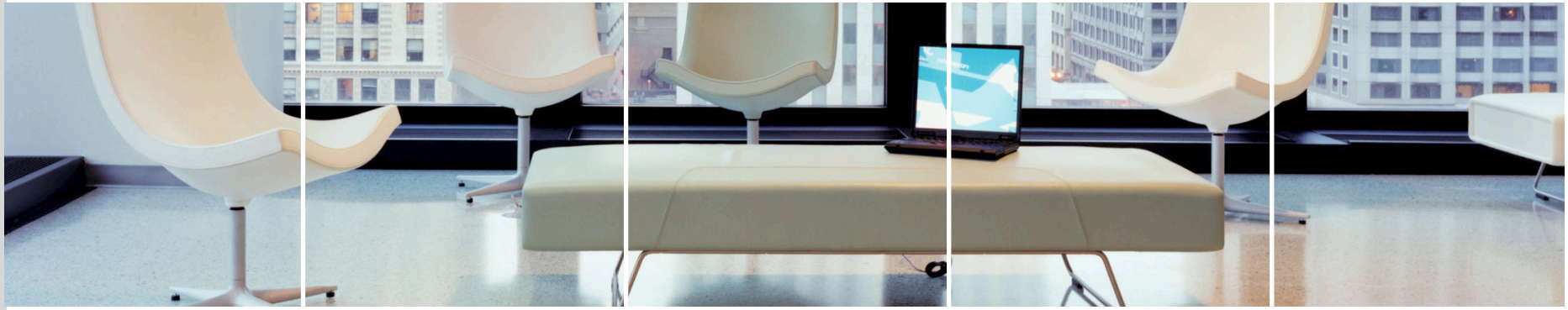
テスト力 / レビュー力向上に向けた 欠陥管理のススメ

The Defect Engineering Technique for Test / Quality engineers



Nobuhiro Hosokawa
(CARVIN@jp.ibm.com twitter:// __Dreamstate)
October 1. 2010

IBM-J Quality Engineering since 2009
... because all defects are also "Artifacts"

はじめに

Nobuhiro Hosokawa

Manager
Quality Engineering
AIS – Global Business Service
IBM Japan Ltd.

carvin@jp.ibm.com

Twitter://__Dreamstate

ようこそ! 欠陥エンジニアリングの世界へ!

欠陥エンジニアリングは、一般的なあまり馴染みのない言葉ですが欠陥の記録／分類／活用を通じて品質管理活動全般へ貢献する、いわば「ゲンバノチカラ」「基礎体力」と言っても過言ではない、重要な概念です。

しかし日々の技術者としての活動では、欠陥管理DBに検出した欠陥を記録しますが、実は「欠陥を記録してどうする？」という目的意識が欠如している場合が少なくありません。

また「欠陥管理DBを参照して利用するのは誰？」「欠陥はどうやって分類整理したらいいの？」という概念等も、国内ではあまり知られていないトピックです。

本プレゼンテーションでは、欠陥管理の一般的な問題点となぜ欠陥管理が負荷ばかり大きくうまく活用されないか、またどのように整理すればよいかを皆さんと一緒に考え、「ゲンバノチカラ」としてあるべき姿を紹介します。

Contents

- 問題定義： 解決すべき問題
- プロジェクトと欠陥の関係
- 医学との対比
- 欠陥の情報分類
- 誰が使う？メリットは？
- 未来に向けて

今日のプレゼンテーション技法：クレショフ効果（**Kuleshov Effect** ）

■複数のストーリーを“つぎはぎ”でつなぐモンタージュという技法を用いて、つぎはぎの前後に位置する他ストーリーの意味に対して及ぼす / 強調する効果のことをいう。

■もとは映画／映像技術のプロップ（＝ストーリー）構成技術の一つとして有名だったが、プレゼンテーション技術として用い、メインメッセージの強調を行うテクニックとして一般的に普及した

今日のテーマ

No Silver Bullet ... ?

But, we really want “new Bullet”...

Problem need to solve : 問題定義

IBM-J Quality Engineering
Quality Inspection Inside

バグ記録してますか？

バグ記録を活用してますか？

プロジェクト内でバグ情報を記録してますか？

病気標本と病理学研究みたいものが、ITの世界には？

プロジェクトと欠陥の関係は？

全検出欠陥を”何でもいいから”DBに登録せよの法則

IBM-J Quality Engineering
Quality Inspection Inside
since 1999

バグ報告者側（テスター、開発者....）

1. 検出順序を意識せず登録する
2. プロジェクトで検出した欠陥は全て、BTS (Bug Tracking System) に記録する
3. 同種の欠陥があるかもしれないがとりあえず入力・入れる
4. 誰がどう使われて、どう意思決定されるかわからないけどとりあえず登録する
5. 粒度はばらばらで入力する
6. 明確な「何を欠陥として記録するか」というルールと欠陥定義がない

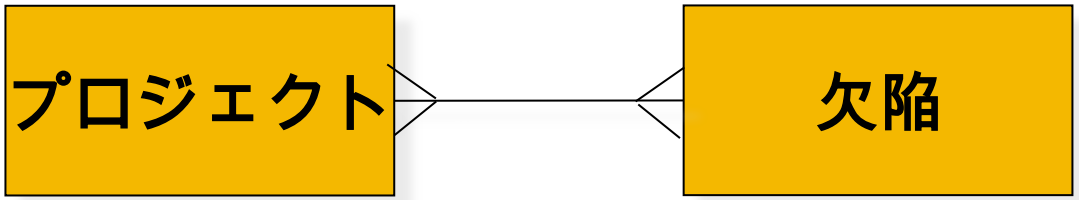
報告される側（PM/品質管理担当...）

1. 未整理で「何を言っているかわからない」がとりあえず多量に登録されてくる
2. 欠陥の関連が不明。全ての欠陥は単独と考える
3. 出てきた順に欠陥除去しようと考える
4. 簡単なものから対処するよう指示する
5. 週次報告書以外にバグ情報を使わない
6. 再発している欠陥に気付かない・気付けない
7. プロジェクトを終わったら使わない

- せっかく時間と工数をかけて記録した欠陥情報を、実は利用していない・活用できていない事例が散見される
- 欠陥管理DB・BTSの情報が活用されない理由はなにか？

プロジェクトと欠陥の関係：ERダイアグラム風

- プロジェクトと欠陥の関係を整理すると次のような関係。
  1. プロジェクトには複数の欠陥が発生する
  2. 一つの欠陥は複数のプロジェクトで検出される



- 上記M対Mの関係は、１対１関係の「関連エンティティ」をおくと整理できる
  1. プロジェクトと欠陥の間に「プロジェクト欠陥」エンティティを設定。
  2. 各１対Mの関係でリレーションを整理

普段BTSに登録している欠陥情報は、実は「プロジェクト欠陥」であり「欠陥」そのものではない！

医学との対比

医学には．．．

✓医学では、稀な症例や現場の事例を学会等を通じて共有する仕組みがある

✓現場の実例から抽象化して「病気」だけを定義・管理・維持する学会・権威機関がある。そういう医学分野がある

✓現場医師は最新動向や検査・対処方法を継続的に学ぶ習慣が付いている

IT業界には．．．

- 稀なバグや被害の大きな欠陥を業界全体で共有する仕組みがない
- そもそもバグを研究する「バグ屋」という分野の人材も・技術分野も存在しない
- 現場は「除去」さえできればよいため欠陥の情報の獲得を行わない

PubMed
PubMed comprises more than 20 million citations for biomedical literature from MEDLINE, life science journals, and online books. Citations may include links to full-text content from PubMed Central and publisher web sites.

医学分野で世界最大の文献データベース。1966年からNLM（米国国立医学図書館）でデータ収集が始まり，現在毎月約3万件の文献が新たに追加される。現在では，米国を中心に約70ヵ国から，900万件を超える文献が収録される。

欠陥の情報分類： #1 「病理学」風の整理

医学のような視点では、定義と用途を重視した永続的・長期的な情報整理を行います

欠陥の情報

定義・ 基礎情報

1. 名前・ 別名
2. 欠陥の定義・ 分類
3. 一般的な推定原因発生・ 確率
4. 併発症・ 合併症
5. 特記事項

検出情報

1. どのように検出するか？
2. どのメトリクスを計れば欠陥兆候を捕らえることができるか？
3. 定性情報？定量情報？

兆候情報

1. どうやったら欠陥に気付くか？
2. どのメトリクスを計れば欠陥兆候を捕らえることができるか？
3. 定性情報？定量情報？

除去方法・ 予防方法

1. **関連する欠陥・ 連鎖欠陥**
2. 除去時の注意点
3. 副作用・ 除去リスク
4. 再発予防方法

欠陥の情報分類： #2 レビュー・ テスト専門家の整理

IBM-J Quality Engineering

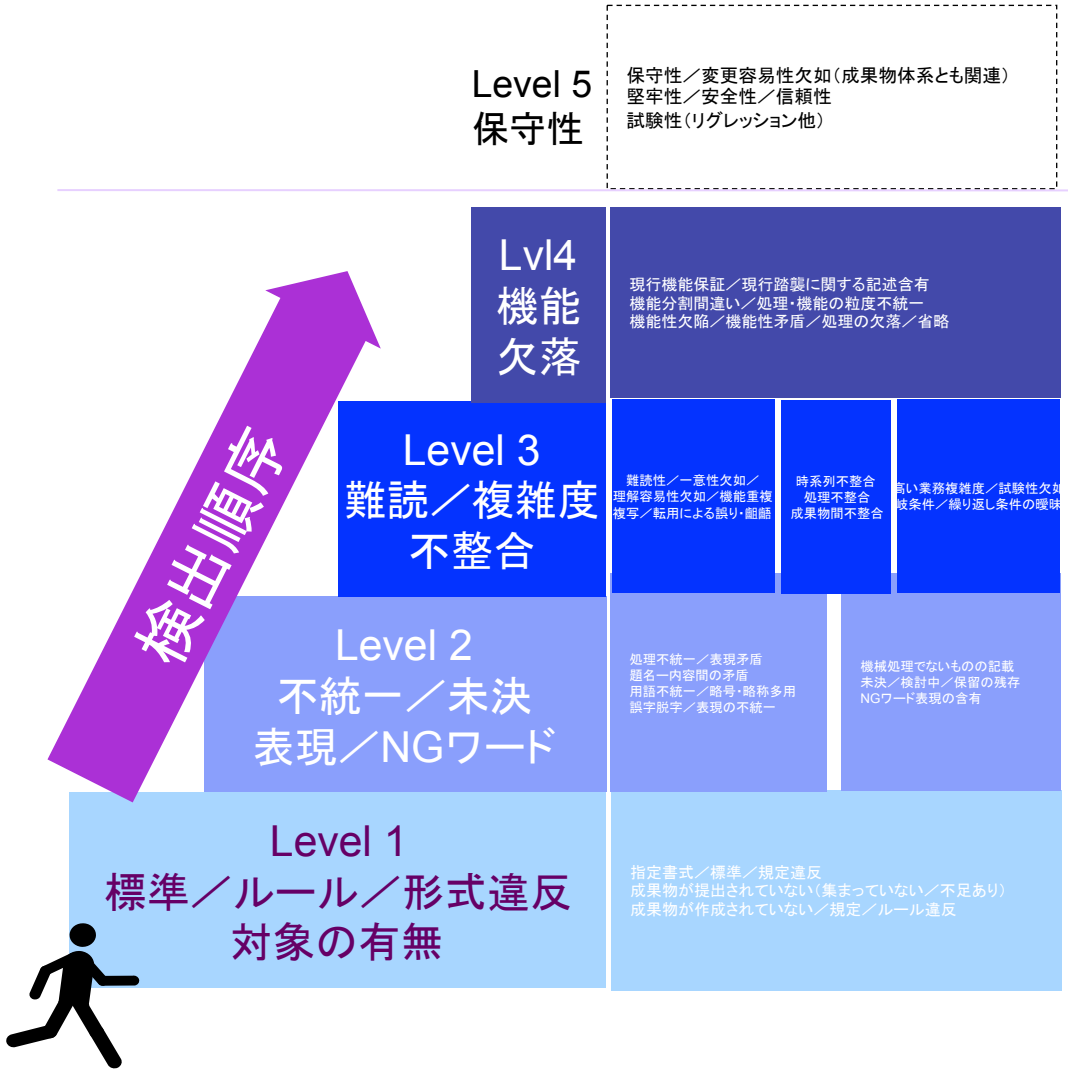
レビュー・テスト担当者の視点では、検出順序を考慮した階層的な欠陥分類が必要です

Level 5 保守性	保守性／変更容易性欠如（成果物体系とも関連） 堅牢性／安全性／信頼性 試験性（リグレッション他）
Lvl4 機能 欠落	現行機能保証／現行踏襲に関する記述含有 機能分割間違い／処理・機能の粒度不統一 機能性欠陥／機能性矛盾／処理の欠落／省略
Level 3 難読／複雑度 不整合	難読性／一意性欠如／ 理解容易性欠如／機能重複 複写／転用による誤り・齟齬
	時系列不整合 処理不整合 成果物間不整合
	い業務複雑度／試験性欠 条件／繰り返し条件の曖
Level 2 不統一／未決 表現／NGワード	処理不統一／表現矛盾 題名一内容間の矛盾 用語不統一／略号・略称多用 誤字脱字／表現の不統一
	機械処理でないものの記載 未決／検討中／保留の残存 NGワード表現の含有
Level 1 標準／ルール／形式違反 対象の有無	指定書式／標準／規定違反 成果物が提出されていない（集まっていない／不足あり） 成果物が作成されていない／規定／ルール違反

検出順序

欠陥の情報分類： #2 レビュー・テスト専門家の整理

それぞれのレベルに応じた検出方法を用いるべきです。



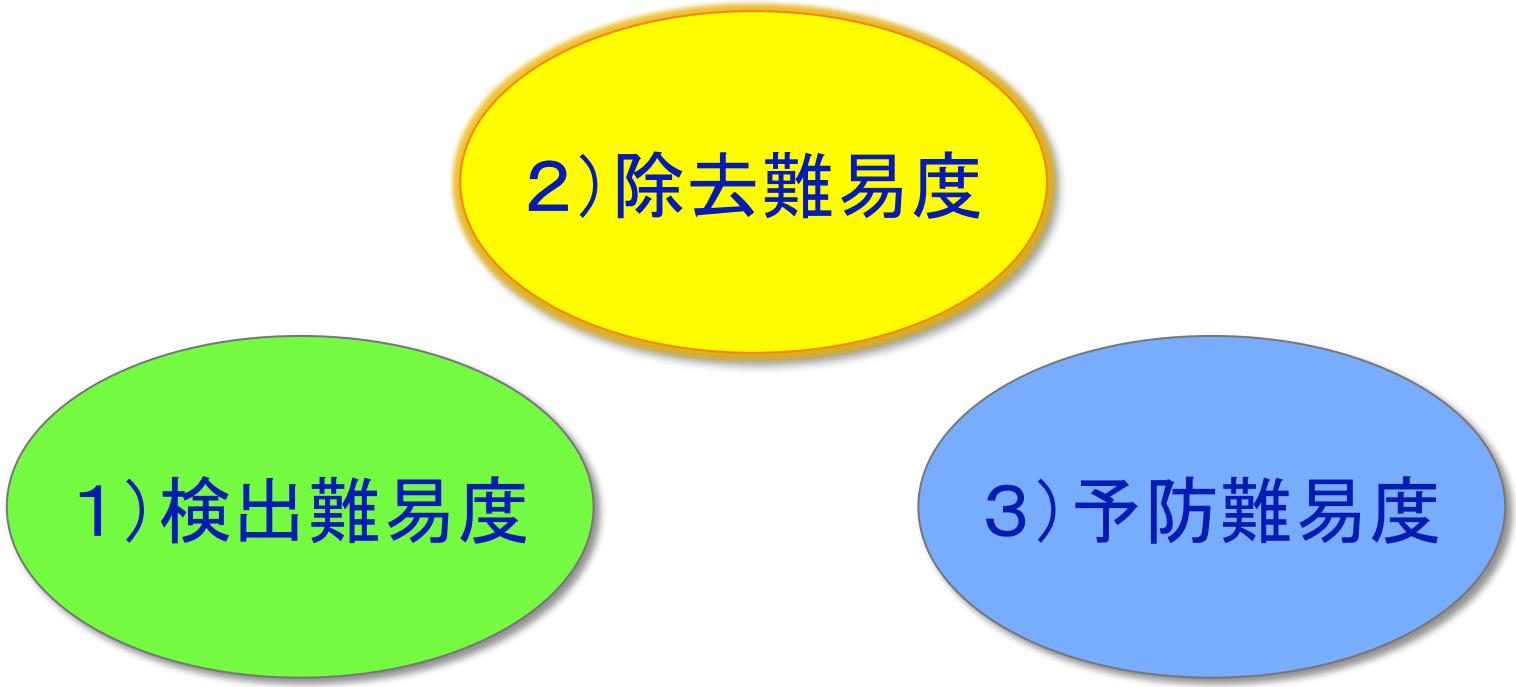
目視でしか検出できない /
検出難易度が高い欠陥

自動化ツール
測定ツールで検出する欠陥

開発者のセルフレビュー /
チェックリストで検出する領域

欠陥の情報分類： #3 「品質管理」専門家の整理

品質管理の専門家の立場からは、以下の3つの観点での欠陥分類が必要です。



検出困難度 ： 開発者またはプロジェクト内部メンバーによる自己レビューで検出しにくい欠陥
除去困難度 ：欠陥の存在は確認できたが本番環境から除去しにくく、除去工数のかかる欠陥
予防困難度 ：他の困難度に関係なく頻発し根本原因対策が難しいため、再発予防の難しい欠陥

難易度別分類の例：記入例

ID	欠陥内容	検出困難度	除去困難度	予防困難度
1	synchronizedによる同期保護の適用範囲誤り	中	難	難
2	Singletonパターンでの非効率な初期化	中	易	易
3	synchronizedを用いることによるボトルネック（スループット低下）#1	中	難	易
4	synchronizedを用いることによるボトルネック（スループット低下）#2	易	難	易
5	SimpleDateFormatの共有による、誤った日付文字列化処理	中	易	易
6	マルチスレッド環境を考慮していないSingletonパターン適用	難	易	易
7	誤ったSingletonパターンでの同期保護	難	易	易
8	誤ったNullチェック	中	易	中
9	配列を返すメソッドで例外ケースにnullを返す弊害	易	難	易
10	Stringの非効率な連結	易	易	易
11	使用されていない変数#1	難	中	中
12	使用されていない変数#2	難	中	中
13	使用されていない変数#3	難	中	中
14	不必要なDBアクセス	難	難	難
15	例外処理漏れの可能性	難	易	易
16	例外処理漏れの可能性	難	易	易
17	synchronized保護抜けによるマルチスレッド動作に支障をきたす可能性	難	難	難
18	誤ったNullチェック	中	易	中
19	誤ったNullチェック	中	易	中
20	誤ったNullチェック	中	易	中
21	例外の握りつぶし	易	中	難
22	疑わしい例外処理	難	難	難
23	類似Javascriptの個別定義	難	中	難

誰が使う？メリットは？

■ プロジェクトマネージャー（開発中）

②兆候情報＋③検出情報＝品質管理上の必要アクション指示

③検出情報＋④除去・予防方法＝品質管理上の予防アクション指示

■ プロジェクトマネージャー・経営層（プロジェクト計画時）

②兆候情報＋④除去・予防方法＝当該プロジェクトのリスク判定＋プロジェクト計画

①定義情報＋④除去・予防方法＝品質管理計画

■ プログラマ・設計者

①定義情報＋③検出情報＝

自己レビュー・自浄

■ テスト／レビュー担当者

①～④の全部！！どうやって？

まとめ

どれが優れた「銀の弾丸」かを模索するより、どの怪物を倒すかが重要。
あなたが戦うべき真の相手は？

もうそろそろ「技」の学習よりも大切なことがあることに気づかない？

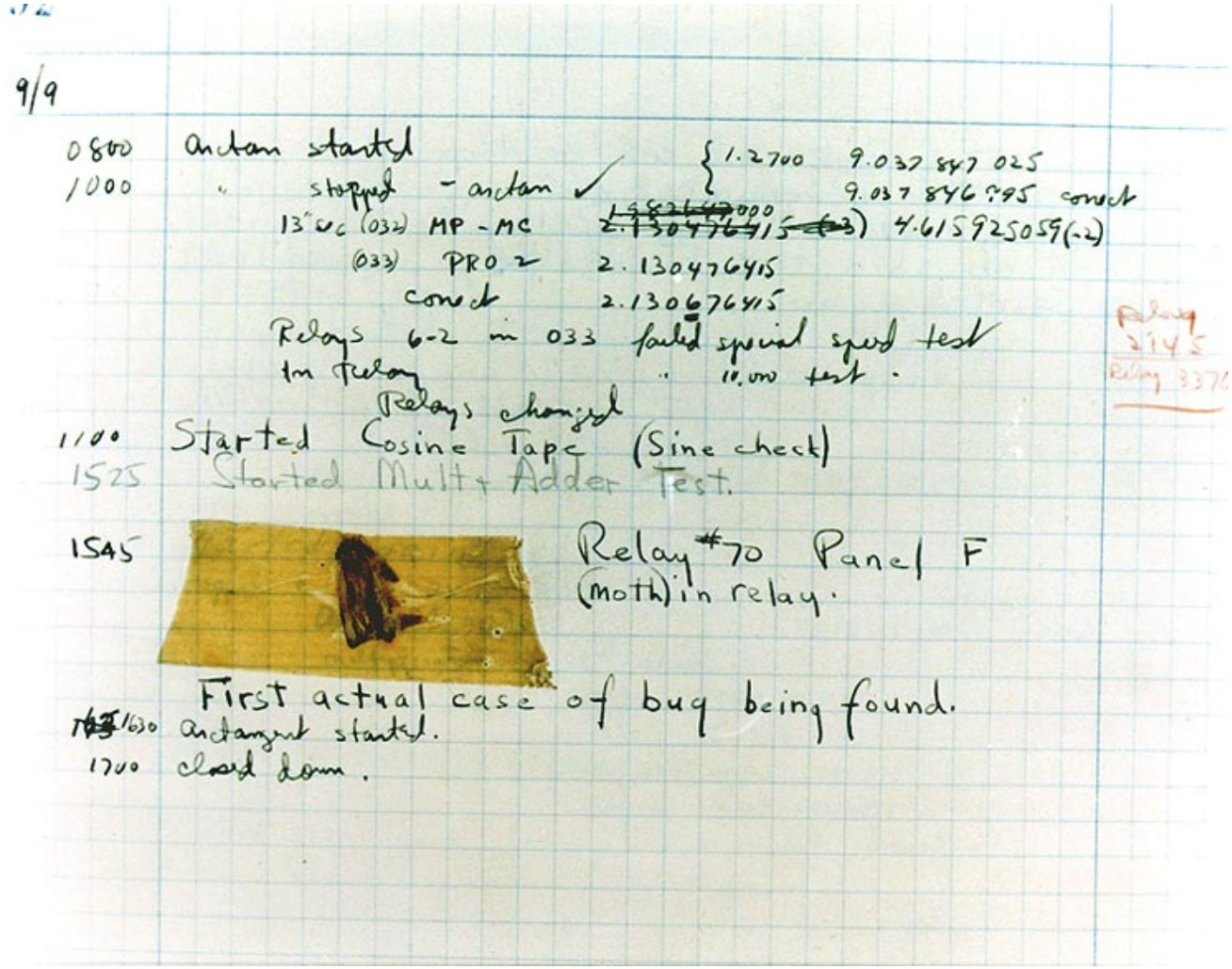
未来に向けて

IBM-J Quality Engineering
Quality Inspection Inside

- 欠陥の検出技術（How）も大切だが、欠陥そのもの（=What）を重視すべきでは？
- 欠陥情報を「武器」「財産」にする活動が必要？
- 業界横断的な「欠陥を共有する仕組み」を作れないか？もしそれが存在したら？
- これから育つ次世代のテスト・レビュー専門家たちを欠陥情報なしでどう育てる？

Thanks ☺

世界で最初のバグ 1947年9月9日(1947-09-09)



Removed caption read: Photo # NH 96566-KB First Computer "Bug", 1945
日付 1947年9月9日(1947-09-09)
原典 U.S. Naval Historical Center Online Library Photograph NH 96566-KN
著者 Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988.
許可 (このファイルの再利用) *This image is a work of a sailor or employee of the U.S. Navy, taken or made during the course of the person's official duties. As a work of the U.S. federal government*
他のバージョン http://americanhistory.si.edu/collections/object.cfm?key=35&objkey=30

概要

The First "Computer Bug" Moth found trapped between points at Relay # 70, Panel F, of the Mark II Aiken Relay Calculator while it was being tested at Harvard University, 9 September 1947. The operators affixed the moth to the computer log, with the entry: "First actual case of bug being found". They put out the word that they had "debugged" the machine, thus introducing the term "debugging a computer program". In 1988, the log, with the moth still taped by the entry, was in the Naval Surface Warfare Center Computer Museum at Dahlgren, Virginia, which erroneously dated it 9 September *1945*. The Smithsonian Institute's National Museum of American History and other sources have the correct date of 9 September *1947* (Object ID: 1994.0191.01). The Harvard Mark II computer was not complete until the summer of 1947.

Defect Definition Chart