.50
Drivers-USB	0.49
Kernel	0.61
Drivers-Infiniband	0.65
Drivers-USB	0.49
Kernel	0.61
Drivers-Infiniband	0.65

Linux : 不具合修正の姿勢が変わった！



2013 年から Dave Jones が Linux の Coverity SCAN 担当に

今日は Coverity 開発チームの内情を...

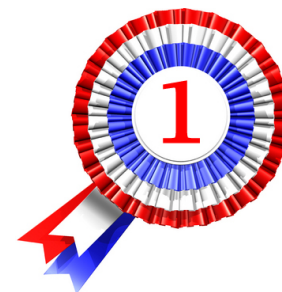


Transforming Testing Through Automation

Coverity's Journey to Automate Testing
as Part of Product Development

Andreas Kuehlmann
Coverity, Inc.
San Francisco, CA, USA
akuehlmann@coverity.com

テスト自動化 5つのチェックポイント



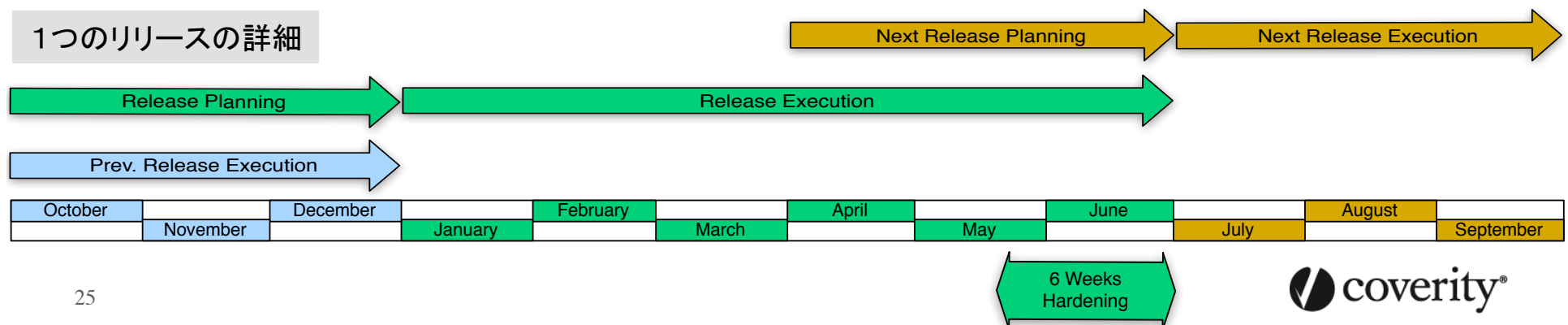
コベリティの製品リリースプロセス

一般的なリリースプロセス:

- 6ヶ月のリリースサイクル
- 3ヶ月の次期リリース計画
- リリース直前の6週間の“ハードニング”サイクル
 - 2週間の“フィーチャーフリーズ”
 - 2週間の“インテグレーションフリーズ”
 - 2週間の“リリースフリーズ”とGAのためのアップロード
- リリースのコードネームはカリフォルニア州の都市名
 - Alameda, Berkeley, Davis, Eureka, Fresno,

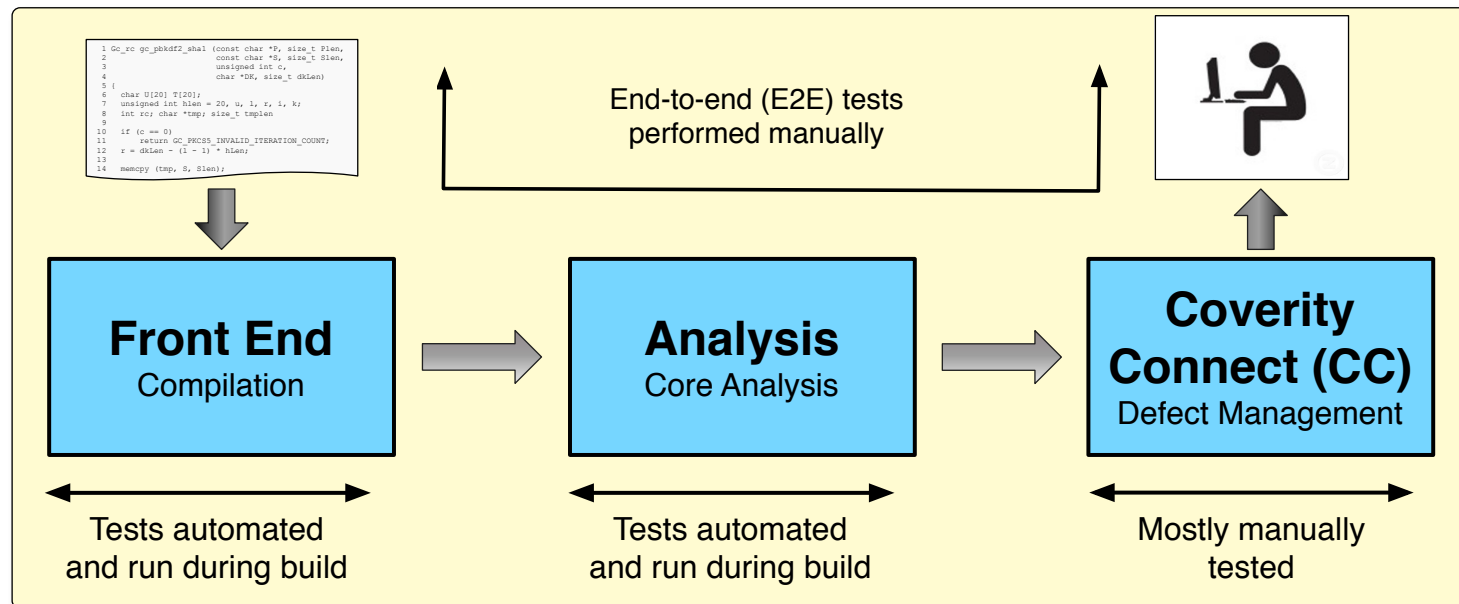
リリースを実行するプロセス:

- アジャイルプロセス
- 部門横断的なスクラムチーム
- 2週間のスプリント
- メールまたは対面での日次スタンド・アップ
- Jira と PivotalTracker を介した進捗管理
- Bugzilla によるバグ管理
- Jenkins-CI による継続的統合
- 週1回のステータスミーティング
 - リリースハードニングでは、週に3回



2年前の状況

コベリティの製品構造概要:



2年前のテストアプローチ:

- フロントエンドと解析のコンポーネントは自動化されたテストでテストされていた
- Coverity Connect はほとんどが手動でテスト
- End-to-End のテストは手動で行われていた

2年前の状況

- QA プロセスは東ヨーロッパでアウトソースし実施していた
 - 伝統的な QA スキル、開発スキルはわずかかもしくは皆無
 - 10時間の時差
 - 言語的な壁
- 6 週間のハードニングサイクルは “ロシアン・ルーレット” のようなものだった
 - “フィーチャー実装完” と “フィーチャーテスト” の時間的解離
 - 開発者は品質に対して、即座の直接的な責任を感じていなかった
 - 何に出くわすかわからない – “簡単なバグ” なのか “致命的バグ” なのか
 - とるに足らないバグに QA サイクルの多くが費やされていた
 - 手動のプロセスでは、テストを 2-3 回しか回せない
- 製品品質による苦しみ
 - よい品質でリリースするために、チームは多くのストレスを抱えていた



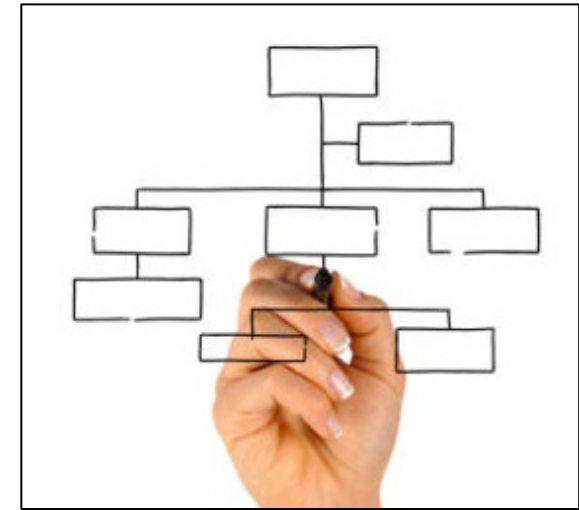
No.5：本当に自動化が効くのか？



- 6ヶ月のメジャーリリースサイクル
- サポートプラットフォームが非常に多い
 - プラットフォームの違いのみのテストケースが多い
 - Jenkins 上のビルドは各プラットフォームに分散

矯正への最初の試み

- QAと開発の間の境界を取り除く
 - 組織的 + 時間的 (開発とテストを織り込んでいく)
- 2011 初頭に組織を再編
 - 2/3 の QA リソースを開発組織に移管
 - 目的を、テストの自動化を名目に、
“開発しながらテストする” ように設定

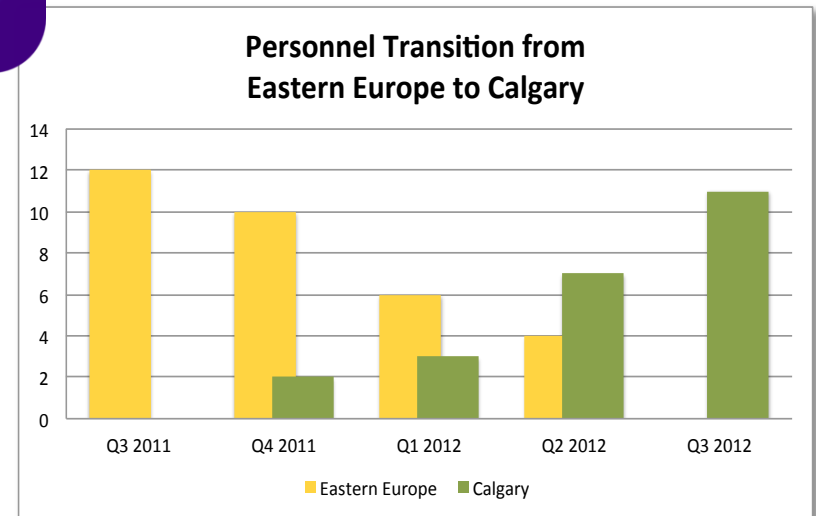


- しかし、結果は期待はずれだった
 - QA チームはプログラミングに関してトレーニングされておらず、自動テストを開発するのが困難であった
 - テストの自動化に関しては、実際ごくわずかしか進歩しなかった
 - 時差と言語の壁が存続していた

No.4:スキルセットが正しいか？

- リソースを東ヨーロッパからカナダ、カルガリーに移動
 - 1 時間の時差
 - サンフランシスコから 2.5 時間のフライト
 - 言語の壁がない
- 2011 年の夏に移行を開始
 - 移行は12ヶ月以上かかった
 - 多くは予算に影響を与えなかった
- スキルセットの変更
 - テスト自動化に焦点をあてたプログラミングスキル
 - テストの自動化に対する高いモチベーション

4



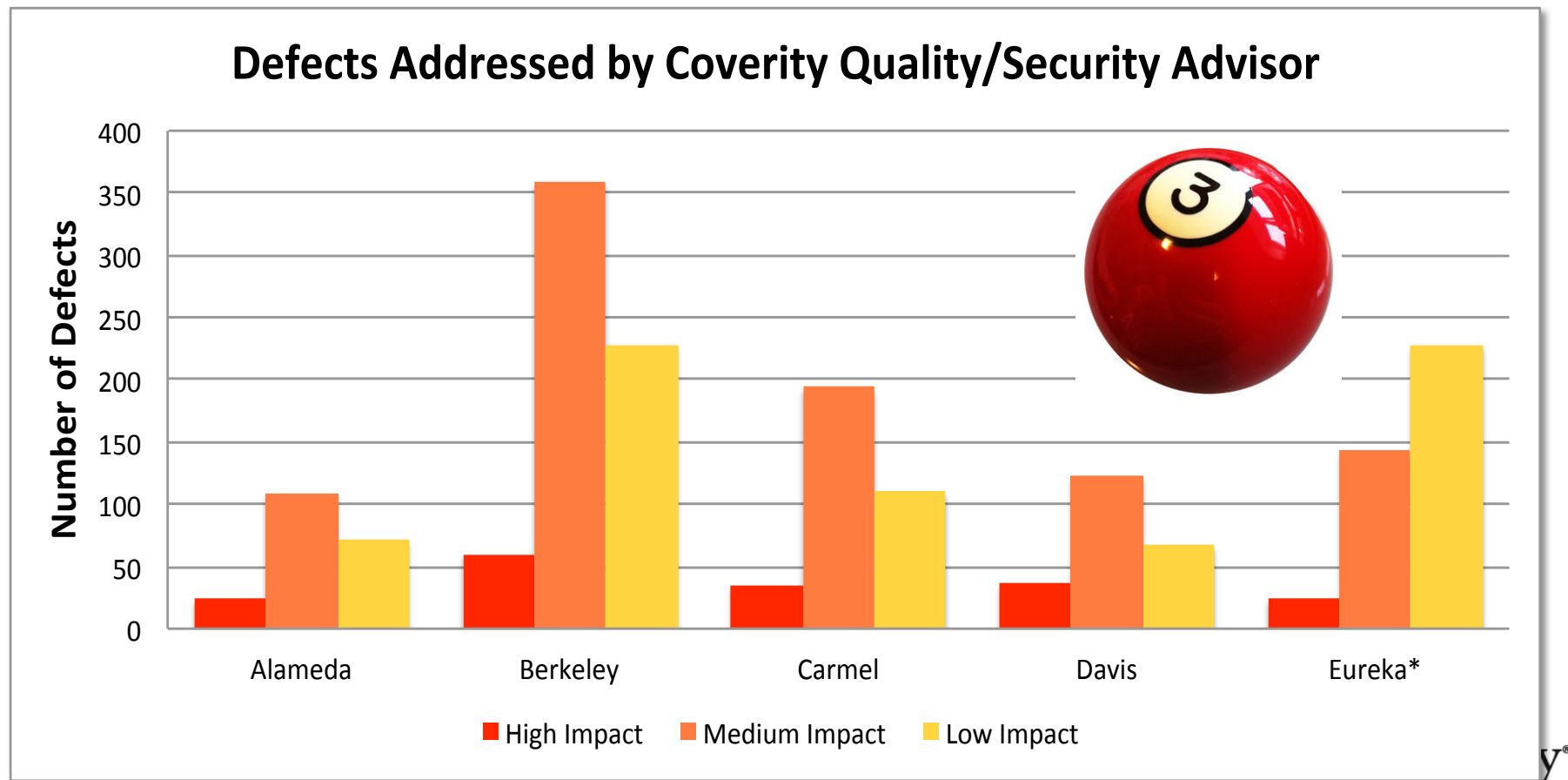
モチベーションの変化

- テストの自動化に集中する。
 - とりわけ、Coverity Connect (CC) と End-to-End (E2E)
- 自動化されたテストにより 2 つの効果:
 - フィーチャーに期待する挙動を成文化:
 - テストと実装の間の“ネゴシエーション”
 - テストがパスすれば、フィーチャーは“完了”と宣言できる
 - 機能が完成したあとのリグレッションの検知:
 - 次のフィーチャーセットの作業をしている間の意図しない機能の変更
 - 将来のフィーチャー開発のための“ガードレール”
- ただし、全てのテストがコードに密に連携して開発できるわけではない
 - テストのためのよいアーキテクチャが必要となる

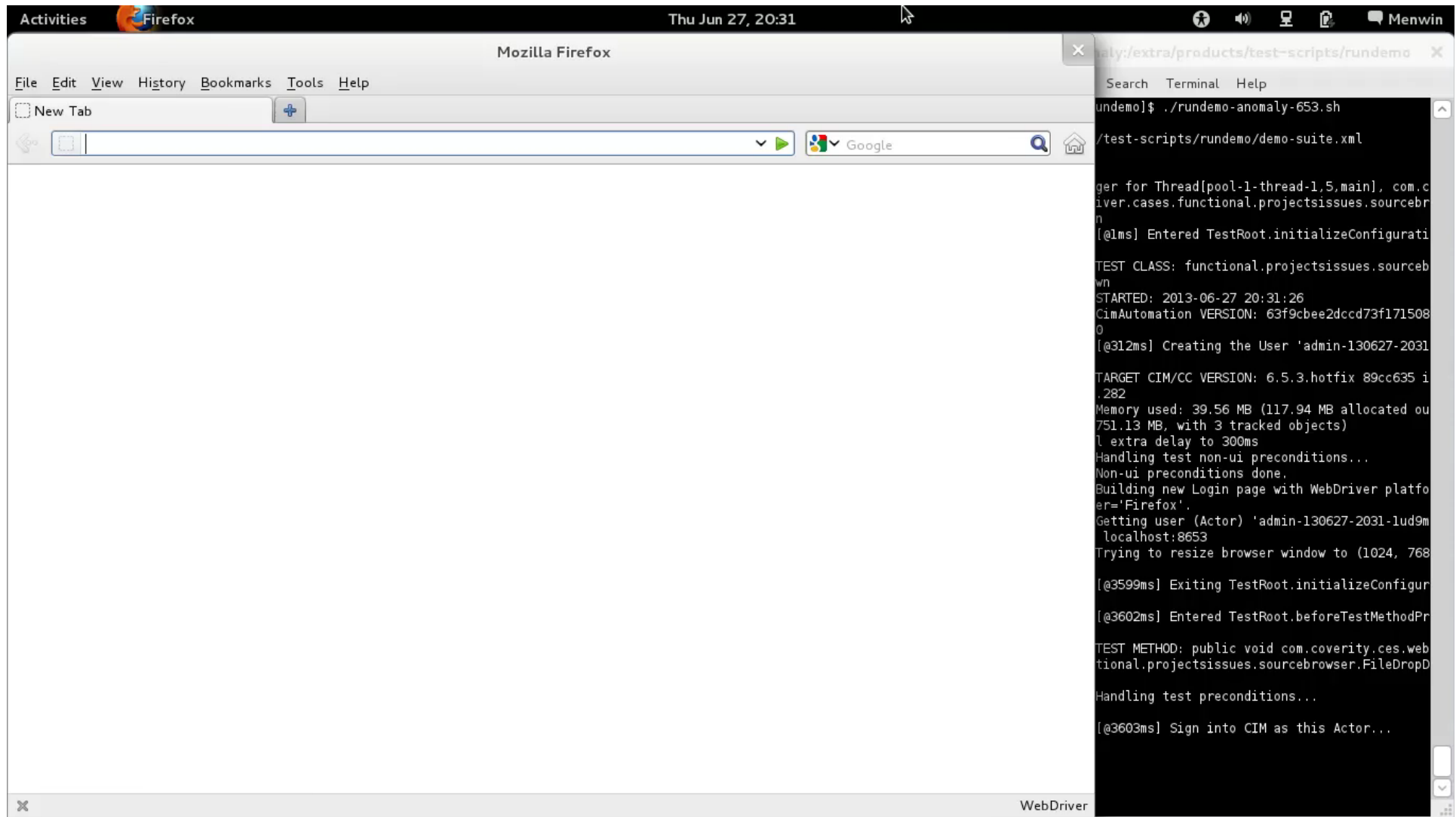


No.3: クリーンなコードか？

- もちろん、自社製品を開発にも自社製品を使用
- クリーンなコードでなければ、自動化も進まない



Coverity Connect の自動テストの例



The screenshot shows a Mozilla Firefox browser window. The address bar is empty, and the page content is blank. A terminal window is overlaid on the right side of the browser, displaying the output of a test script. The terminal text includes:

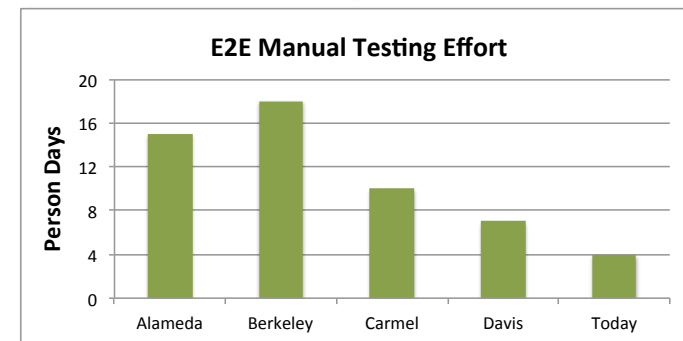
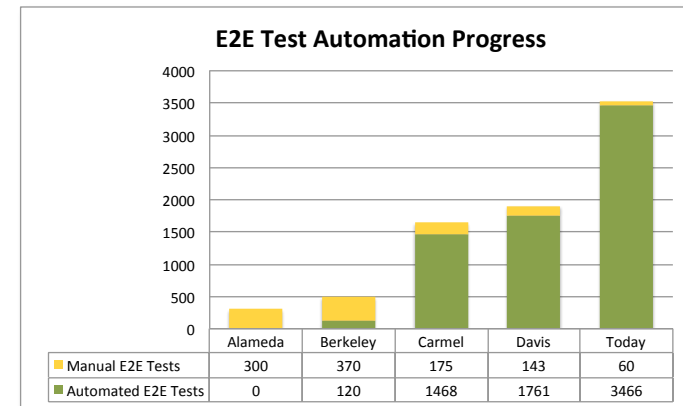
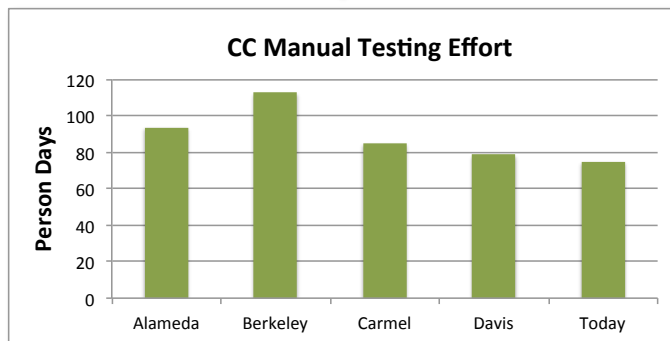
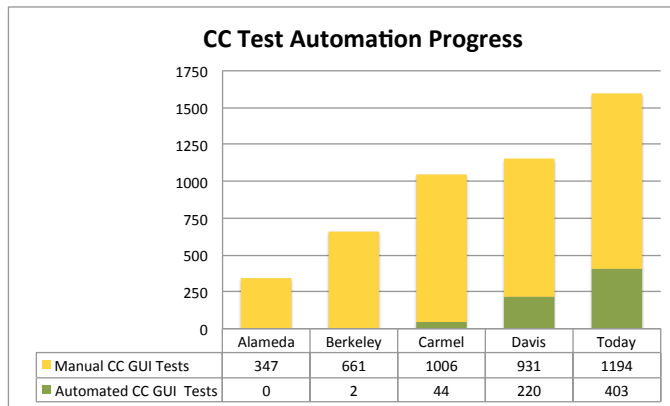
```
undemo]$ ./rundemo-anomaly-653.sh
/test-scripts/rundemo/demo-suite.xml

ger for Thread[pool-1-thread-1,5,main], com.c
iver.cases.functional.projectsissues.sourcebr
n
[@1ms] Entered TestRoot.initializeConfigurati
TEST CLASS: functional.projectsissues.sourceb
wn
STARTED: 2013-06-27 20:31:26
CimAutomation VERSION: 63f9cbee2dcd73f171508
0
[@312ms] Creating the User 'admin-130627-2031
TARGET CIM/CC VERSION: 6.5.3.hotfix 89cc635 i
.282
Memory used: 39.56 MB (117.94 MB allocated ou
751.13 MB, with 3 tracked objects)
l extra delay to 300ms
Handling test non-ui preconditions...
Non-ui preconditions done.
Building new Login page with WebDriver platfo
er='Firefox'.
Getting user (Actor) 'admin-130627-2031-lud9m
localhost:8653
Trying to resize browser window to (1024, 768
[@3599ms] Exiting TestRoot.initializeConfigur
[@3602ms] Entered TestRoot.beforeTestMethodPr
TEST METHOD: public void com.coverity.ces.web
tional.projectsissues.sourcebrowser.FileDropD
Handling test preconditions...
[@3603ms] Sign into CIM as this Actor...
```

The terminal window also shows the command `Search Terminal Help` and the file path `aly:/extra/products/test-scripts/rundemo`.

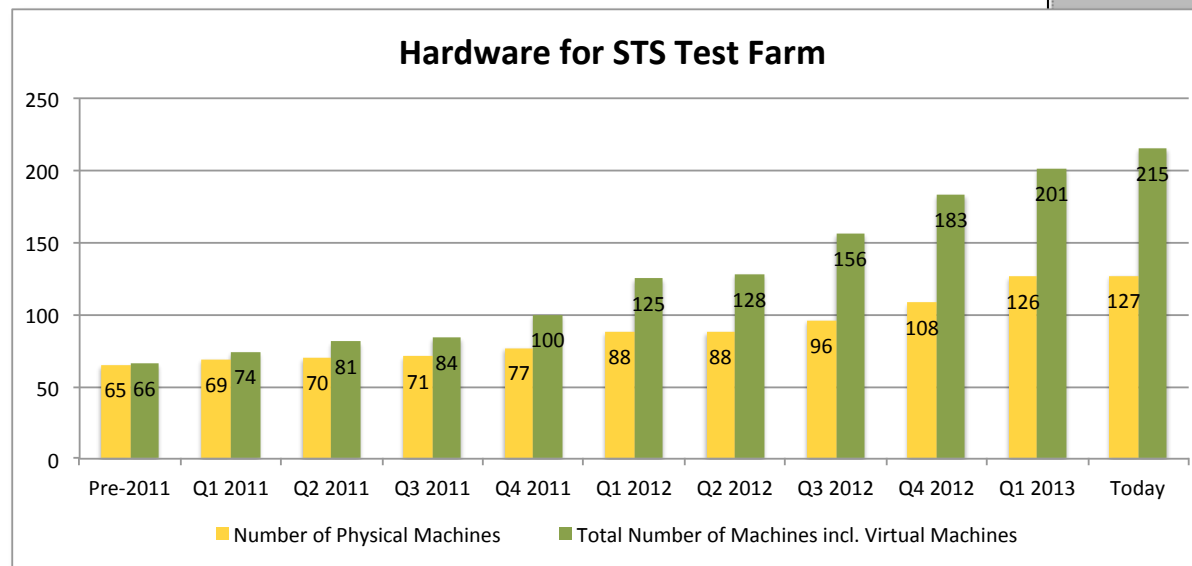
テスト自動化の進捗

- 2011年秋～2013年春までの Coverity Connect と End-to-End のテスト自動化の結果



インフラ

- 短いテストのほとんどは、Jenkins 上のビルドプロセスの一部として実行される
- 自動化テストのスケジュールとレポートのため、システムを拡張
 - すべてのプラットフォームのためのテストを自動化するため、サーバに大きな投資
 - システムの自動割当



STSweb Coverity Connect eureka

Dashboard | Summary | Overview | Reports | Job Rules | Products | Versions | Emails | Co

Test	Configuration	ubuntu12.04-64-chrome-linux	ubuntu12.04-64-chrome-macos	ubuntu12.04-64-firefox-64-linux	ubuntu12.04-64-firefox-64-mac	ubuntu12.04-64-chrome-linux
		X	X	X	X	X
		X	X	X	X	X
	crud.configuration.attributes.Add	P	P	P	P	P
	crud.configuration.attributes.Copy	P	P	P	P	P
	crud.configuration.attributes.Delete	P	P	P	P	P
	crud.configuration.attributes.Edit	X	X	X	X	X
	crud.configuration.components.Add	P	P	P	P	P
	crud.configuration.components.Copy	P	P	P	P	P
	crud.configuration.components.Delete	P	P	P	P	P
	crud.configuration.components.Edit	P	P	P	P	P
	crud.configuration.projectsstreams.StreamsAdd	P	P	P	P	P
	crud.configuration.projectsstreams.StreamsCopy	P	P	P	P	P
	crud.configuration.projectsstreams.StreamsDelete	P	P	P	P	P
	crud.configuration.projectsstreams.StreamsEdit	P	P	P	P	P
	crud.configuration.projectsstreams.StreamsLink	P	P	P	P	P
	crud.configuration.system.CrudSignInSettings	P	P	P	P	P
	cim/crud-sequential:crud.configuration.system.EmailAdd	P	P	P	P	P
	cim/crud-sequential:crud.configuration.system.EmailEdit	P	P	P	P	P
	cim/crud-sequential:crud.configuration.system.IssueCategorizationAdd	P	P	P	X	P
	cim/crud-sequential:crud.configuration.system.IssueCategorizationDelete	P	P	P	X	P
	cim/crud-sequential:crud.configuration.system.IssueCategorizationEdit	P	P	P	X	P

自動化を支えた新しい考え方と技術...



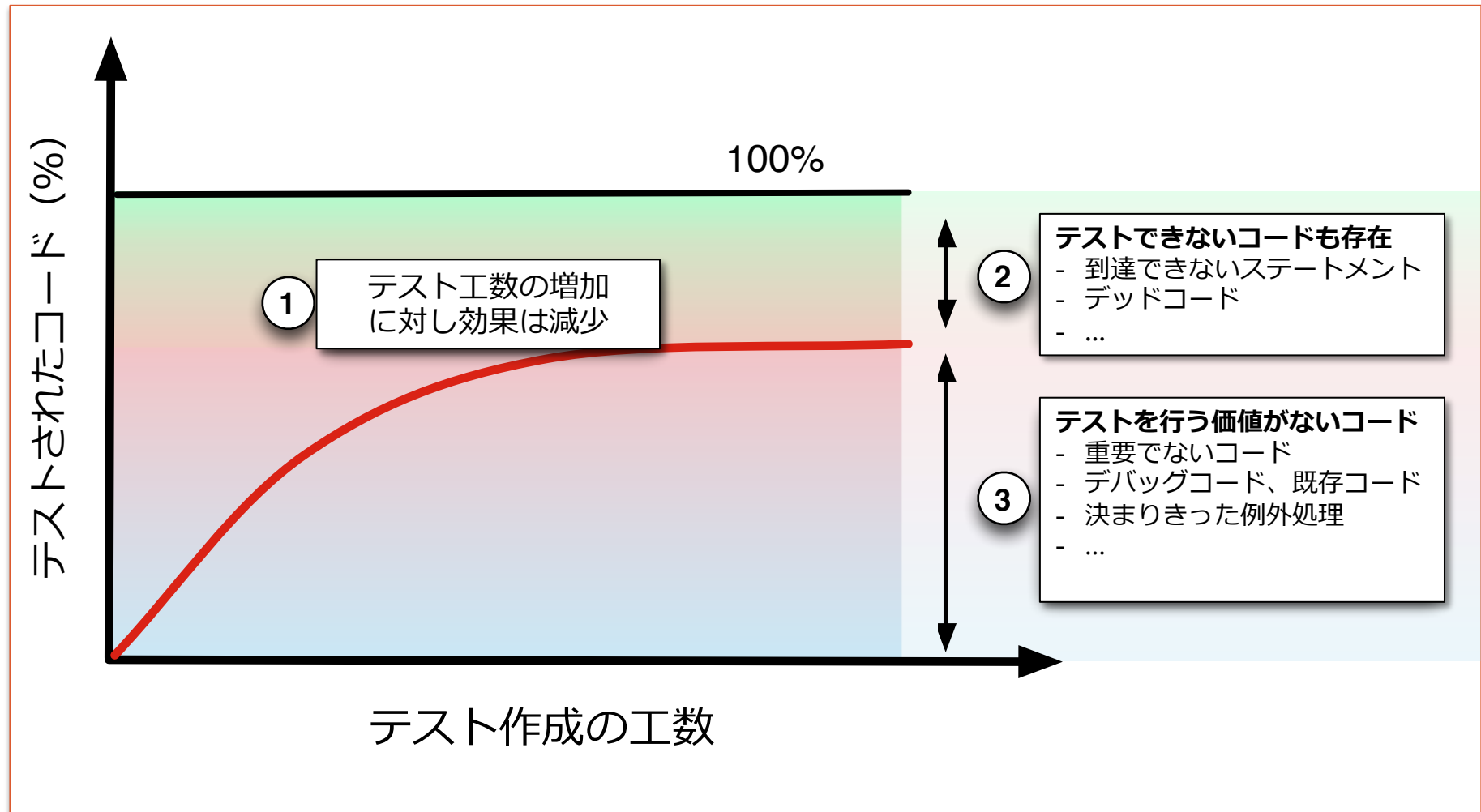
The Ultimate Machine – Claude Elwood Shannon
http://www.kugelbahn.ch/sesam_e.htm

No. 2: カバレッジの罠にはまらない

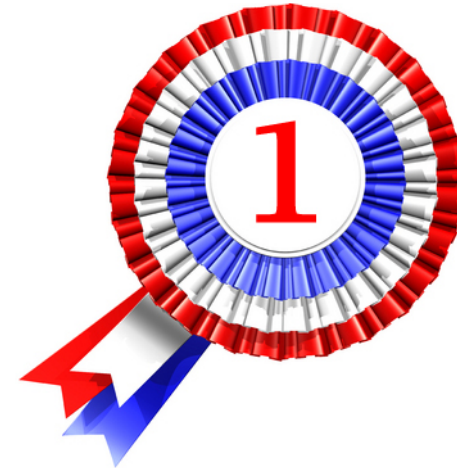
- 進捗を計測する良い指標がないことに気づいた
 - コードカバレッジはテスト工数に関しては良い指標ではなかった
 - 新機能のコードは、全体のカバレッジで把握しにくい
 - “開発しながらテストする”ことを牽引する計測可能な基準の欠如



コードカバレッジに対するチャレンジ



No.1: 優先度を設定しているか？

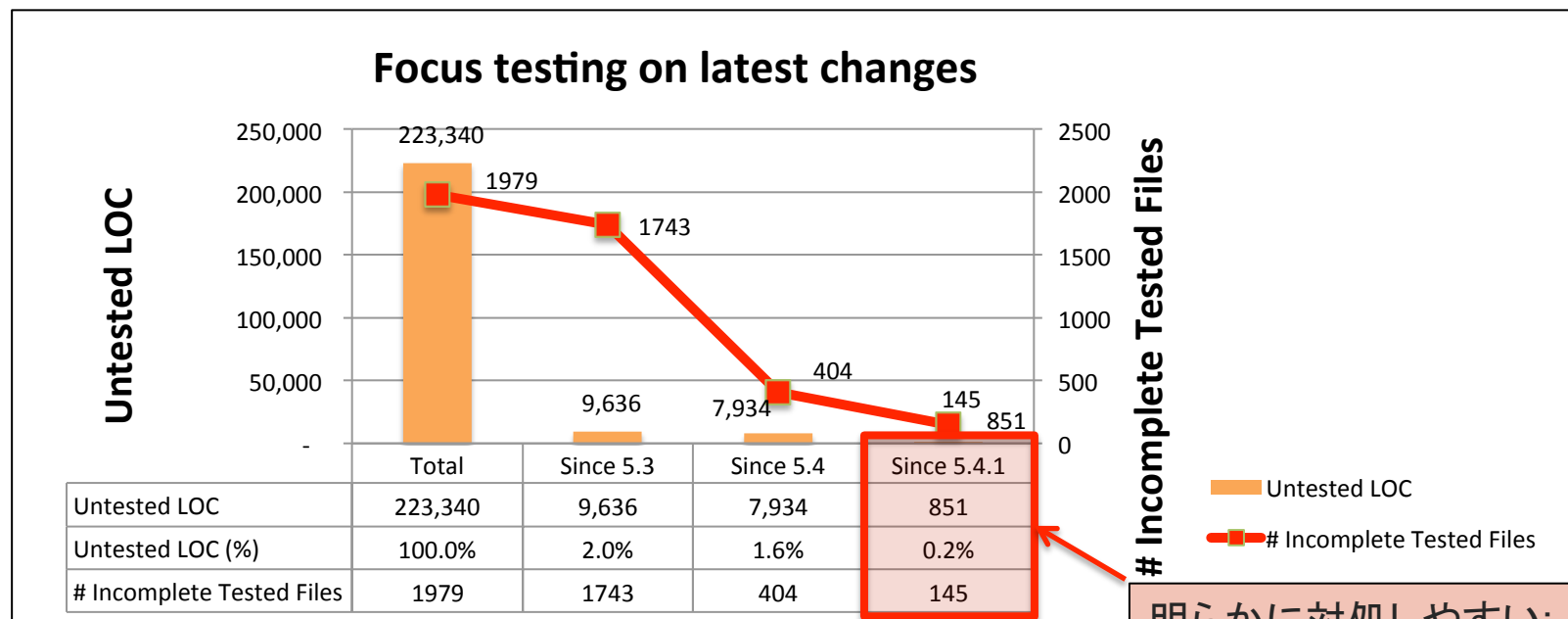


- どこからテストすべきか？
 - 以下のコードに着目した自動テスト開発をガイドする
 - 新規コード
 - リスクの高い部分を特に識別
 - 関連のないコードを除外
 - テストが不十分なコードを開発者に自動的に通知し、その対処を促す
 - アジャイルな受入れプロセスの一部として“テスト十分性”の施行を可能に

実験

“チェンジセット”にフォーカスしたテストポリシーを評価する単純な試み

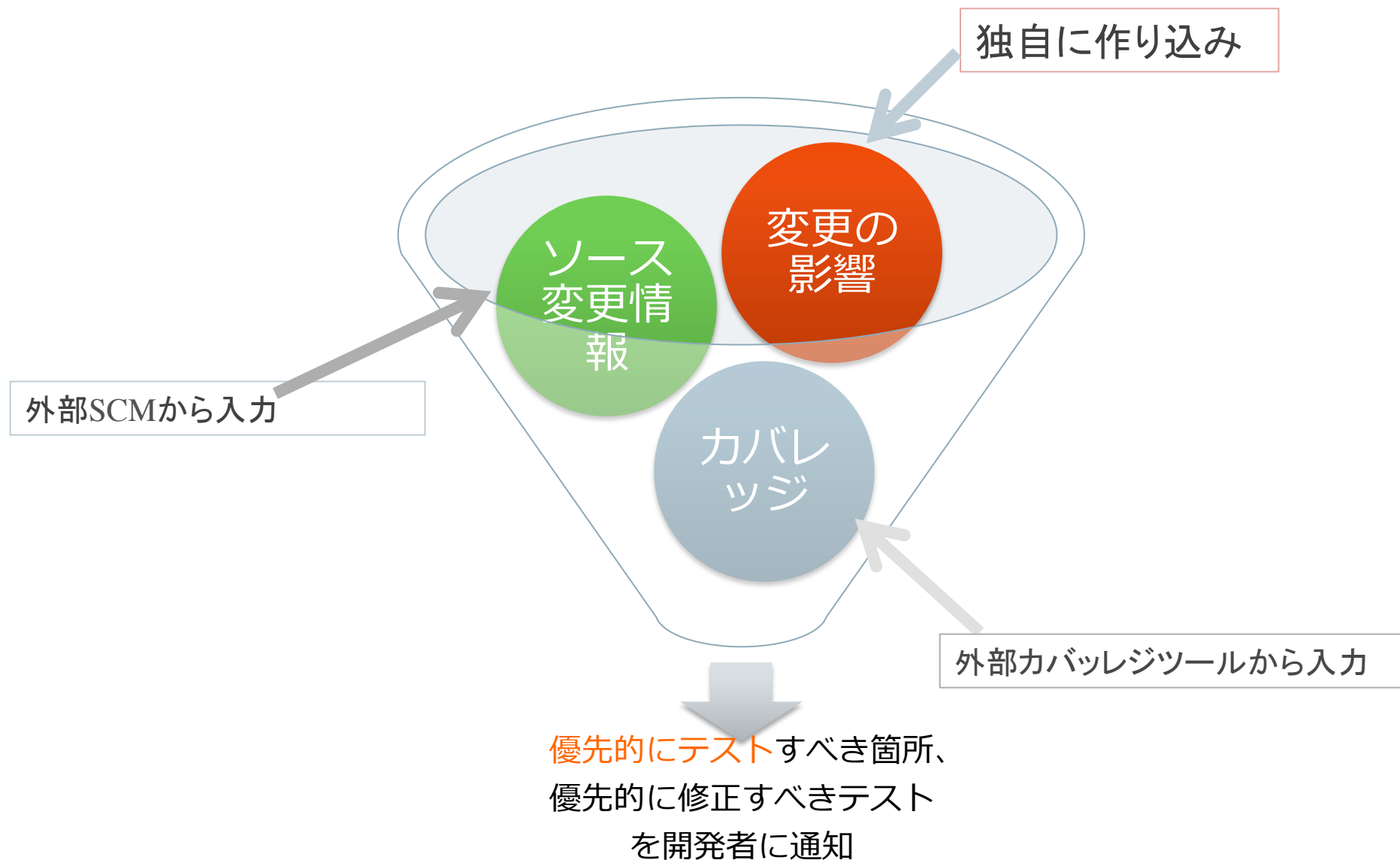
- フロントエンドと解析コード数：1,372,376 行
- 1,149,036 行がテストでカバー (83.7% カバレッジ), 223,340 行が未テスト
- 直前の変更にフォーカスすると、145 ファイル内の 851 行をカバーする必要がある



明らかに対処しやすい:

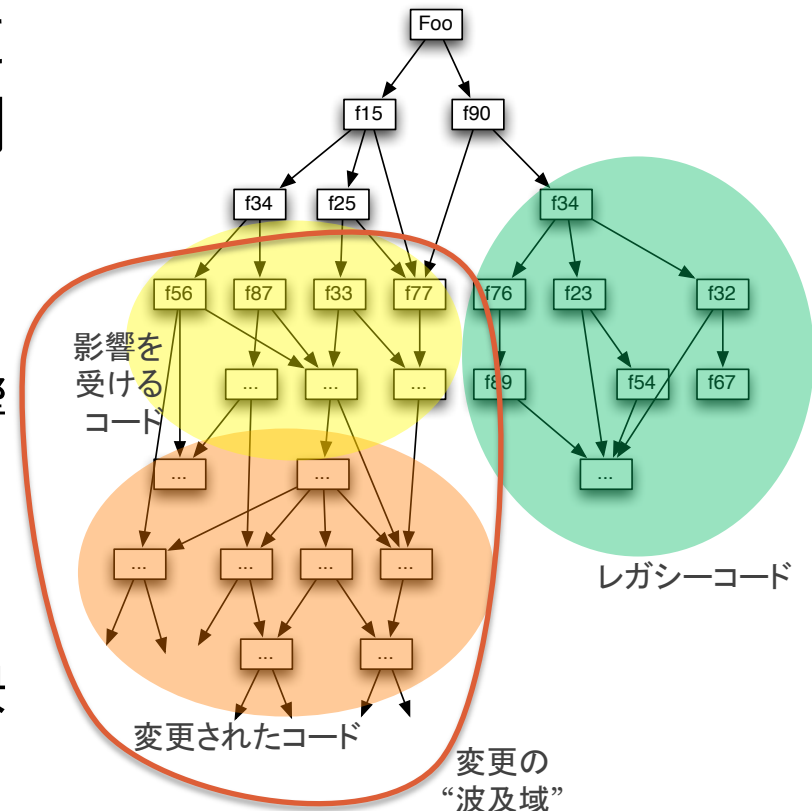
- 未テストコードの 0.2%
- 未テストファイルの 7%

自動テスト開発支援ツールの実装

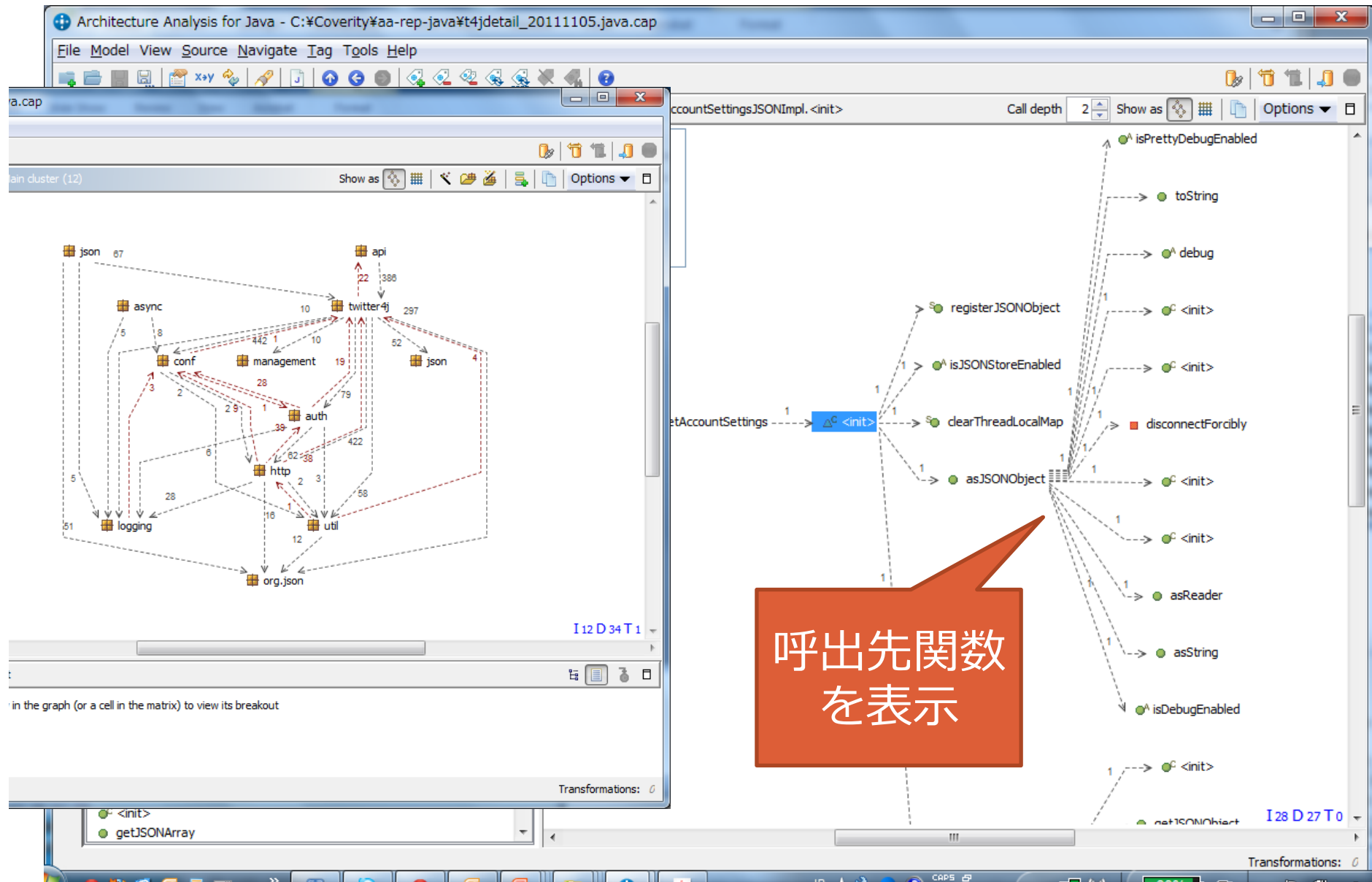


変更の影響解析

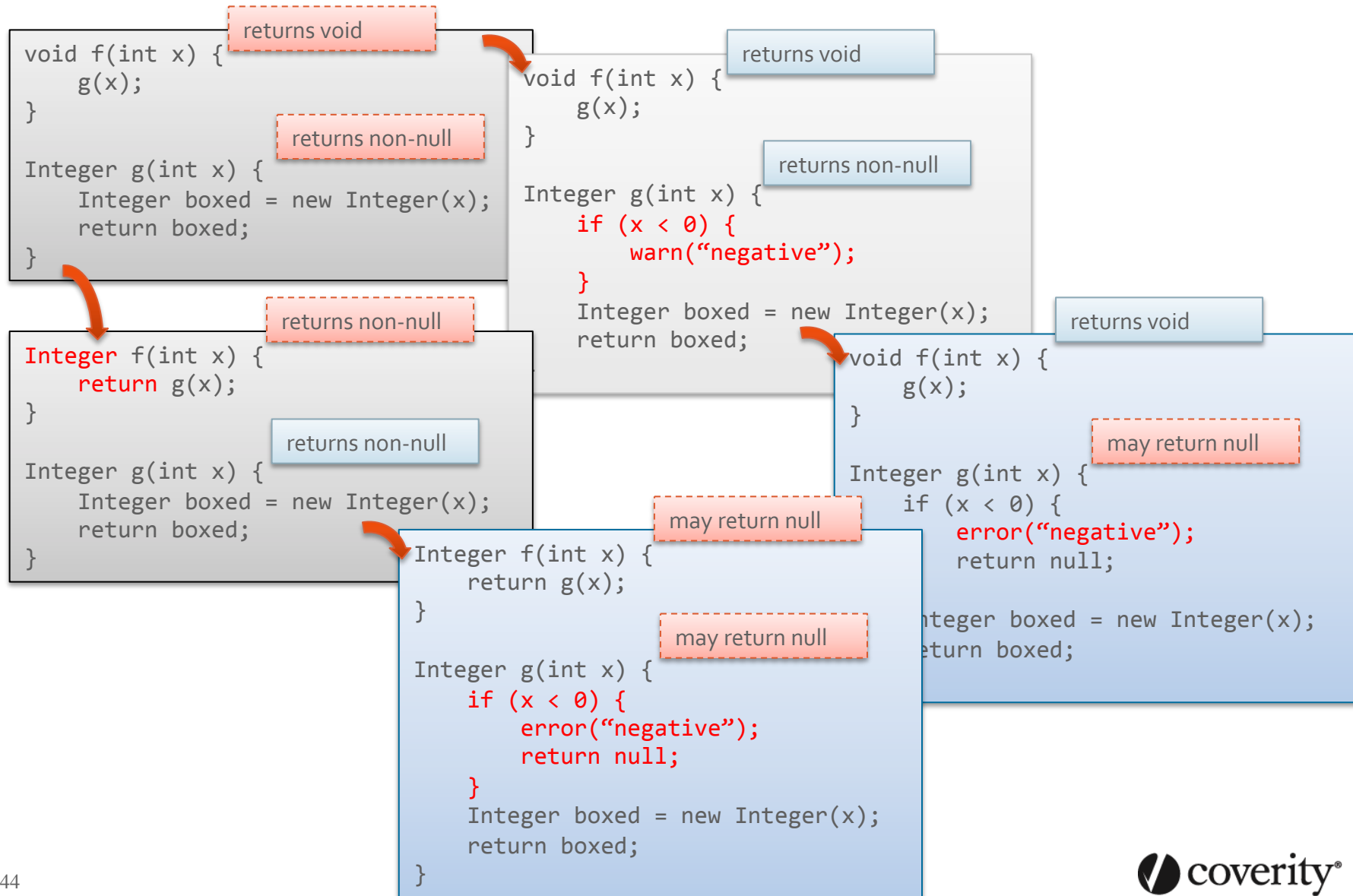
- チェンジインパクト解析は、変更があったコードが、他のコードに影響を与えるかどうかを的確に判断する。
- 例 1: 関数の挙動の変更は、その関数を呼び出すコードに影響を与える場合がある
- 例 2: クラスの継承関係の変更は、仮想関数呼出し時の実装決定に影響を与える場合がある



これまでは構造的に見るのが主流

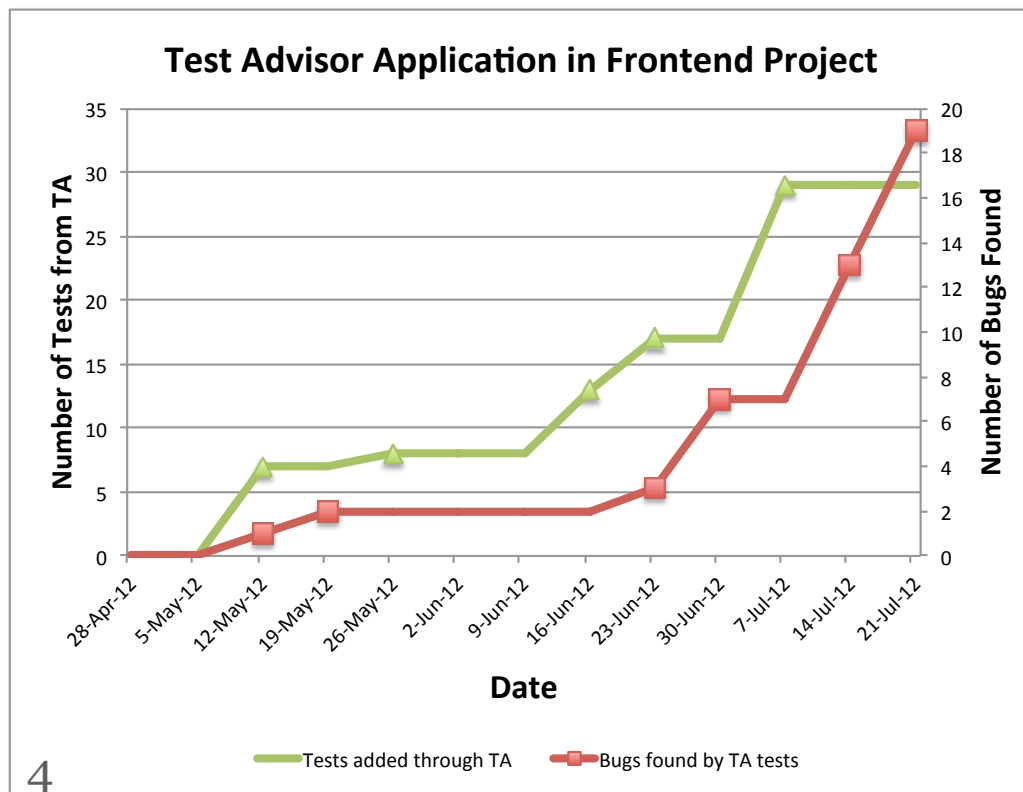


もう一つ先へ。変更の影響解析の例



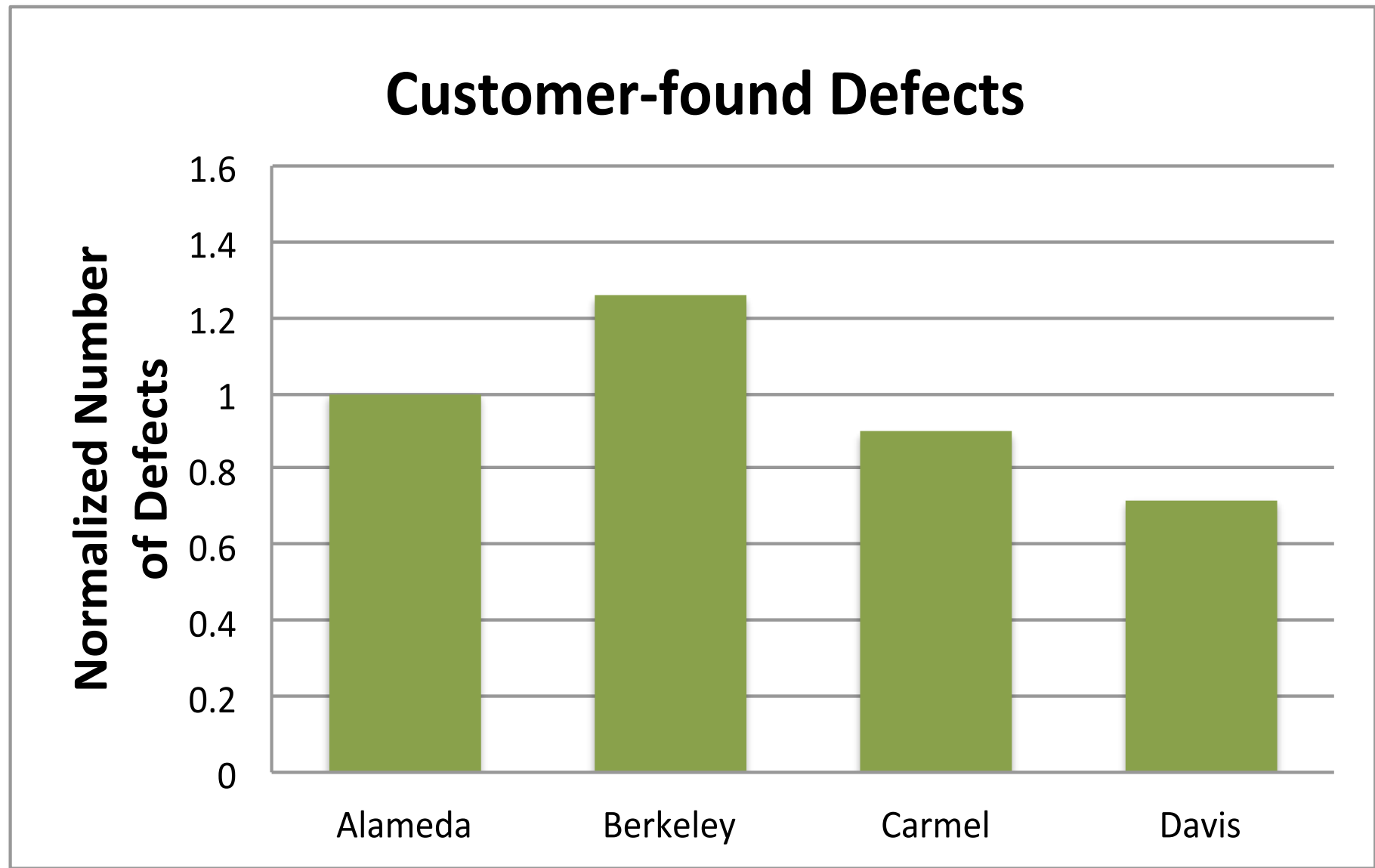
社内での経験 – ケーススタディ

- 十分テストされている製品のコンポーネントにこの社内ツールを適用
 - 3ヶ月間、1テストエンジニアが1週間のうち4時間、テストすべきと判断された箇所にテストを追加した
 - テストポリシー: テスト可能なコードは 100% テストする (除外: デバッグコード, 実行されないコード, その他)



- 1人週間換算で29件のテストを追加
- 製品から19件のバグを発見 – いくつかは重大なもの
 - 例: Keil コンパイラの正しい機能を阻害するバグなど

製品顧客が発見した不具合



Coverity 製品にもこの技術を組み込む

- Coverity Test Advisor (Java, C/C++, C#)
- 変更の影響箇所、高リスク箇所をお知らせします

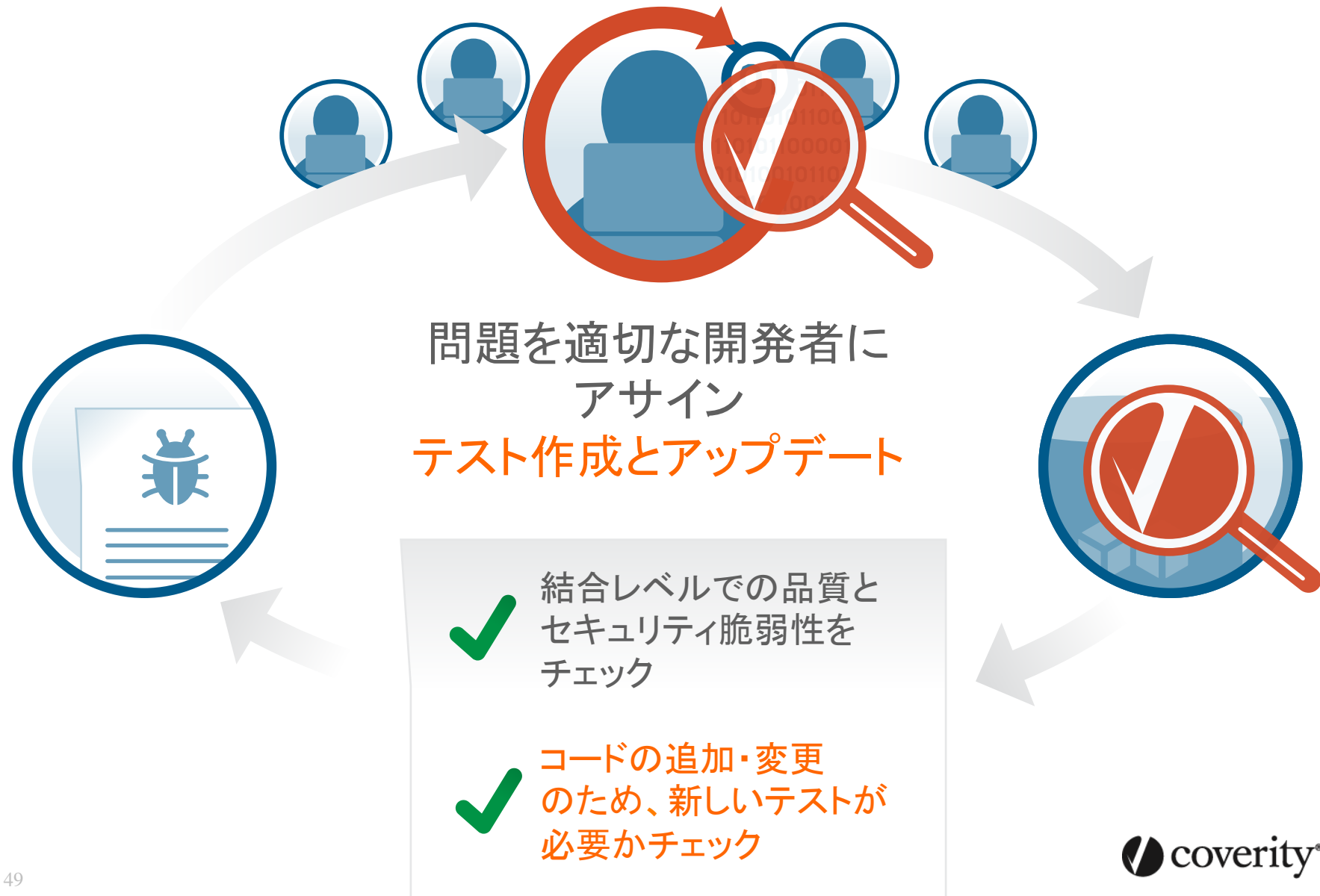
The screenshot displays the Coverity Test Advisor web interface. At the top, there's a navigation bar with 'coverity' logo, 'git' dropdown, 'コンフィギュレーション', 'ヘルプ', 'taro coverity', and a search bar for 'CID を入力'. Below this is a table of issues:

CID	タイプ	影響	状態	初回検出日	担当者	分類	重要度	アク
14667	不十分な関数カバレッジ	低	新規	2013/03/20	未アサイン	未分類	未指定	未
14648	不十分な関数カバレッジ	低	選別済み	2013/03/20	yukio	未テスト	高	未
14634	不十分な関数カバレッジ	低	新規	2013/03/20	未アサイン	未分類	未指定	未

Below the table, it says '64 件中 1 件の問題を選択'. The main area shows a code editor for the file '/home/nonroot/Coverity/samples/git/builtin/receive-pack.c'. The code is a C function 'receive_pack_config'. A red box highlights a warning: 'CID 14648 (#1/1): 不十分な関数カバレッジ (TA.INSUFFICIENT_COVERAGE)'. The text explains that the function 'receive_pack_config' has insufficient coverage, with 3 lines (66%) failing the rule filter and tests. It suggests increasing coverage to 100% by adding more lines.

On the right, a detailed view of CID 14648 is shown. It includes the title '14648 不十分な関数カバレッジ', a description of the issue, and a '選別' (Filter) section with dropdowns for '分類' (未テスト), '重要度' (高), and 'アクション' (未確定). There are also input fields for '外部参照' and 'アウトソース', and a '担当者' (yukio) field. At the bottom, there are buttons for '適用して次へ' and '適用'.

インパクト解析を取り入れた テスト自動化ワークフロー



まとめ

- No.5: テスト自動化が解決策？
- No.4: スキルセットは正しい？
- No.3: クリーンなコード？
- No.2: 正しい計測？
- No.1: 優先度を設定せよ！

**Make every developer
a rock star !**



Coverity Development Testing Platform

解析 | 修正 | 統制

