ソフトウェアテストシンポジウム
JaSST ージャストー Japan Symposium on Software Testing

S4）セッション

「これからの“バグ”の話をしよう」

JaSST‘13 Kyushu

ソフトウェアテストシンポジウム 2013 九州

2013年11月1日(金)

はじめに

概要:

▪欠陥エンジニアリングという未踏分野が注目されています。未だに欧米ではエンジニアリング研究が進んでいない「欠陥」を科学することで、開発者、PM、経営者等にとって様々な利得について考察します。

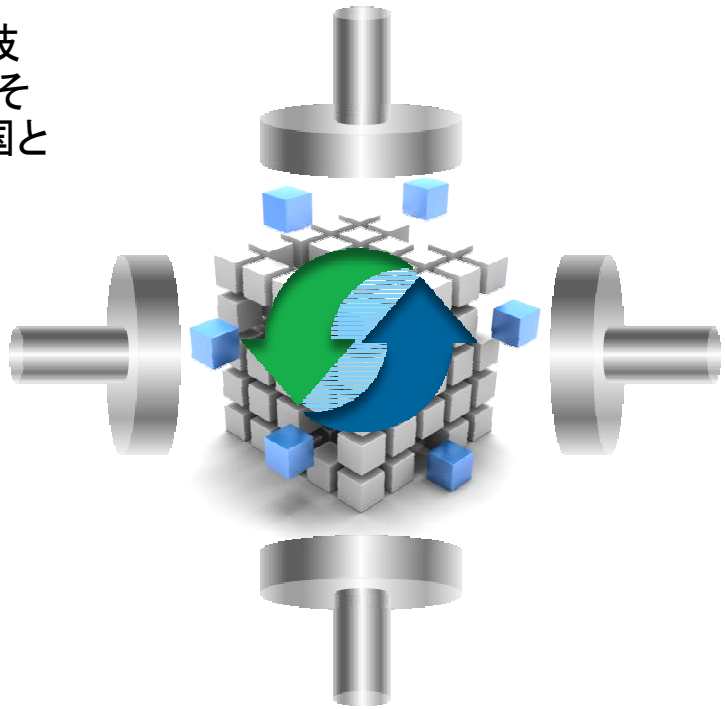
▪今後のIT産業発展の一翼を担う欠陥エンジニアリングの技術について歴史的俯瞰、今まで**なぜ普及しなかったのか**、そしてどこに向かうべきなのか？を議論し、九州から日本全国と世界に向けて新しいレビューやテストの概念を提案します。

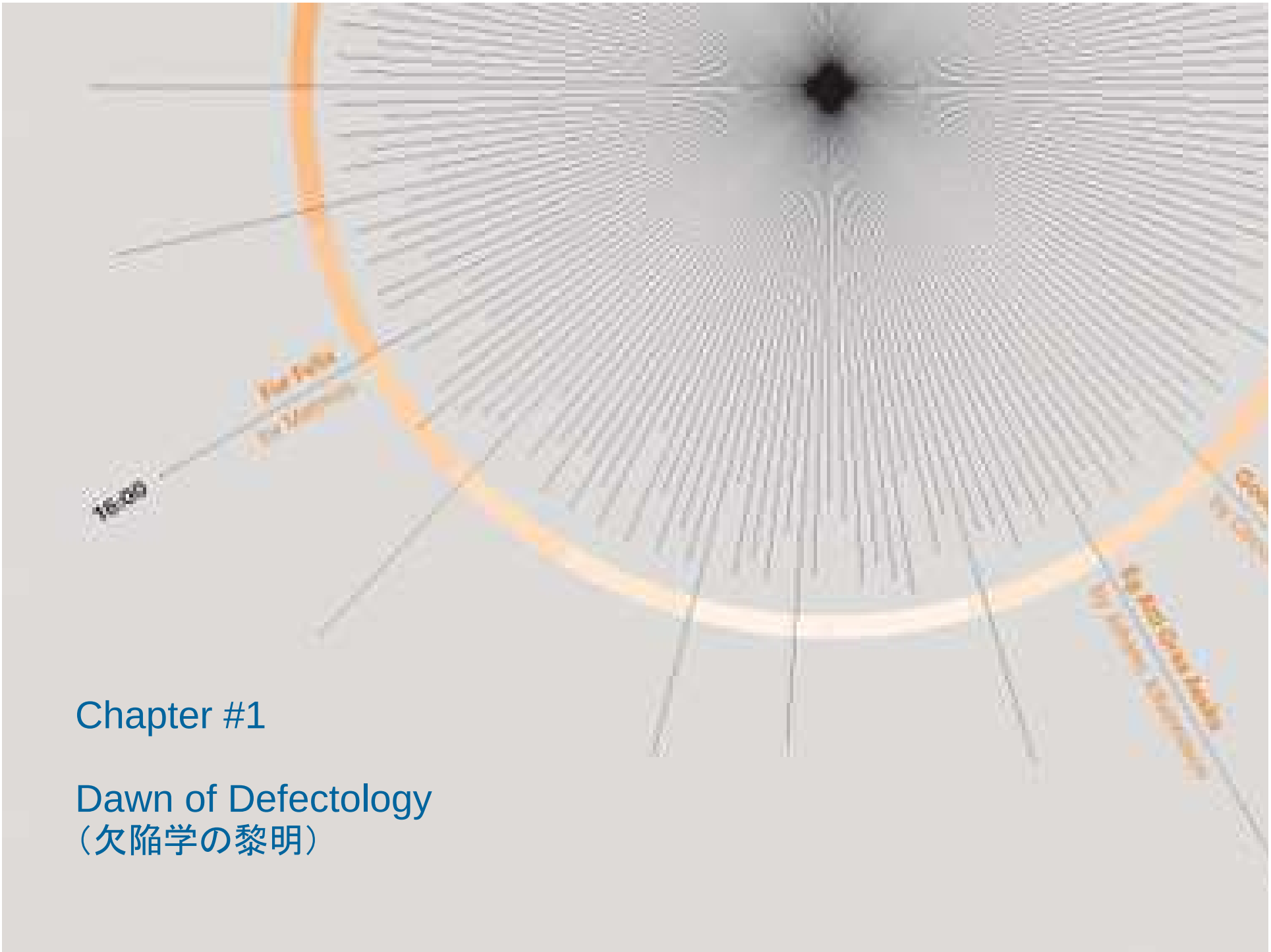
ルール:

Lots of Parking…（寄り道をたくさん）

– **Raise your hands**

- たくさん発言してください



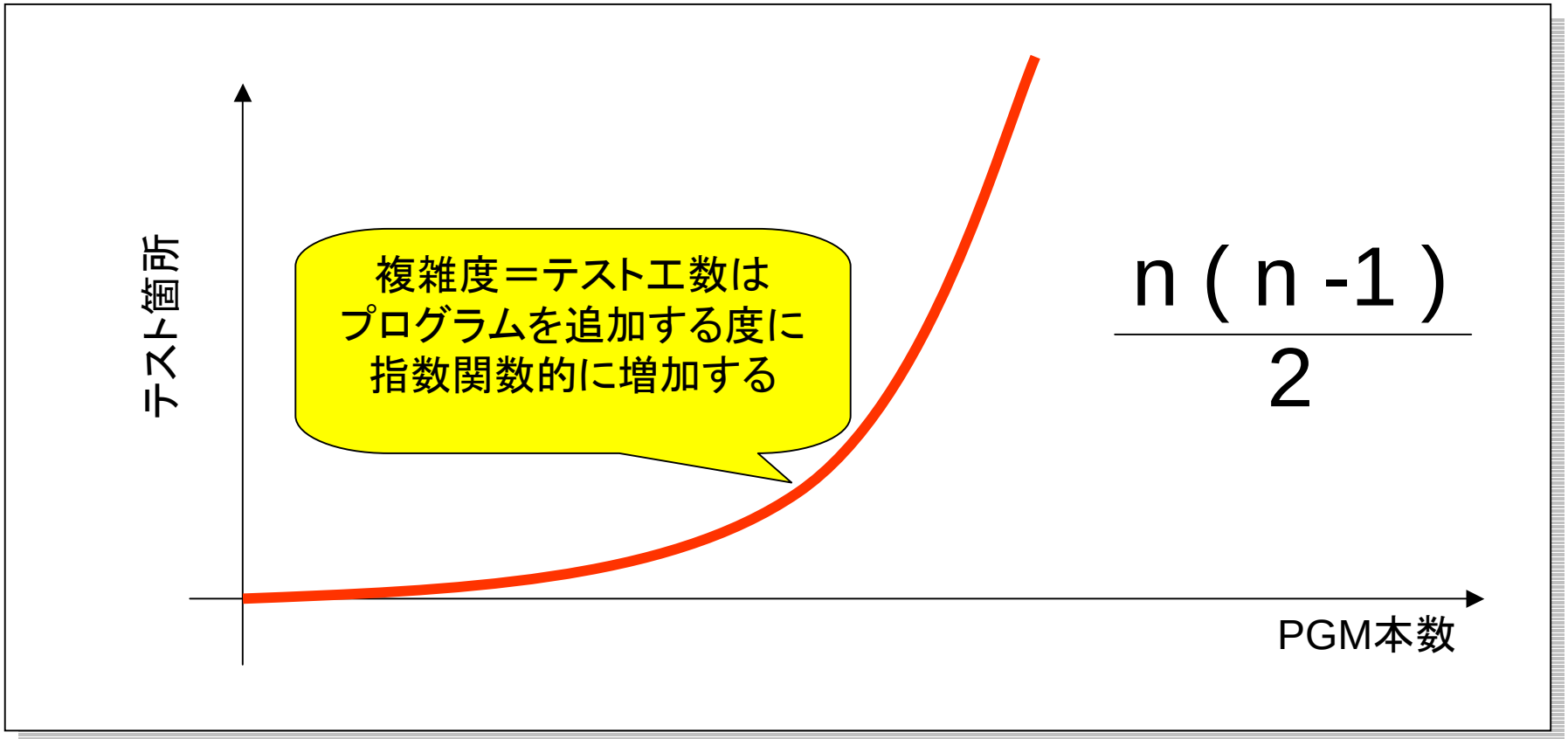


Chapter #1

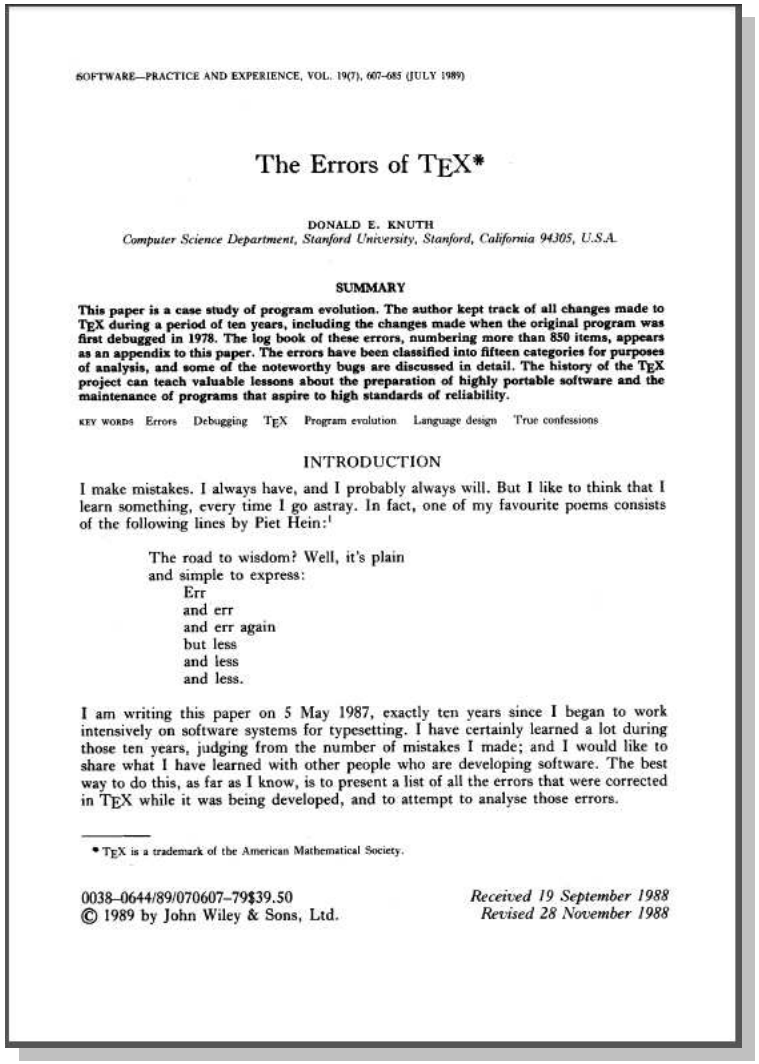
Dawn of Defectology
（欠陥学の黎明）

1976 : L.L. Lehman : リーマンの法則

「進化するシステムは特別な取り組みをしない限り、複雑さは上昇し続ける」という法則



1978～1989 : Donald Ervin Knuth : The Errors of TeX



SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 19(7), 607–685 (JULY 1989)

The Errors of TEX*

DONALD E. KNUTH
Computer Science Department, Stanford University, Stanford, California 94305, U.S.A.

SUMMARY

This paper is a case study of program evolution. The author kept track of all changes made to TEX during a period of ten years, including the changes made when the original program was first debugged in 1978. The log book of these errors, numbering more than 850 items, appears as an appendix to this paper. The errors have been classified into fifteen categories for purposes of analysis, and some of the noteworthy bugs are discussed in detail. The history of the TEX project can teach valuable lessons about the preparation of highly portable software and the maintenance of programs that aspire to high standards of reliability.

KEY WORDS Errors Debugging TEX Program evolution Language design True confessions

INTRODUCTION

I make mistakes. I always have, and I probably always will. But I like to think that I learn something, every time I go astray. In fact, one of my favourite poems consists of the following lines by Piet Hein:[1]

> The road to wisdom? Well, it's plain
> and simple to express:
> Err
> and err
> and err again
> but less
> and less
> and less.

I am writing this paper on 5 May 1987, exactly ten years since I began to work intensively on software systems for typesetting. I have certainly learned a lot during those ten years, judging from the number of mistakes I made; and I would like to share what I have learned with other people who are developing software. The best way to do this, as far as I know, is to present a list of all the errors that were corrected in TEX while it was being developed, and to attempt to analyse those errors.

* TEX is a trademark of the American Mathematical Society.

0038–0644/89/070607–79$39.50

Received 19 September 1988
Revised 28 November 1988

- 「アルゴリズムの神様」と呼ばれるドナルド E. クヌース博士がプログラム進化の事例研究として発表した1989年の論文。
- 組版システム「TeX」の10年に渡る開発を通じたプログラム進化と、開発中に得られた850項目のエラーログを分析し、15のカテゴリ分類を提示した最初の論文

Source: Software – Practice & Experience, eds. Cambell, J. and Comer, D., Vol.19, No.7, 1989, PP. 607-685

賢者への道とは
そう それは平凡で
言うのは簡単
誤りを犯し
そして 誤りを犯し
また 誤りを犯すこと

しかし その誤りを減らし
そして 誤りを減らし
誤りを減らすこと

Source: Piet Hein, *Grooks, MIT Press 1966*

The history of “Defect Engineering” #1 : The Errors of TeX

15種類のカテゴリー（最初のバグ分類）

A. アルゴリズムの不首尾（an algorithm awry)
B. 不手際または台無し（a blunder or botch）
C. 一貫性または明快さのための美文書化（a clean-up for consistency or clarity)
D. データ構造の崩壊（a data structure debacle)
E. 効率の増強（an efficiency enhancement）
F. 忘れられた機能（a forgotten function）
G. 一般化または能力の成長（a generalization or growth of ability）
I. 対話性の改善（an interactive improvement)
L. 言語の責任（a language liability)
M. モジュール間の不適当な組合せ（a mismatch between modules）
P. 移植性の増進（a promotion of portability）
Q. 品質の追求（a quest for quality)
R. 堅牢性の強化（a reinforcement of robustness）
S. 想定外のシナリオ（a surprising scenario)
T. ちょっとした誤植（a trivial typo)

The history of “Defect Engineering” #1：欠陥は進化する

欠陥に対する最初の所見

- 簡単に修正できるタイプの欠陥（**Level1**）
  - 正しく掛けていたが実際のタイピングで誤る（T）
  - 正しく考えていたが書くときに誤る(B)
  - 正しく知っていたが考慮するのを忘れる（F）
  - ツール（L）
  - 仕様の知識不完全（M）
- Level1より一段難しい問題 （**Level2**）
  - アルゴリズム欠陥（A）
  - データ構造上の誤り（D）
- 機能強化する派生で増加する欠陥（**Level3**）
  - 美文書化（C）
  - 効率性向上（E）
  - 一般化（G）
  - 対話性（I）
  - 移植性（P）
  - 品質（Q）

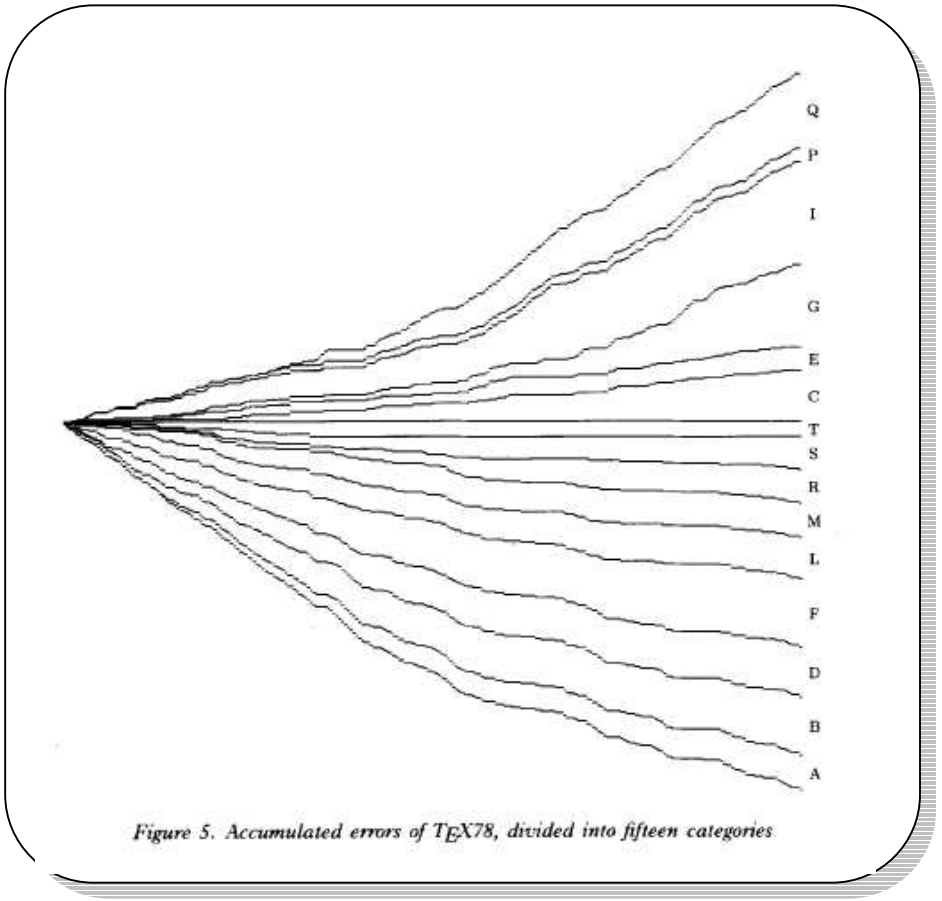


Figure 5. Accumulated errors of TEX78, divided into fifteen categories

TeX78のタイプ別エラー数累積

Boris Beizerによる分類（1983年）全欠陥数に対する欠陥分類毎の比率例

Beizerによる分類：番号付けと出現頻度に着目した分類

- バグの分類について述べたものである。当初，IEEEのバグの分類に関する標準化原案(ANSI87E)を採用しようとしたが，網羅範囲が不十分であったために断念した。
- ここでは，バグを4桁の数字に使ったポイントシステムで分類する。たとえば，「1234.1.6」である。このポイントシステムに拡張性を持たせるために「x」という記号を設ける。「x」は，今後バグの分類を拡張するときのための予備として用いる。たとえば，

3xxx	開発したソフトウェアの構造不良
32xx	処理不良
322x	数式評価不良
3222	演算不良
3222.1	誤操作

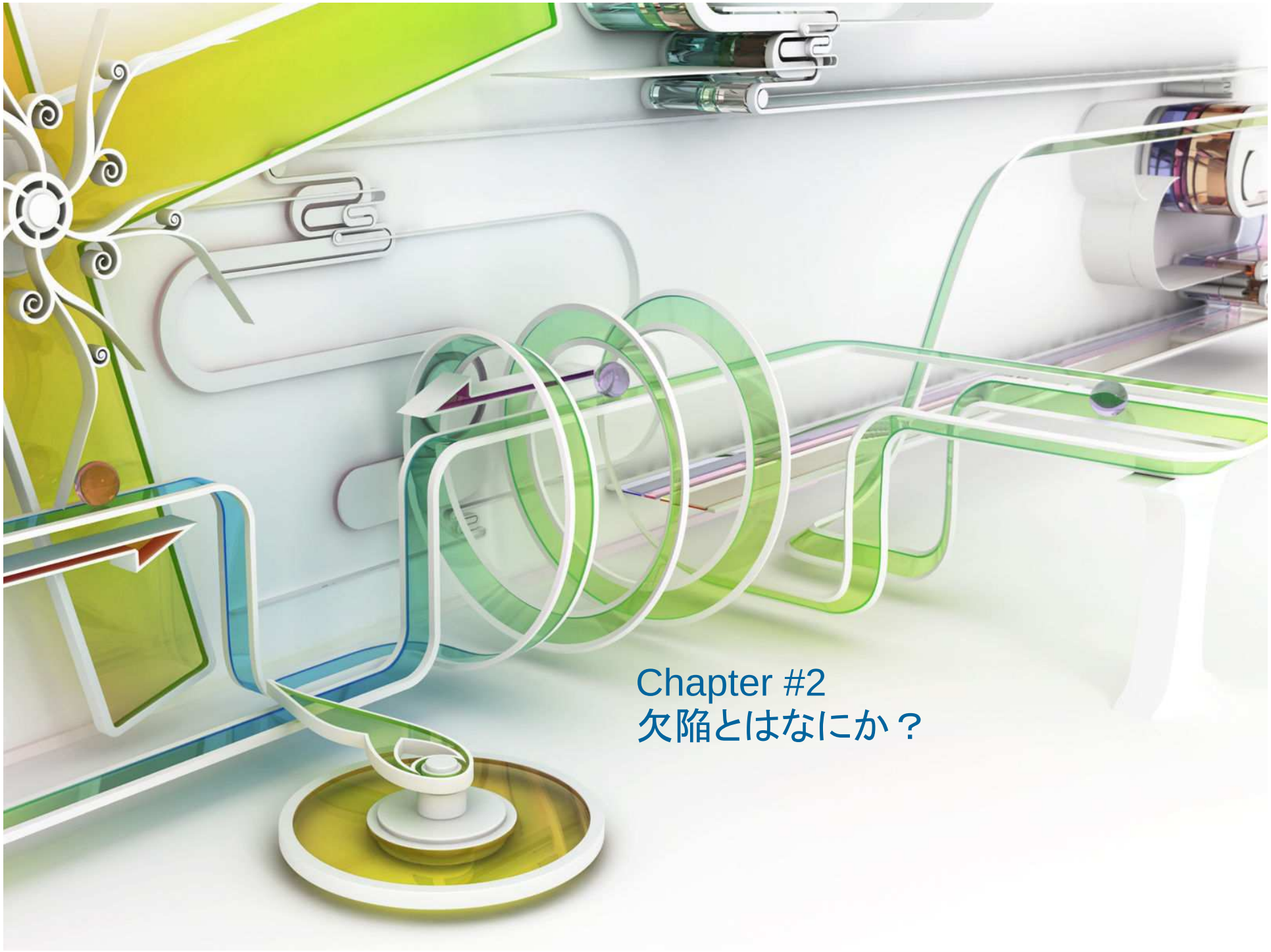
- それぞれの分類の最後のコードは，つねに9である。たとえば，9xxx，39xx，329x，3229，3226.9である。より詳細な適切な分類が出来ない場合に，ここに分類するためのものである。たとえば，以下のとおりである。

9xxx	:	未分類の「不良」
39xx	:	未分類の「構造不良」
329x	:	未分類の「処理不良」
3229	:	未分類の「数式評価不良」
3222.9	:	未分類の「演算不良」

- この分類では，いろいろ異なるさまざまな資料からデータを収集できるようにするために，このような数字によるポイントシステムを採用することにした。

Source: Boris Beizer, “Software Testing Techniques”, Van Nostrand Reinhold; 1990 ISBN 978-0442206727
「ソフトウェアテスト技法—自動化、品質保証、そしてバグの未然防止のために」ボーリス バイザー (著), Boris Beizer (原著), 小野間 彰 (翻訳), 山浦 恒央 (翻訳) , 1994, ISBN-13: 978-4822710019

事例）Boris Beizerによる分類（1983年）によって何がわかるか？



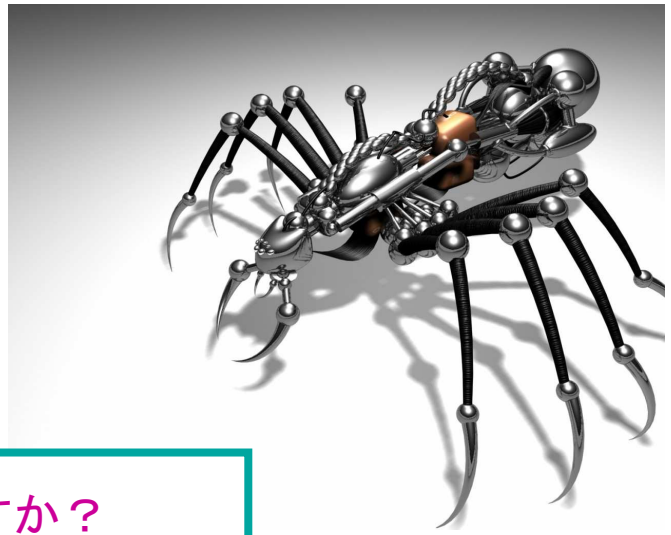
Chapter #2
欠陥とはなにか？

欠陥を測定・分析することは必要。しかし限界も存在

欠陥を定量的側面だけでは管理するには限界がある

- 欠陥は「消し去るもの」「あってはならないもの」「憎むべきもの」
- 今目の前にある欠陥を除去すればいい（=“討ち取る”もの）

→ 本当に除去するだけでよいのか？



そもそも定量的に欠陥が測定できた
時点は「混入後」＝ 手遅れに！！

組織で発生頻度の高い欠陥ベスト10があげられますか？

その頻出欠陥10種類のうち、再発確率が最も高いのは？

その10種類の欠陥のうち同一原因のものは？

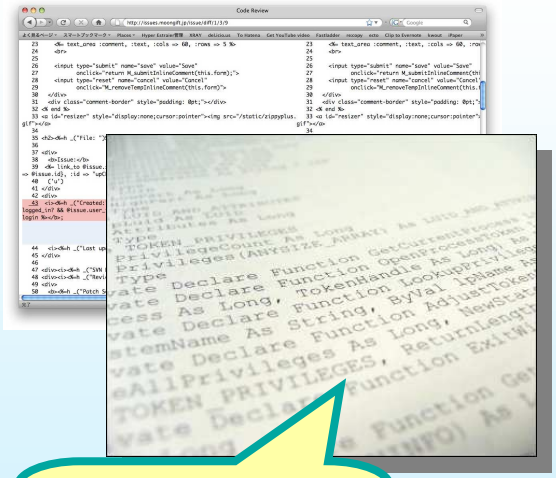
そもそも、欠陥とは何か？

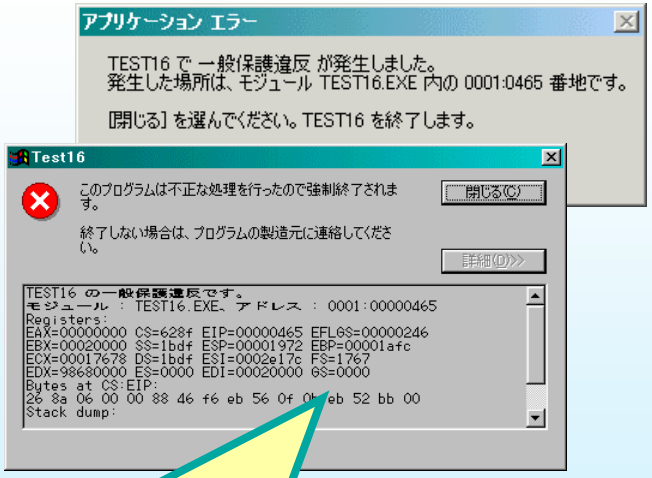
- メモリーリークの例：

State1：
コードに欠陥が
埋め込まれた状態

State2：
コードがメモリ上で
実行された状態

State3：
リソースが食いつぶされ
マシンが停止した状態





コード中にメモリを
解放するコードが
ない・抜けた状態

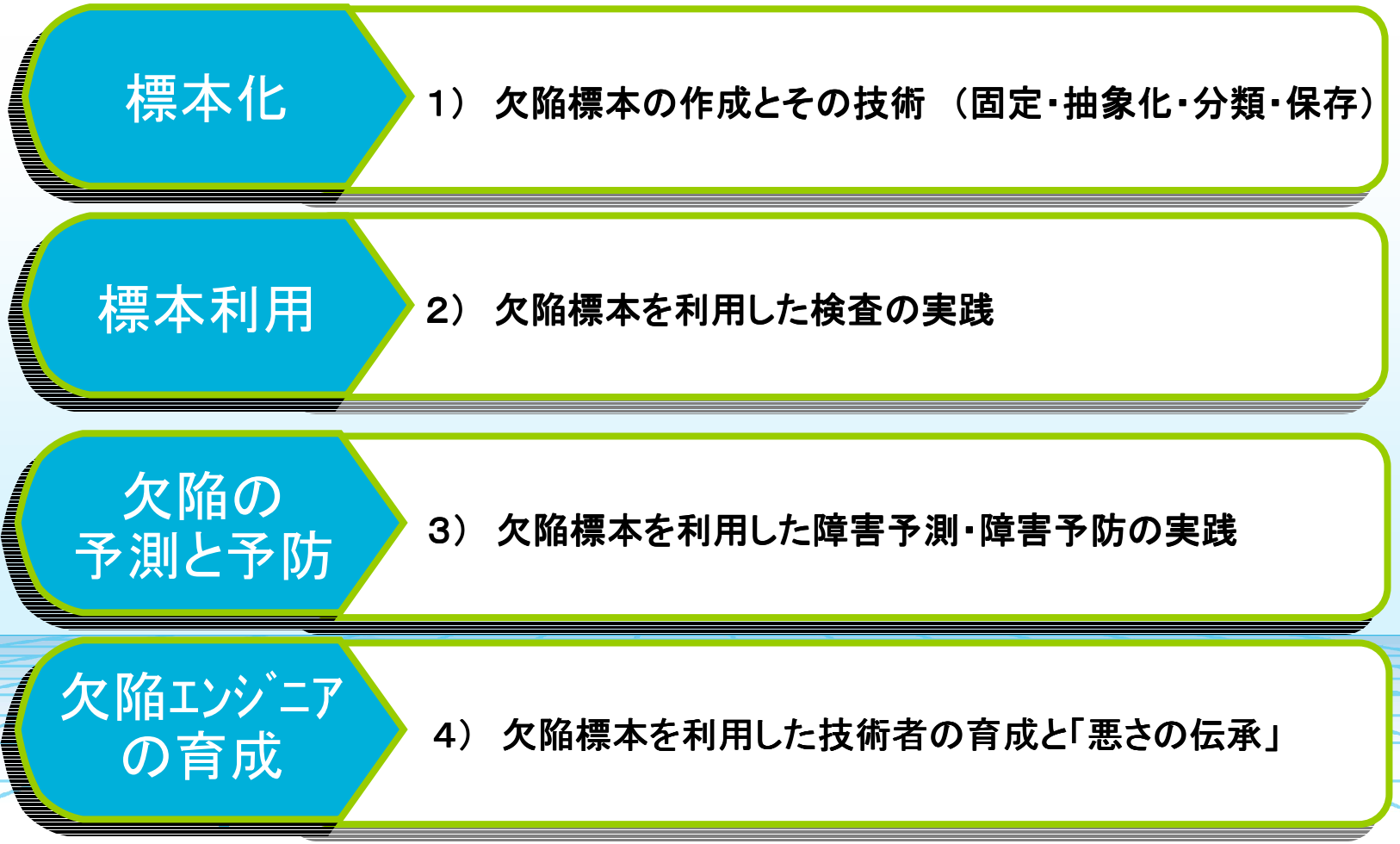
コードがメモリにロードされ
処理が終わり、わずかに解
放されないメモリが残る状態

メモリ資源が徐々に食いつ
ぶされ（＝リークし）、マシン
が停止する状態

欠陥に関する厳密な定義・ノウハウが伝承されていない典型例

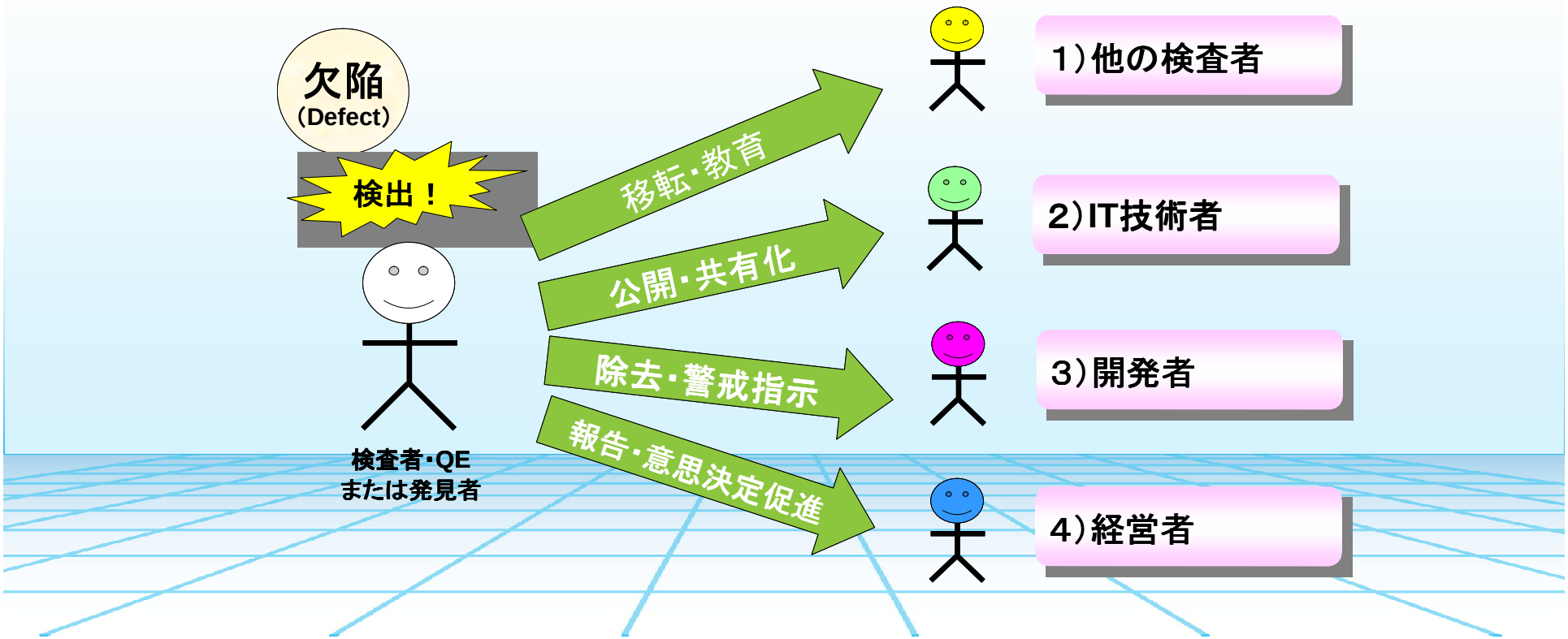
欠陥学とは？

欠陥学（Defectology）とは、欠陥の混入確認・存在確認や、各種ソフトウェア開発と検査を助けるための欠陥情報と標本の提供を目的とする研究です。



“悪さの知識” = 欠陥知識移転の重要性

- 欠陥情報は、誰かに伝達・伝承して初めて意味を持ちます。様々な「欠陥のユーザー」は、発見者が伝達した欠陥情報から利得を得ます



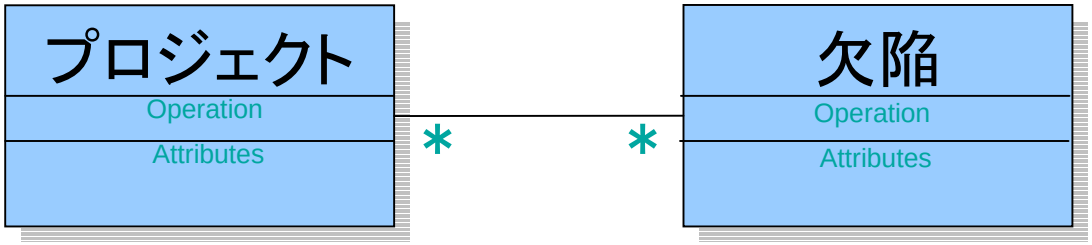
従来のバグ票でダメな理由、この研究をしているか

1）プロジェクトで発生する個々のバグ票は、プロジェクト固有条件を含むため欠陥の「インスタンス」である

2）現場プロジェクトでいつでも観察・参照可能な欠陥情報として、欠陥マスターの存在が必要不可欠

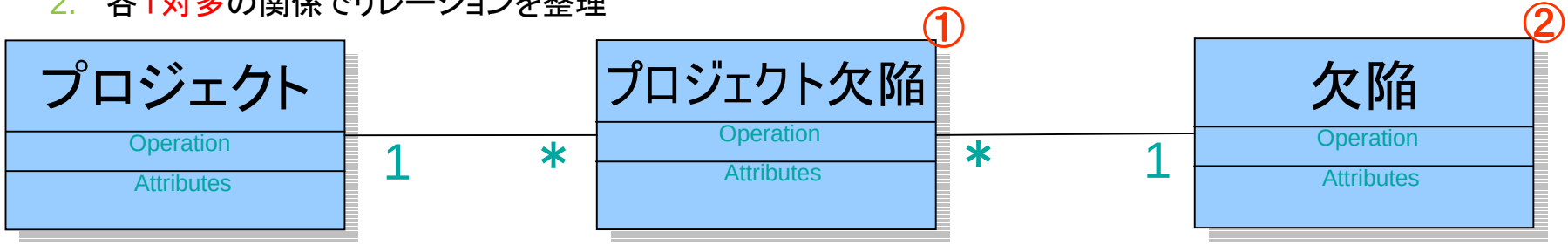
プロジェクトと欠陥の関係を整理すると次のような関係。

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



上記多対多の関係は、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対多の関係でリレーションを整理



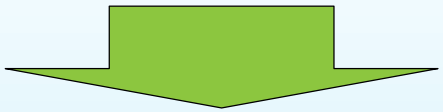
- プロジェクト欠陥（①）をどれほど多量に集めても、観察・参照・利用可能できない
- 欲しいのは②の「欠陥マスター」情報。固定化・抽象化・整理されなくては参照できない

欠陥情報の記録 ： 従来の欠陥記録とマスターDBの必要性

- 従来、開発中・改変時に関わらず「バグ報告書」として欠陥情報が記録されてきました。
  しかし、本来は欠陥をインスタンス情報として捉えてはいけません。

従来の欠陥記録 = バグ報告書

ソフトウェアのライフサイクルにおける特定の品質特性の
一静的断面をあらわす情報の塊



将来の欠陥記録 = 欠陥のマスター情報記録

不連続に変化するソフトウェア品質において、期待と現状の
GAPが大きくなった時に欠陥を固定化・抽象化・整理して保存
し誰かに改善・受容を促すための情報群

参考） 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

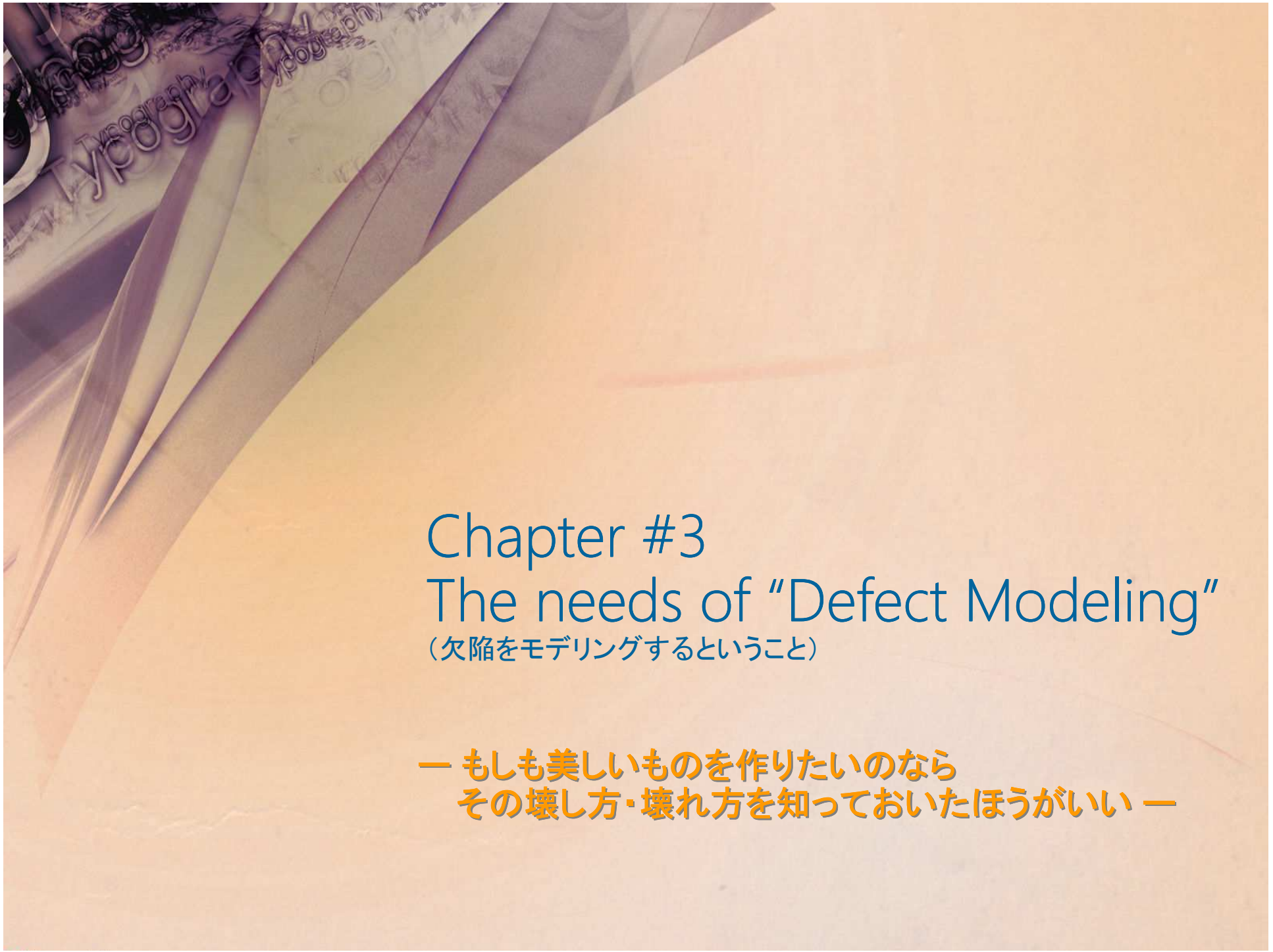
IT業界には．．．

- ✓ 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.

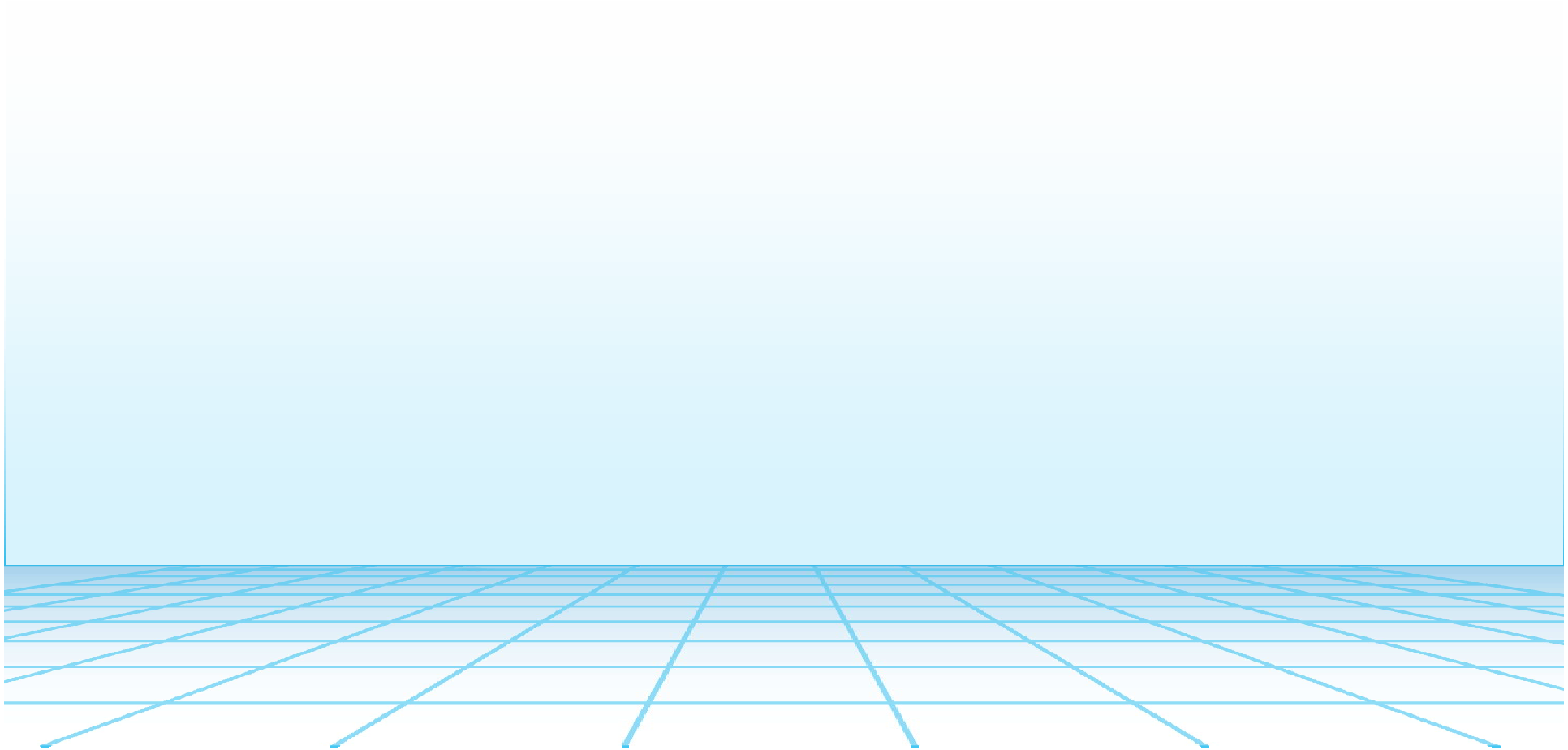
医学分野で世界最大の文献データベース。**1966**年からNLM（米国国立医学図書館）でデータ収集が始まり，現在**毎月約3万件の文献**が新たに追加される。現在では，米国を中心に約70ヵ国から，**900万件**を超える文献が収録される。



Chapter #3
The needs of “Defect Modeling”
（欠陥をモデリングするということ）

― もしも美しいものを作りたいのなら
その壊し方・壊れ方を知っておいたほうがいい ―

例題1）例外処理が抜けている



例題1）例外処理が抜けている　：　代表的な現象

1.コメントが一切記載されていない形式。完全に空欄

2.「処理なし」「NOOP」という記述。処理がない

3.例外処理が記載された後コメントアウトされている

パターン1）時間がないから正常ケースのみ

- コメントが一切記載されていない形式。完全に空欄

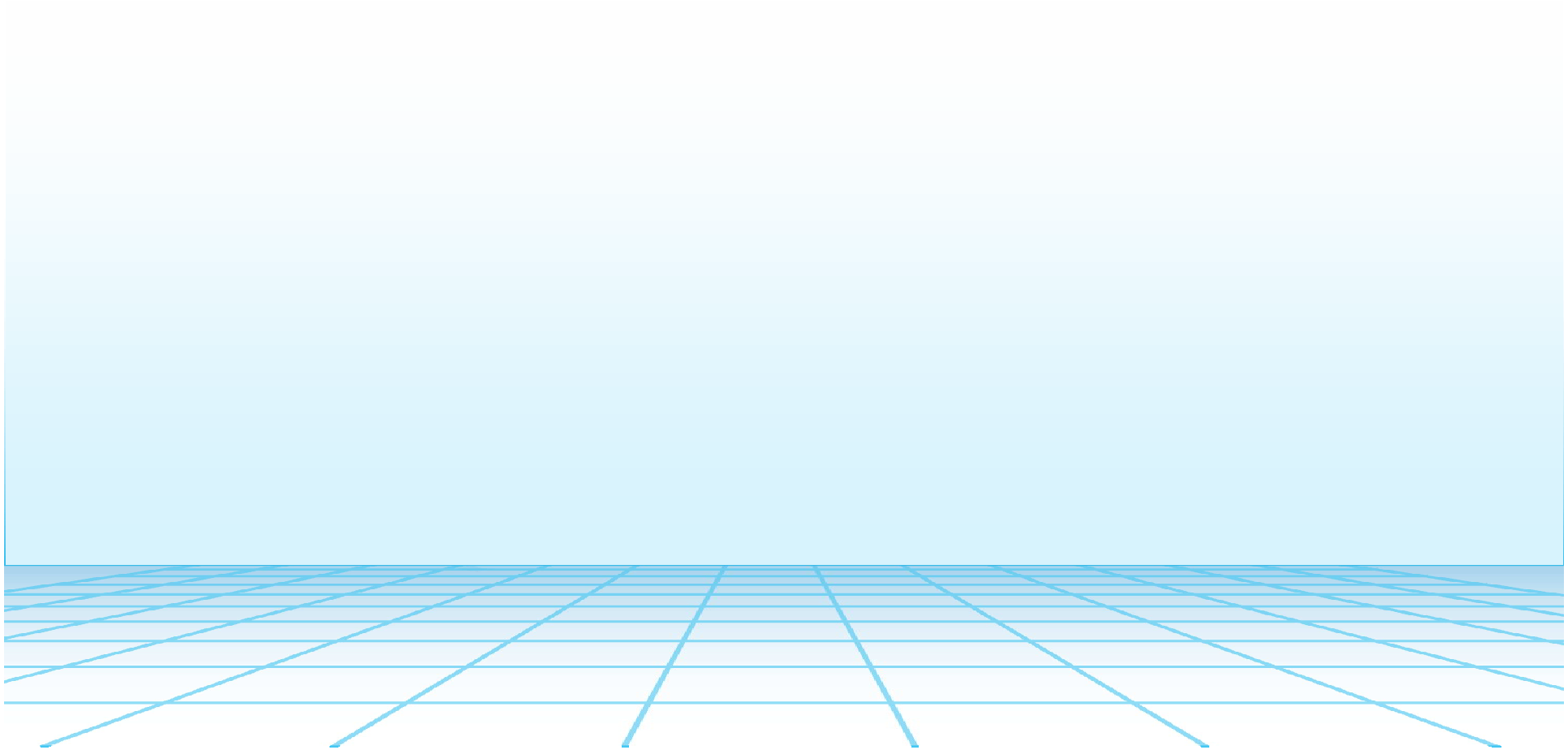
典型的な例1：

```
16:         } catch (Exception e) {
17:         }
```

典型的な例2：

```
16:         } catch (Exception e) {}
```

原因：多くの場合「時間制約」が原因



パターン2）「しようがない」または「負荷移転」

- 「処理なし」「NOOP」という記述。処理がない

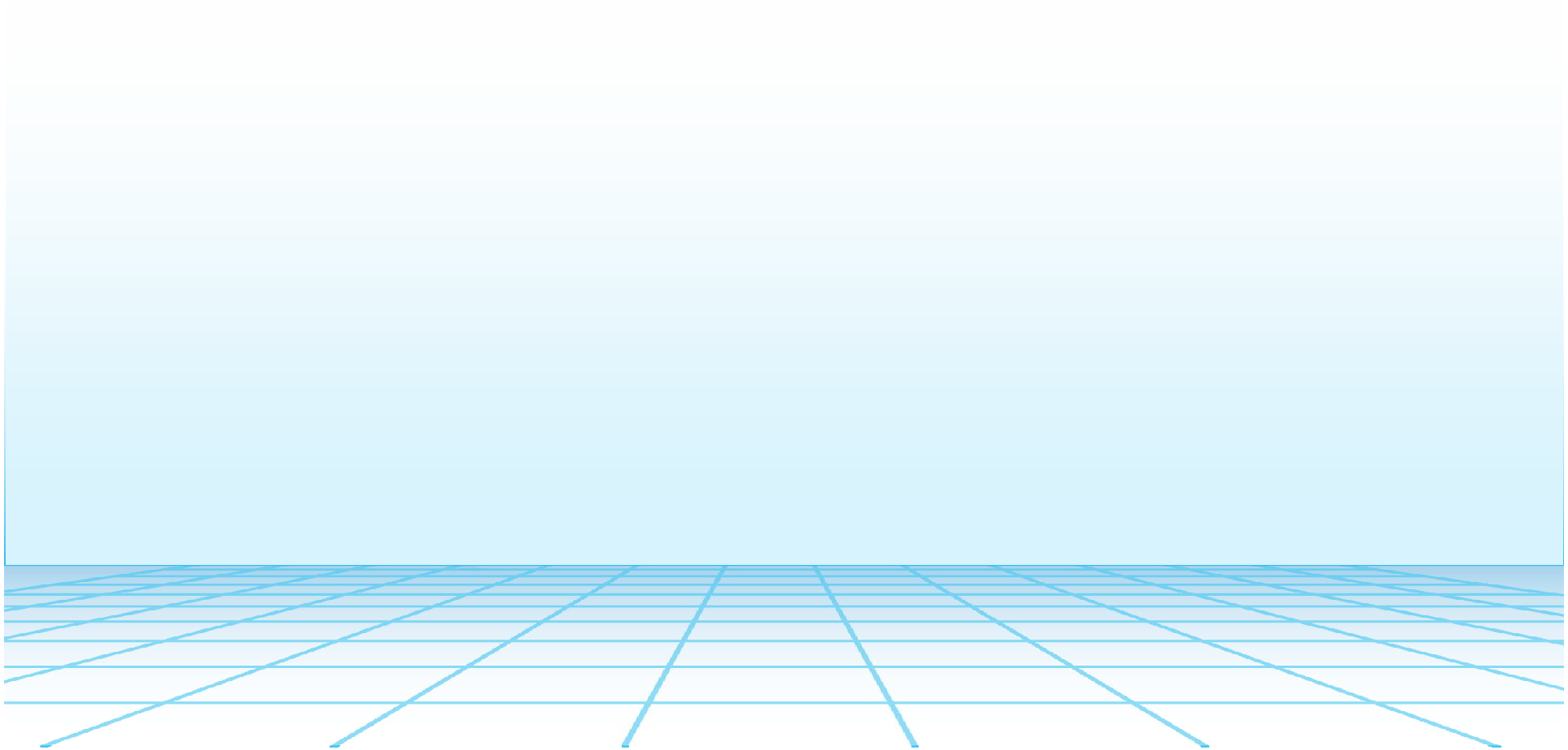
典型的な例1：

```
16:         } catch (Exception e) {
17:                 // 処理なし（NOOP)
18:             }
```

典型的な例2：

```
16:         } catch (Exception e) {
17:                 // ここには制御がこないハズ
18:             }
```

原因：部分最適化・責任範囲・「言われたことだけ実装」



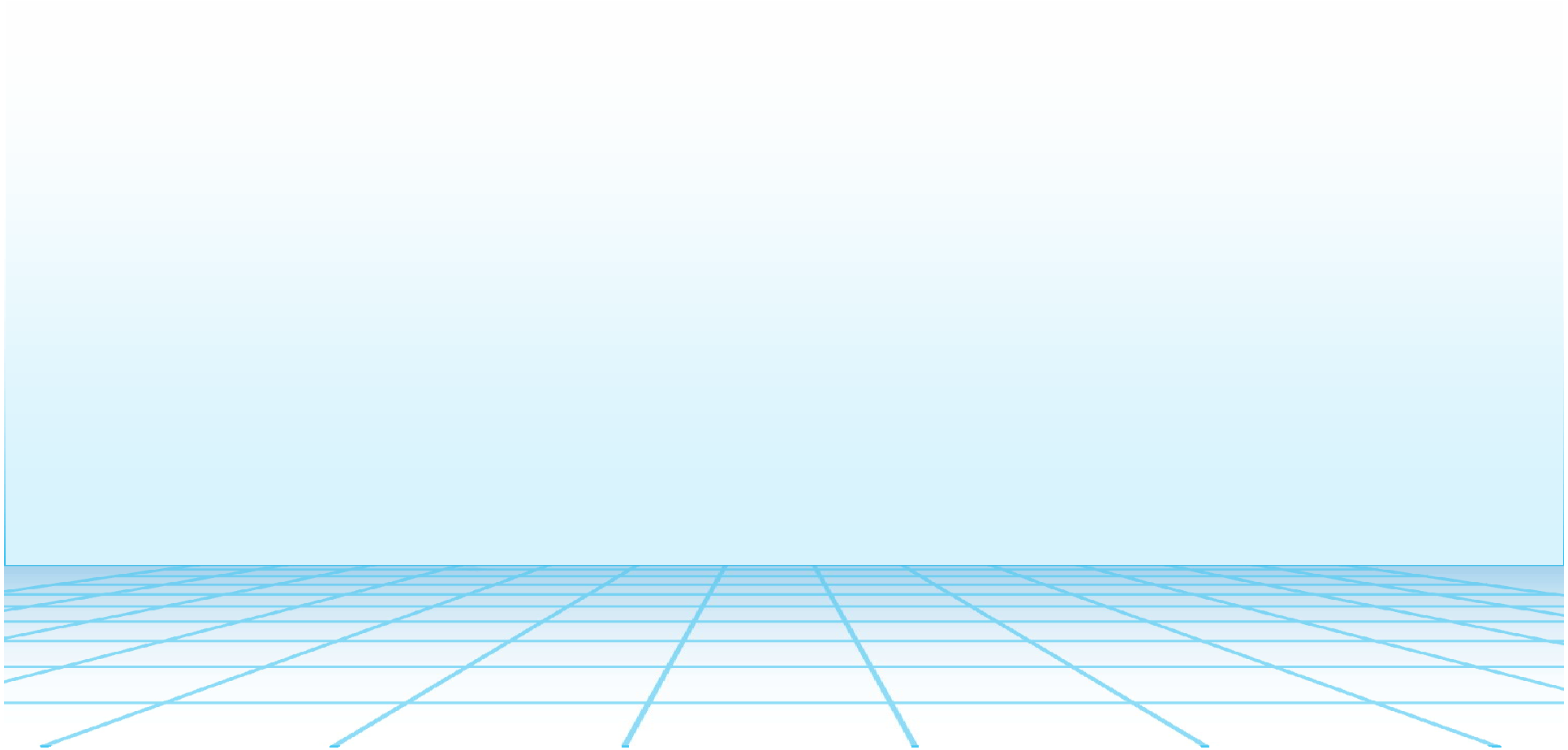
パターン3)「やっぱやめた」「努力は認めてもらおう」

- 例外処理が記載された後コメントアウトされている

典型的な例1：

```
16: //         } catch (Exception e) {
17: //    Logging(msgID, message.toString(msgID));
18: //    fis.close();
19: //    // ファイルクローズに失敗した場合、エラーRaise＋表示
20: //    addActionError(e);
21: //    showError();
22: //         }
```

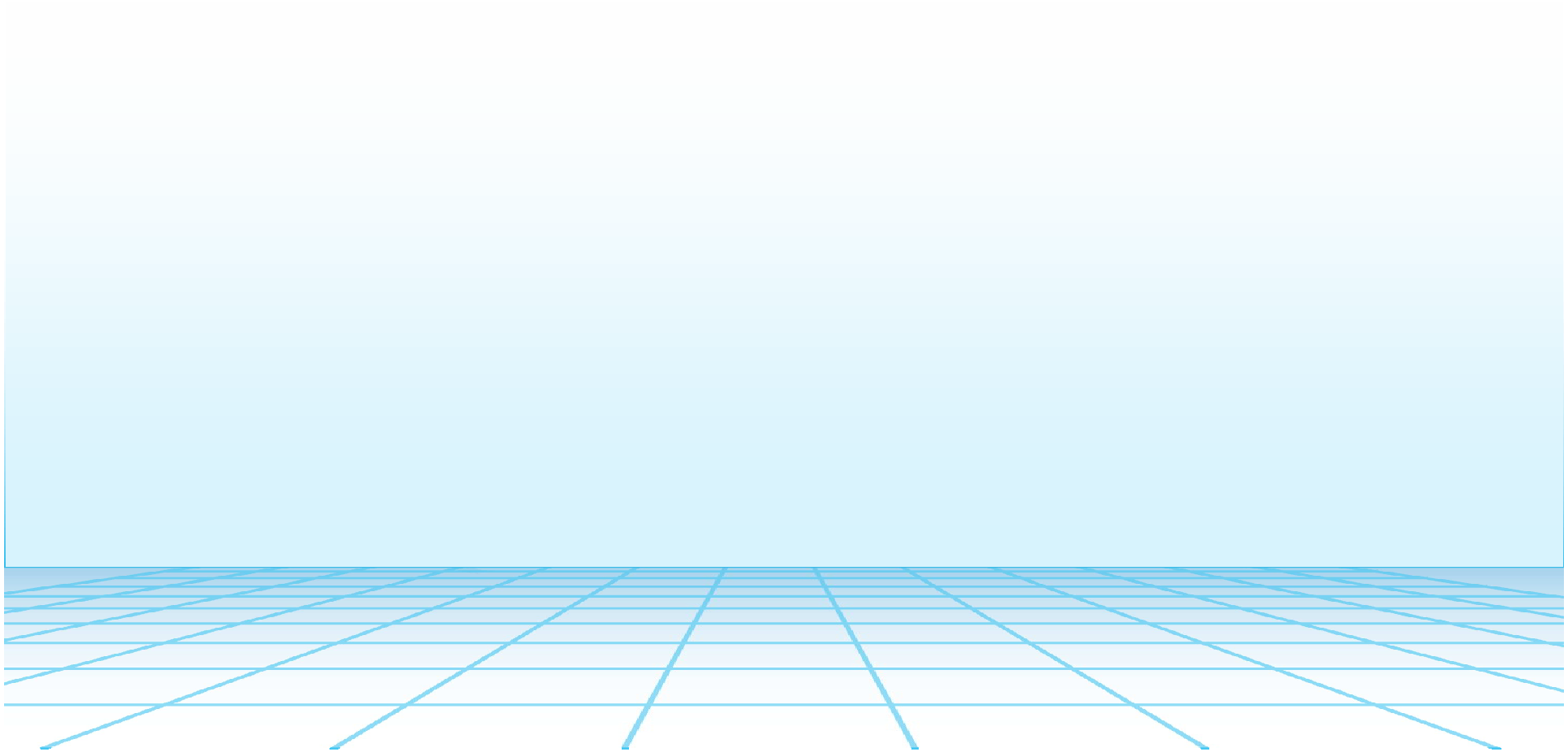
原因：圧倒的な時間不足+短期的な関係



例題1-a）例外処理が抜けている

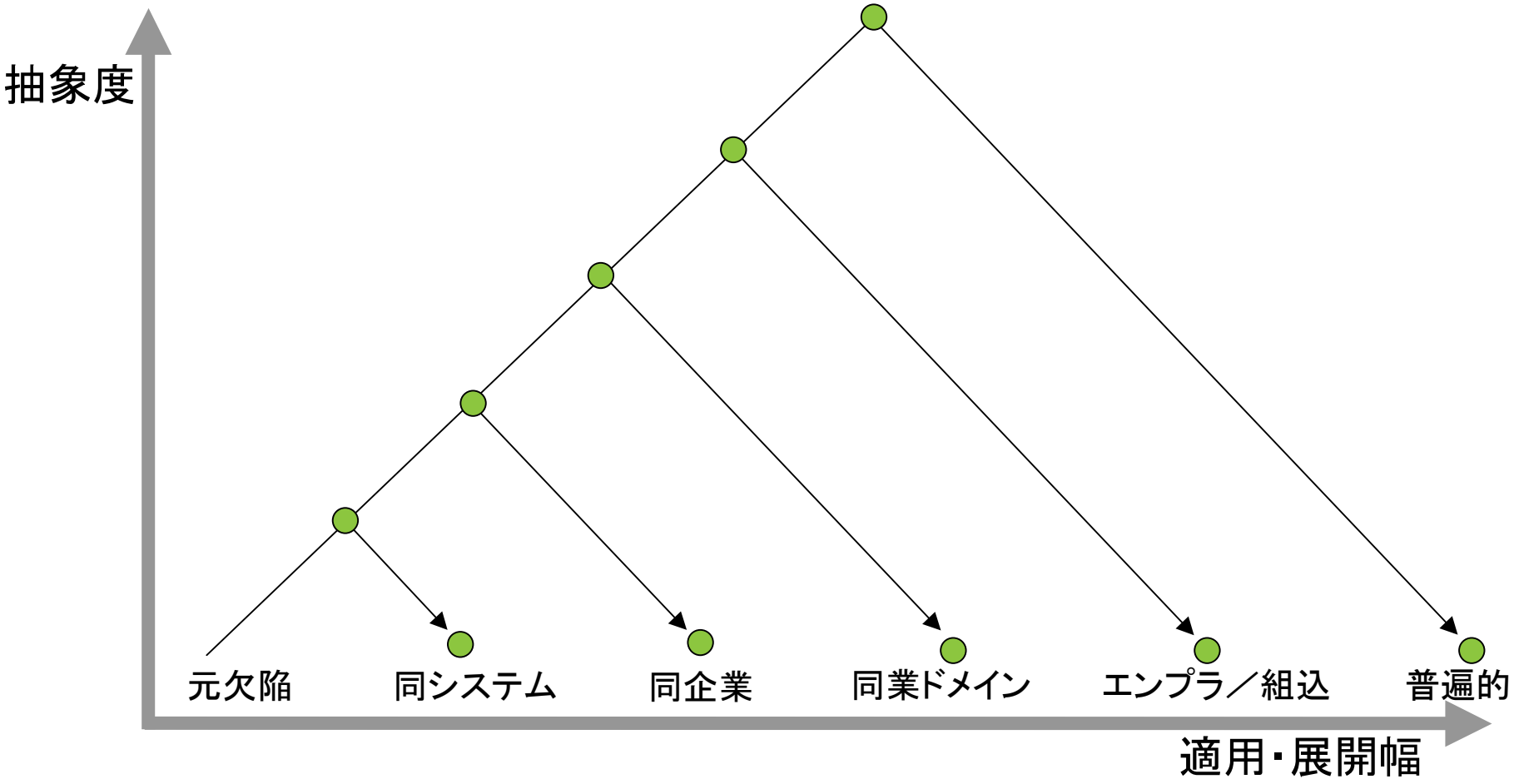
```
11:       } else {
12:         try {
13:           //log.debug( "REMOVE_SESSION--->" +  paramString);
14:           // その他操作の場合、Sessionに値があれば削除する
15:           this.session.removeAttribute(paramString);
16:         } catch (Exception e) {
17:                 ;
18:         }
19:       }
```

パターン3）の変形：「例外の握りつぶし」または「怠慢」



欠陥の抽象度と展開幅

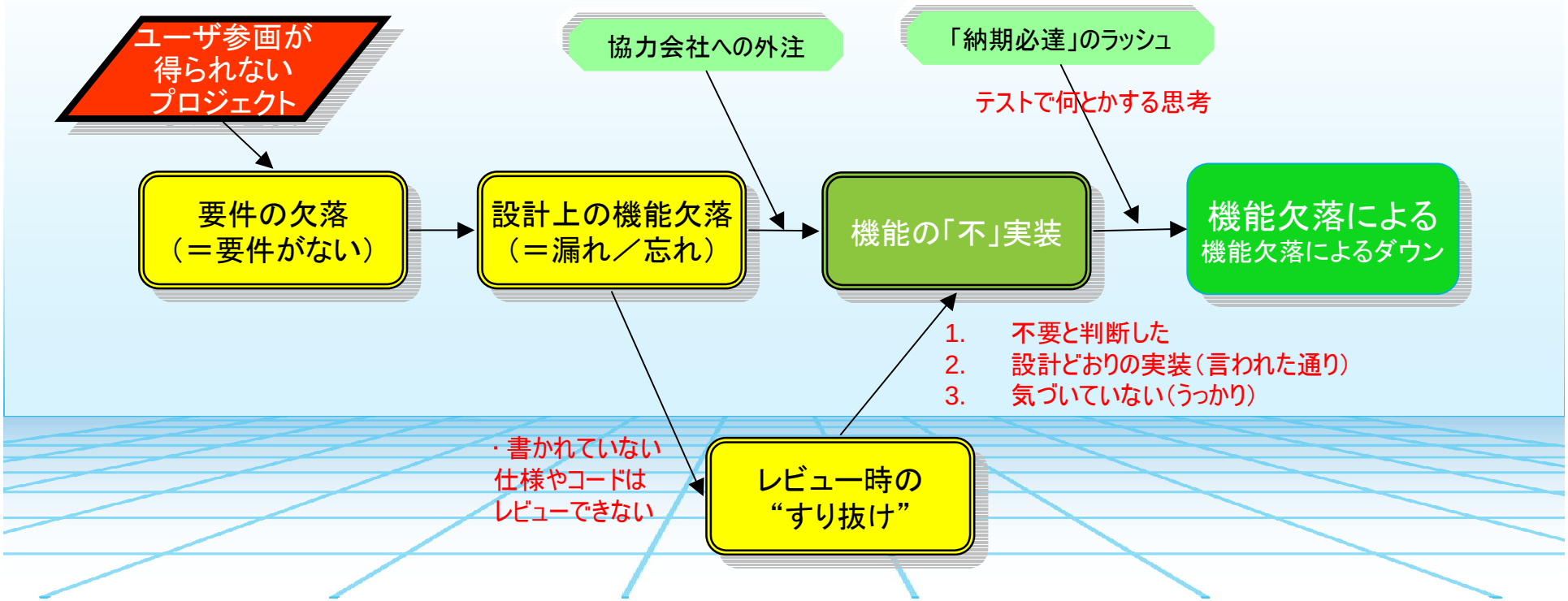
- 欠陥保存時の情報抽象度の高さに依存して横展開幅（＝欠陥情報の適用範囲）が決定される
- 逆に広範囲な展開を狙わない場合には、抽象度を高めなくてもよい（←本当？）



パターン認識とフレーミング

- 医学の世界で「誤診」を避けるための方法として「フレーミング・バイアスの考慮」があります。フレーミング・バイアス（枠への偏執）とはある特定の特徴／兆候のみに固執して、重大欠陥（＝医学の場合は病気）を看過することです。

- 欠陥マスター情報に登録される情報は十分に抽象化された欠陥情報の「枠組み」すなわちパターンの集合体です。このフレームに固執することは、一般的でない・稀な症状を呈する重大な欠陥を見過ごすリスクがあります。
- 品質エンジニア（レビュー担当者／テスト担当者）が品質活動を行うためには、欠陥マスター情報や欠陥モデルの持つ典型的な症状に固執することなく、広く中立的な視点での品質検査が必要です。



Chapter#3

Defects are also artifacts

(欠陥もまた人工物である)

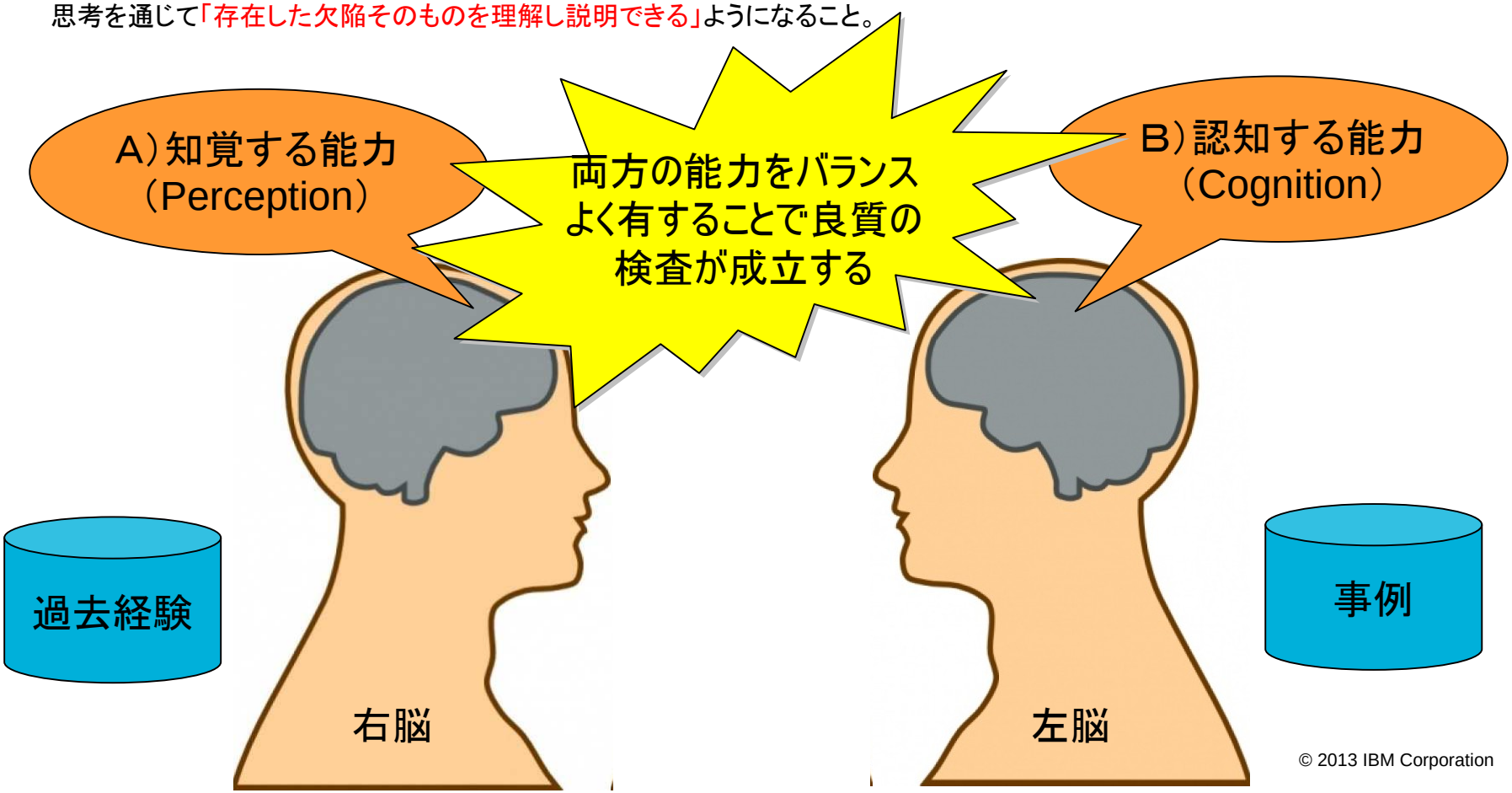
欠陥を検出する能力：知覚と認知

A）知覚する能力（Perception）：

観察（ぼんやりと全体を俯瞰すること）や洞察（ある程度細部を注視）すること、系統立てた点検作業を通じて
「欠陥がそこに存在すること」を感覚的に対象を捕らえること。経験的学習や推測・過去パターンとの照合によって実現。

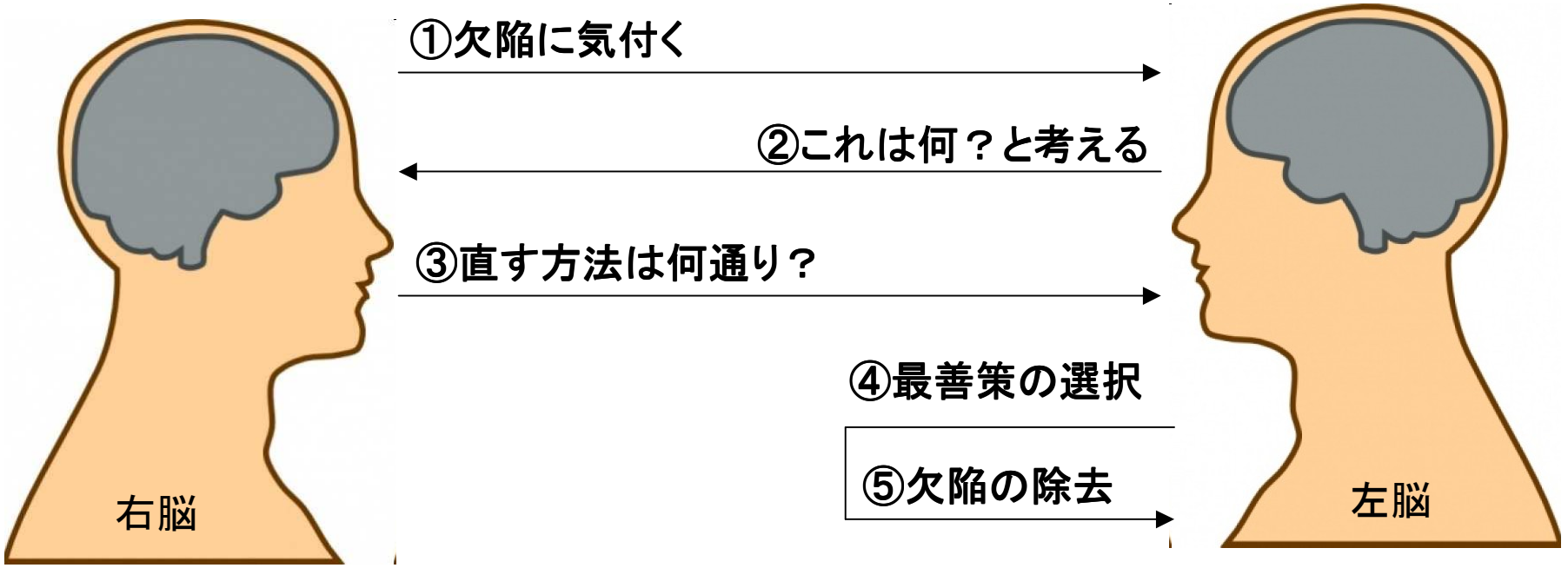
B）認知する能力（Cognition）：

知覚した欠陥の意味づけ、所見を確率する作業等を含む分析作業（種類や位置）。特に異常性や特異性を検出する際に論理的思考を通じて「存在した欠陥そのものを理解し説明できる」ようになること。



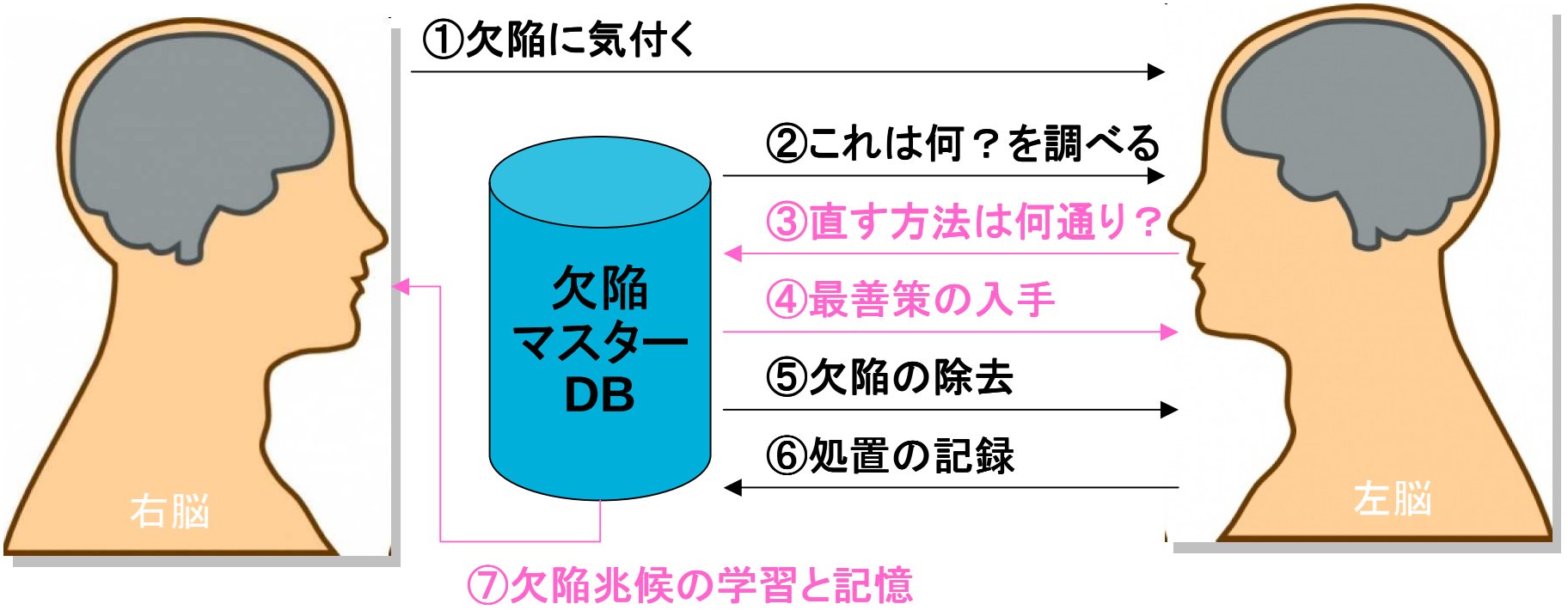
Chapter3: バグ検出する際の脳の動き

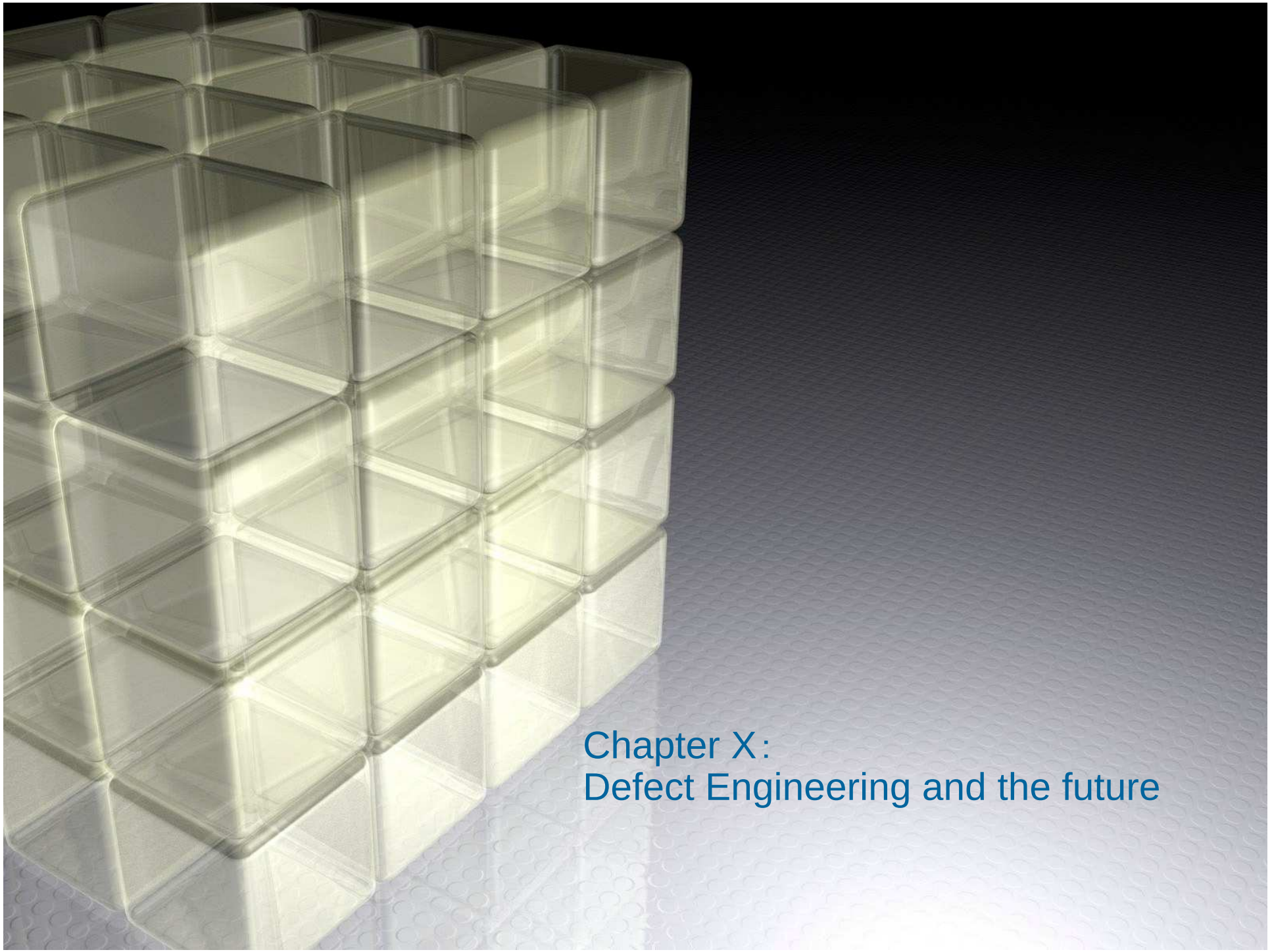
- 一般的な開発者は日々の活動（テスト、レビュー、デバッグ、その他）の中で、「バグの存在に気付く」瞬間があります。最初に欠陥に気付く瞬間は多くの場合、右脳が認知します。
- 次にこの欠陥を論理的に「これは何のバグ？」「こんなバグがどうしてここに？」とその原因を考えますが、その真因考察の結果が得られる・得られないに関わらず、いずれかの脳で「除去方法」を考察します。
- そして選択した最良（であると思われる）除去方法を実践し、除去できたことを再度確認します。これはほぼ左脳で論理的に除去していくと思われます。



Chapter3: バグ検出する際の脳の動き

- 前頁図の「③直す方法は何通り？」のところで、開発者個人の過去経験に基づいて欠陥の除去方法を感覚的に選んでいました。
- これを「欠陥のマスターDB」を使って論理的かつ確率論的に左脳で考察することができれば、更に欠陥の検出効率を高めることができるばかりでなく、①の欠陥に気付く力（欠陥兆候情報を追加）を強化したり、③で他者の除去方法（欠陥除去情報）を利用し、長期的には混入を防ぐことができるようになります。





Chapter X:
Defect Engineering and the future

DEBOK & DefectEngineering.NEXT

- Review / Inspection / I-V&Vs
- Process / Cost of Quality
- Quality Metrics
- Carrier of Quality Engineer
- Defect Engineering

- Defect Engineering
- Worldwide standards
- Open source?
- Socialize?

医学的アプローチ::免疫メカニズムの実装

- 免疫のメカニズムはAD1000年前後には「天然痘」のかさぶたを乾かしたものを経口摂取することで、能動的免疫処置の原型を実現していた

病原体採取

分析・観察

培養

投与

抗体増により
病気予防

欠陥採取

分析・観察

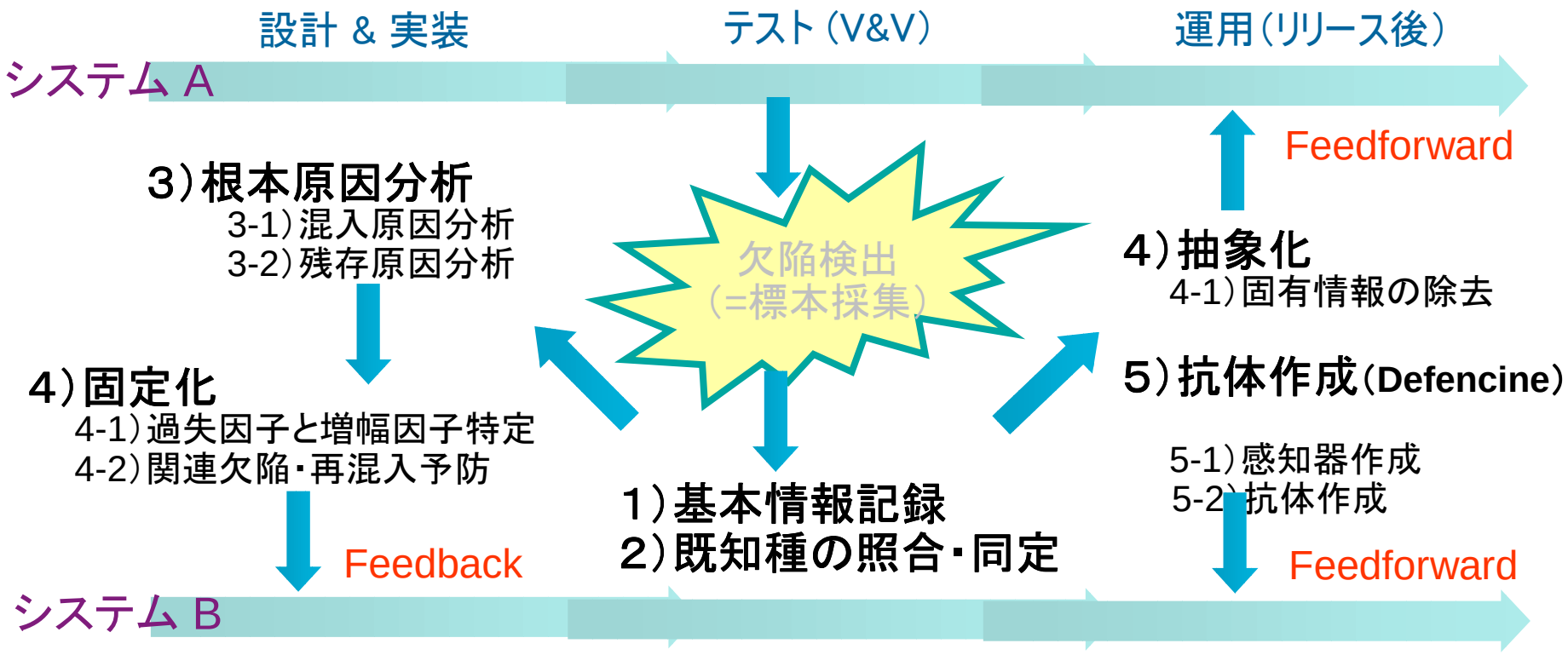
培養

抗体注入

既知種欠陥を
含むコードの
リリース予防

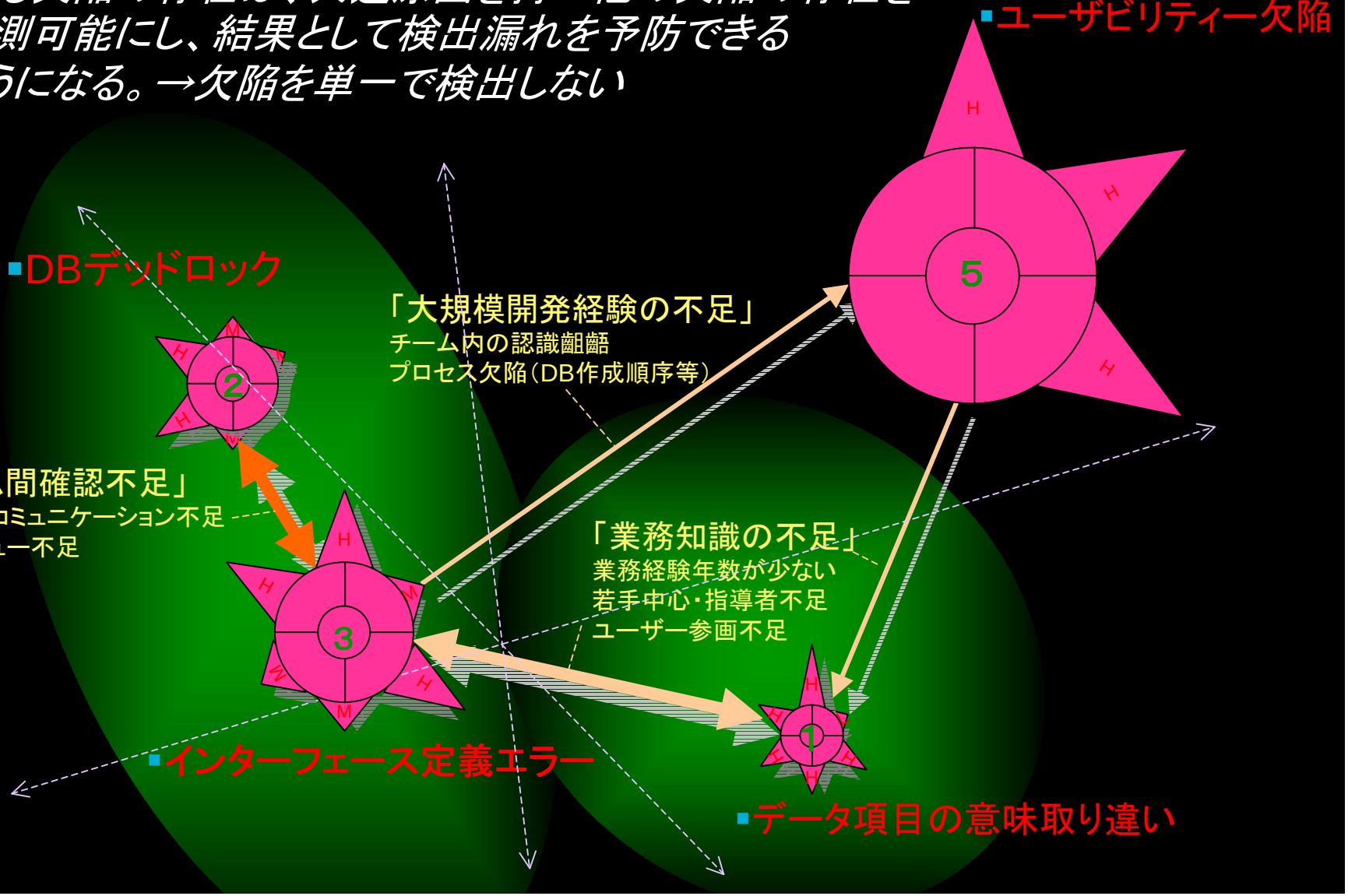
医学的アプローチ::免疫メカニズムの実装

- 検出した欠陥を採集すると、抽象化／固定化を経てフィードバックとフィードフォワードのプロセスを実施。他システムに「欠陥ワクチン（感知器と抗体）」を投与することで、同種欠陥の再発と新規の混入（＝コードリリース）を予防する

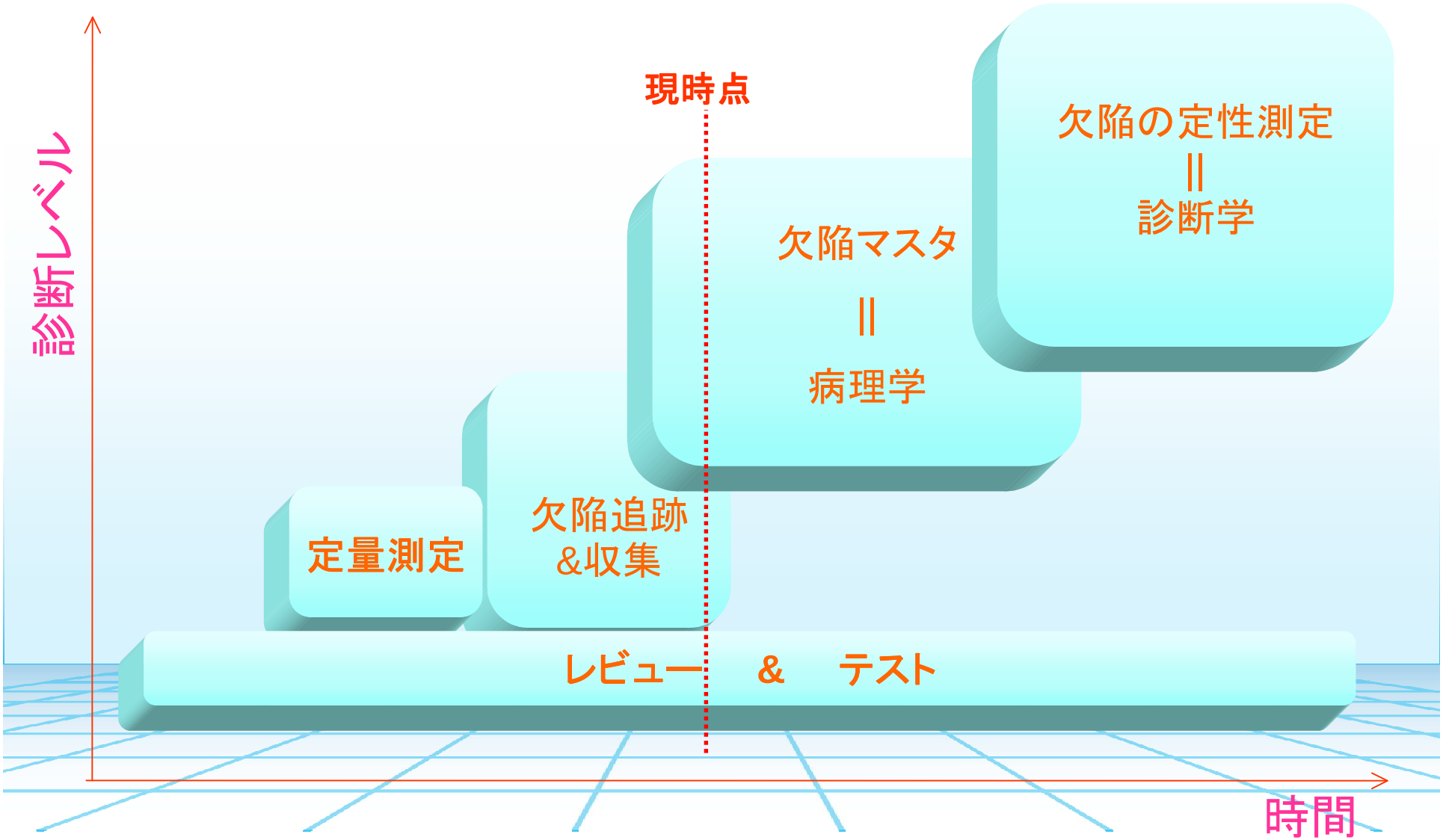


医学的アプローチ::併発症の研究

1）ある欠陥兆候を検知できると、一つの欠陥の存在が予測できる
2）ある欠陥の存在は、共通原因を持つ他の欠陥の存在を
予測可能にし、結果として検出漏れを予防できる
ようになる。→欠陥を単一で検出しない



医学的アプローチ::診断学への応用



ソフトウェアの歴史が進化しない理由は、欠陥を研究しないから

To be continued …..

ソフトウェアテストシンポジウム
JaSST ージャストー Japan Symposium on Software Testing

Thank you ☺