KKD（勘、経験、度胸）も使いよう。
一歩進んだレビュアーになるために。
～ レビュー眼 ～

㈱日立システムアンドサービス
角口 勝隆
（JaSST Kansai 実行委員）

自己紹介

・（株）日立システムアンドサービスで、主にパッケージソフト および ソリューションサービスの品質保証業務に従事。（職種は、いわゆるＱＡ）

・優れたレビュアーの「ヒラメキ」は何故おこるのか、そのノウハウを他人へ伝授することはできないのか検討を始め、JaSST Kansai実行委員会内でブラッシュアップを受けて、今回の発表に至る。

Kadoguchi Yoshitaka
角口 勝隆
E-Mail:y-kadoguchi@hitachi-system.co.jp

◆「レビュー眼」とは

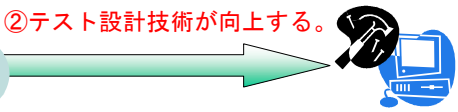
レビュアーの持つ秘訣（ノウハウ）
・心得
・観点、着目点
・思考法（発想法）

ここでは、
優れたレビュアーの持つ秘訣（ノウハウ）を
「レビュー眼」と称し、
その内容について、考察します。

※大辞泉によると「眼」には「物事を洞察する能力」「大事なところ。かなめ。」という意味合いがあるようです。

◆「レビュー眼」を極めれば、

①プロジェクト全体の品質リスクを、
事前に予測できる。
②テスト設計技術が向上する。



レビュー眼で
不具合（リスク）を先読み
（レビューとテストは、補完関係）
先読みした不具合が
存在しないことを、
テストや監査で実証。

不具合の作り込みを防止

・レビュー時の着眼点を実機検証
・不具合が、存在しないことを確認

早期に「品質」を確保することで、
コスト／納期が予定通りに推進され、
プロジェクトを成功に導くことが可能。

アジェンダ

・レビューポイントの考察
・レビューにおける改善課題
・優れたレビュアーのノウハウについて
　①怪しいポイントを押さえるコツとは？
　②的確に不具合を摘出できるコツは？
　　仕様の矛盾・抜けを見破るコツは？
・レビュー向け発想法について
　①定型連想型
　②エラー推測型
　③自由発想型
・「レビュー眼」の効果と、今後の展望

レビューの定義について

レビュー（Review）：批評、再検討、再確認
Re(再)-View(見)：つまり、見直し・再確認という意味合いでしょうか。

レビューの目的
①埋め込まれた不具合の摘出
②埋め込まれる不具合の予防
上流工程から発生する不具合を除去することで、
「品質を作りこむ」ことを目的とする。

**レビューの考え方は様々ですが、
ここでは、プログラムを動かす前に、
再確認を行うものと定義します。**
いわゆる、デザインレビューを対象とします

レビューの重要性について

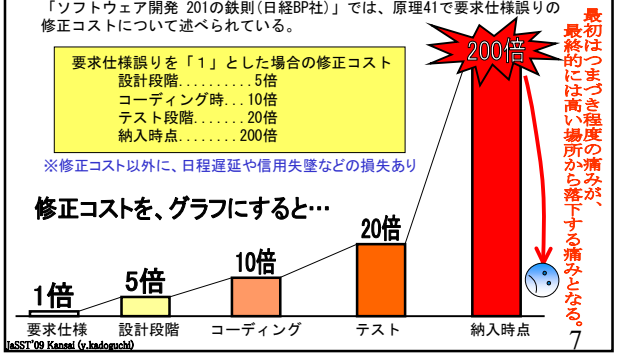
レビューによる問題摘出の、コストへの貢献度について

「ソフトウェア開発 201の鉄則(日経BP社)」では、原理41で要求仕様誤りの修正コストについて述べられている。

要求仕様誤りを「1」とした場合の修正コスト
設計段階.....5倍
コーディング時...10倍
テスト段階.....20倍
納入時点.....200倍

※修正コスト以外に、日程遅延や信用失墜などの損失あり

修正コストを、グラフにすると…



レビュー効果が上がらない現場の例

「Aシステム」の開発現場を題材に、一例を挙げる。

◆形式的なレビューになっている

機能仕様レビュー結果報告：
- 100ページあたり、30件の不具合を指摘しました。（指摘内容は、誤字脱字，様式不正などばかり）
- 40項目のチェックリストに従い、全て確認し、問題はありませんでした。

⇒ **本当に大丈夫なのか??**

◆不具合を見逃した反省は、あまり活かされていない

社外不良反省会：
- そのような使われ方は想定していませんでした。
- テストケースが思い浮かびませんでした。
- 仕様書に記載がなく、テストが漏れました。

⇒ **何故、レビューやテスト設計時に思い浮かばない?**

※再発防止策は、作り込み側へ求めることが多い。
※チェックリストを強化するだけでは、確認項目が膨大になり、結局見なくなる。

レビューで問題を見逃す主な原因:

①着眼漏れ(見ようとしない、気付かない)

②思考の偏(かたよ)り(思い込み、思い違い)

※思い込み：指示内容の制約等に、気付かない
※思い違い：知らない または 間違った確信。

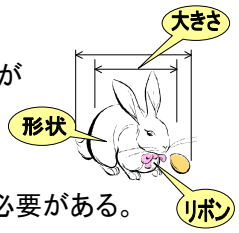
- ソフトウェアに対する「要求」や「仕様」は、目に見えない。
- 仕様書があるといっても、完全な情報を得る／伝えることは難しい。

いわゆる、「行間を読め」ということ?

見えないものを、どうやって読む??

例えばですが、、、

隣の部屋にいるなど、右の図が見えない人に、ウサギの詳細を伝達する場合。
見た目だけではなく、様々な観点で観察をし、情報を得る必要がある。



ソフトウェアも同じで、内容を深く理解するには、対象物を見る目(観点)が大切と考える。

(観点を持てば、「仕様書の行間」も見えてくる)

様々なレビュー技法の活用や、プロセス改善を行ったとしても、レビュアーが「観点」を持たなければ、効果は上がらないと考える。

レビューポイントのまとめ

レビューで問題を見逃す主な原因

①着眼漏れ(見ようとしない、気付かない)

②思考の偏(かたよ)り(思い込み、思い違い)

※思い込み：指示内容の制約等に、気付かない
※思い違い：知らない または 間違った確信。

レビュアーは、対象物を見る目(観点)が大切

レビュアーの質が向上すると、
①確認が十分に行われ、活発なレビューが期待できる。
②各種レビュー手法やプロセスを併用することで、レビュー成果の相乗効果を期待できる。

では、具体的に、どうすれば良いのか？これらを踏まえた上で、レビューにおける改善課題を検討する。

アジェンダ

- レビューポイントの考察
- レビューにおける改善課題
- 優れたレビュアーのノウハウについて
  - ①怪しいポイントを押さえるコツとは？
  - ②的確に不具合を摘出できるコツは？
  - 仕様の矛盾・抜けを見破るコツは？
- レビュー向け発想法について
  - ①定型連想型
  - ②エラー推測型
  - ③自由発想型
- 「レビュー眼」の効果と、今後の展望

Aシステムを題材に、レビュー時の思考プロセスを分析

レビューにおける改善課題を検討するために、
下記、Aシステムの新機能(仕様概要)を題材に、
レビュー時の思考プロセスを分析する。

新機能の仕様概要について:

携帯電話から、画像データをWebサーバへアップする機能を備える。ただし、関連者のみ利用可能。

(詳細は、仕様概要を説明後に実施されるという前提で実施)

※思考プロセスは人それぞれ多様であり、あくまで一例として挙げます。

まずは、気になる点に着目する。

契約との齟齬

提案書との整合性

新機能の仕様概要について:

携帯電話から、画像データをWebサーバへアップする機能を備える。ただし、関連者のみ利用可能。

<レビュー時の思考プロセス>
1)関心を持つ(着目する)
2)ヒラメキ、勘が働く(勘)
3)仮説を立てる(経験)
4)指摘する(度胸)

気になるキーワードを捕らえる。

関連者って、誰?

新機能の仕様概要について:

携帯電話から、画像データをWebサーバへアップする機能を備える。ただし、関連者のみ利用可能。

<レビュー時の思考プロセス>
1)関心を持つ(着目する)
2)ヒラメキ、勘が働く(勘)
3)仮説を立てる(経験)
4)指摘する(度胸)

キーワードから、何か、ヒラメキが生じます。

関連者って、誰?

新機能の仕様概要について:

・"関連者"の定義は?
・認証の仕組みは大丈夫?
・関連者の登録は、どのように行う?
・関連者以外がアクセスすると、どうなる?

<レビュー時の思考プロセス>
1)関心を持つ(着目する)
2)ヒラメキ、勘が働く(勘)
3)仮説を立てる(経験)
4)指摘する(度胸)

この内容で正しいのか、仮説を立てた上で、検討する。

関連者って、誰?

新機能の仕様概要について

・"関連者"の定義は?
・認証の仕組みは大丈夫?
・関連者の登録は、どのように行う?
・関連者以外がアクセスすると、どうなる?

<レビュー時の思考プロセス>
1)関心を持つ(着目する)
2)ヒラメキ、勘が働く(勘)
3)仮説を立てる(経験)
4)指摘する(度胸)

仮説に基づいて質問・指摘を行う。

優れたレビュアーの指摘内容を分析

1) 優れたレビュアーは、
怪しいポイント(着眼点)を押さえている!
2) 優れたレビュアーは、
的確に不具合を摘出できる!
3) 優れたレビュアーは、
仕様の矛盾・抜けを見破れる!

優れたレビュアーの秘訣は、後述する。

4) 優れたレビュアーは、
不具合を予防するための、適切なアドバイスを行うことができる!

QA技術分野であり、紙面 および発表時間制約の都合上、別の機会に取り扱う。

課題①
一歩すすんだレビュアーになるために、優れたレビュアーのノウハウを一般化できないか?

エラーを作り込む箇所の分析

レビュー時の思考プロセス：

1)関心を持つ（着目する）

2)ヒラメキ、勘が働く（勘）

3)仮説を立てる（経験）

4)指摘する（度胸）

ここが、エラーの根本原因！！

着眼漏れ

思考の偏り

いわゆる「KKD」（勘・経験・度胸）

課題②

着眼漏れや、思い込み／思い違いを低減し、蓄積された勘や経験を活かすには？

※ここでは、パーソナルスキルに絡化して、検討する。

レビューにおける改善課題のまとめ

課題① 優れたレビュアーのノウハウを一般化

・怪しいポイントを押さえるコツとは？
・的確に不具合を摘出できるコツは？
そして、
仕様の矛盾，抜けを見破るコツは？

→ これらの要点を、次章で考察

課題② 着眼漏れや、思い込み／思い違いを低減し、蓄積された勘や経験を活かす

・対象物を見る目（観点）を体系化
・思考の偏りをなくす（陥りやすい盲点をなくす）
・蓄積された勘や経験を、スムーズに引き出す

→ レビュー時の思考をスムーズにする「発想法」を考察

レビュー時の思考手順を整理する

アジェンダ

・レビューポイントの考察

・レビューにおける改善課題

・優れたレビュアーのノウハウについて
①怪しいポイントを押さえるコツとは？
②的確に不具合を摘出できるコツは？
仕様の矛盾・抜けを見破るコツは？

・レビュー向け発想法について
①定型連想型
②エラー推測型
③自由発想型

・「レビュー眼」の効果と、今後の展望

①怪しいポイントを押さえるコツとは？

怪しいポイントの共通点

優れたレビュアーは、バグのありそうな箇所を狙っている。

そのときの意識は「バグが**あるかも知れない**」。

BUG

ここが大事！と考える。

トム・デマルコ と ティモシー・リスターの著書
「熊とワルツを　～リスクを愉しむプロジェクト管理～」では、
「…かも知れない」というフレーズが多数出てくる。

このポイントを、身近な例に置き換えて考えてみよう。

怪しいポイントの身近な例

車の運転で、見通しの悪い角を曲がる場合。

初心者ドライバー	ベテランのドライバー
大丈夫だろう！ GoGo！	ひょっとしたら、人がいるかも知れない。いったん、停止しよう。
リスクを予知しない、危険運転	リスクを予知した、安全運転

怪しいポイント発見のコツ(心構え)

・よく観察して「的確に情報をキャッチ」

・レビュー時は「かもしれない」推論

レビュー時の思考フレーズ

[着目点]について	[考察]	[推論]
もし、○○が □□を	××すれば △△だったら	×××になるかもしれない（または、どうなるの？）

レビュー時の思考フレーズを使うことで、
多くの事柄をひとつずつ分解して、考察を行うことが可能。

優れたレビュアーの秘訣(心得)

1	不良を見つけようとするから、見つかる。(第三者検証は効果あり)
2	実際にその仕様に基づいて次工程で作れるのか、想像する。
3	「How(どのように)」の議論よりも「Why(なぜ)」「What(なに)」が大事。
4	「こうあるべき」という姿を思い浮かべて対比する。(価値感を持つ)
5	後から仕様を付け加えたような、「ただし」で始まる箇所が怪しい。
6	「A」でなく、「B」でないなら「C」とは限らない。(論理的に考察)
7	「○○○のはず」という発言は怪しい！(曖昧さを排除)
8	「絶対」という言葉は「絶対」にない。(「絶対安全!」って、ホント?)
9	設計者が面倒臭がるようなところに、バグがある。(ある意味、執念!)

先入観を持たない

その他、作成日時が深夜や休日となっている成果物に不具合が偏在していることも、良く聞く話である。

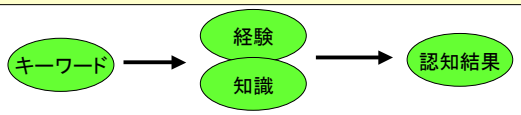
アジェンダ

- レビューポイントの考察
- レビューにおける改善課題
- 優れたレビュアーのノウハウについて
  - ①怪しいポイントを押さえるコツとは？
  - ②的確に不具合を摘出できるコツは？
    仕様の矛盾・抜けを見破るコツは？
- レビュー向け発想法について
  - ①定型連想型
  - ②エラー推測型
  - ③自由発想型
- 「レビュー眼」の効果と、今後の展望

②的確に不具合を摘出できるコツ、仕様の矛盾・抜けを見破るコツ

ベテランのレビュアーは、
無意識に【ヒラメキ、勘】で不具合を連想している。
いわゆる「アナロジー」が働いている状態。

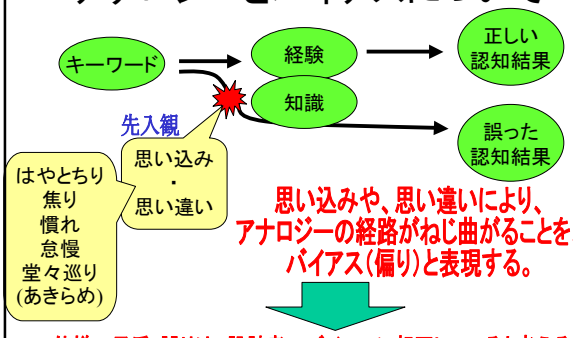
アナロジー:「類推する」という意味。



だれしも無意識に、キーワードから類推される知識・経験を連想している。

ヒラメキ・勘は、「アナロジー」により産み出されると考える。

アナロジーとバイアスについて

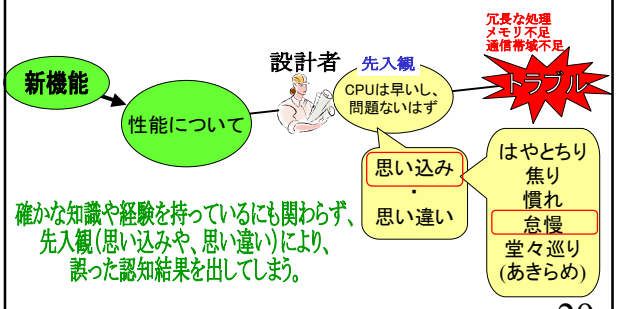


思い込みや、思い違いにより、
アナロジーの経路がねじ曲がることを、
バイアス(偏り)と表現する。

仕様の矛盾・誤りは、設計者のバイアスに起因していると考える。
どうすれば、バイアスを排除できるのか？

バイアスの一例

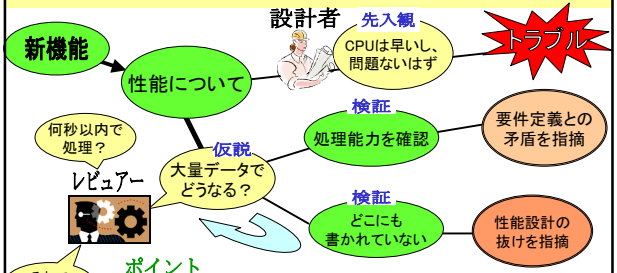
作成した当事者は、「大丈夫だろう」という、
先入観を持った物事の見方をしてしまいがち。



確かな知識や経験を持っているにも関わらず、
先入観(思い込みや、思い違い)により、
誤った認知結果を出してしまう。

レビュー時のバイアス排除例

先入観を除いて、「大丈夫なの？」と、
単純な問い(仮説)と確認(検証)を繰り返し、
設計者のバイアス(偏り)を排除している。



ポイント
①多角的に仮説を考察する。
②仮説と検証を繰り返す。
} いわゆる「洞察する」ということ

的確に不具合を摘出できるコツ、仕様の矛盾・抜けを見破るコツのまとめ

先入観を取り除いて、

・多角的に仮説を考察する。そして、

・仮説と検証を繰り返す。

では、どうすれば、的確な仮説や推論を出せるのか？

仮説を産み出す(発想する)手順を、「レビュー向け発想法」として考察

31

アジェンダ

・レビューポイントの考察

・レビューにおける改善課題

・優れたレビュアーのノウハウについて

①怪しいポイントを押さえるコツとは？
②的確に不具合を摘出できるコツは？
仕様の矛盾・抜けを見破るコツは？

・レビュー向け発想法について

①定型連想型
②エラー推測型
③自由発想型

・「レビュー眼」の効果と、今後の展望

32

レビュー向け発想法の形式について

レビューの目的は、不具合の発見と予知。
不具合を摘出する指向で考察を行う場合、下記3種類の発想形式があると考える。

考察フェーズ

定型連想型	正常、異常、境界、限界など、定型的な単語と組み合わせて考える。
エラー推測型	着目点から、想定されるエラーを推測し、該当するか否かを、確認する。
自由発想型	ある程度の方向性を与えつつ、自由な発想で考える。

着目点 → 推論

もし、○○が □□を → ××すれば △△だったら → ×××になるかもしれない（または、どうなるの？）

JaSST'09 Kansai (y.kadoguchi)

33

①定型連想型発想法について

考察フェーズにおいて、定型的なパターンを当てはめて、推論を導く。

着目点	考察	推論
もし、○○が □□を	××すれば △△だったら	×××になるかもしれない（または、どうなるの？）

■テスト設計の定石（セオリー）パターン

「正常」「異常」「境界」「限界」「状態遷移」「組合せ（マトリクス）」

テスト設計の定石パターンによる発想例

もし、**画像データ**が
- **異常値(画像以外)** だったら → **不正データを登録する** かもしれない
- **限界値(極大サイズ)** だったら → **性能が遅くなる** かもしれない

考察フェーズをパターン化することで、思考をスムーズ化する

JaSST'09 Kansai (y.kadoguchi)

34

その他の、定型連想型発想法について

発想法の例：SCAMPER

複数のキーワードを関連付けるなどして、発想する。

これらの頭文字をとって、「SCAMPER」。

- Substitute（代える、代用する）
- Combine（組み合わせる）
- Adapt（適応させる）
- Modify（修正する）
- Put to other uses（ほかの使いみち）
- Eliminate（省略する、除去する）
- Rearrange（再調整する）

Aシステムにおいて、①携帯電話を ②パソコンで ③代用すると…

参考文献：日経ITプロフェッショナル(日経BP社), 2004/05号

35

その他、使えそうな定型連想パターンを取り上げる。

定型連想名	連想パターン	概要
テンプレート	規格、規則類 類似システムなど、過去の成果物 観点リスト など	テンプレートに当てはめたり、何かと比較をすることで、新たな気付きを得たり、検討漏れを防止することが可能。
トランザクション	追加(作成)　競合 更新　排他 参照(検索)　中断 削除 復元	データ操作の定型的パターン。住所録やスケジュール帳など、データを扱うケースに当てはめることで検討漏れの防止につながる。
データ組合せ	AはBと同じ AにBを含む BにAを含む AとBの一部が重なっている AとBは一致しない AおよびBでないけれど、Cとは限らない	左記はデータ組合せの定型パターンであるが、組合せをマトリクス表で網羅することで、検討漏れ防止が可能。
ソフトウェア品質特性	機能性，信頼性，使用性，効率性，保守性，移植性	品質特性の観点で、検討漏れの防止が可能
非機能要件のパターン	機能性，信頼性，使用性，効率性，保守性，移植性，障害抑制性，効果性，運用性，技術要件	非機能要件の観点で、検討漏れの防止が可能
システムテストのカテゴリ	機能部分テスト，ボリューム，ストレス，有用度(ユーザビリティ)，セキュリティ，効率(パフォーマンス)，ストレージ，構成，互換性/構成/変換，設置，信頼性，回復，サービス性，文書，手続き	「ソフトウェア・テストの技法 第２版 (Glenford J. Myers著)」で、システムテスト実施内容について実践的な内容が述べられており、参考となる。

※詳細については、資料末尾の「付録4」を参照ください。

36

アジェンダ

- レビューポイントの考察
- レビューにおける改善課題
- 優れたレビュアーのノウハウについて
  - ①怪しいポイントを押さえるコツとは？
  - ②的確に不具合を摘出できるコツは？
  - 仕様の矛盾・抜けを見破るコツは？
- レビュー向け発想法について
  - ①定型連想型
  - ②エラー推測型
  - ③自由発想型
- 「レビュー眼」の効果と、今後の展望

37

②エラー推測型発想法について

着目点から、想定されるエラーを洗い出し、該当するか否かを確認する。

着目点	推論	考察
○○で □□が	×××の不具合が あるかも知れない。	▲▲▲は大丈夫？ △△△しても大丈夫？

(1)発生しやすそうな不具合を洗い出す　(2)不具合が発生する要因を考察する

エラー推測による発想例

- 新システムで
  - 脆弱性がある かも知れない
    - ・OSのセキュリティ脆弱性
    - ・プログラムの不具合
    - ・通信の傍受
  - 移行に失敗する かも知れない
    - ・移行作業の手順ミス
    - ・データ移行プログラムの不良
    - ・インタフェース不良

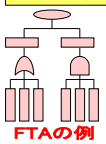
JaSST'09 Kansai (y.kadoguchi)

38

不具合を考察する時の問題点

FMEAまたはFTAで、発生する不具合(現象)から要因を検討すると、少し手間。
(図表や、リスク分析までは、必須ではない)
※FMEA(Failure Mode and Effect Analysis)
FTA (Fault Tree Analysis)

発生し得る不具合(現象)を予測するには、ある程度の知識・経験が必要。
不具合事例やナレッジデータベースは存在するけれど、探すのが手間。
一文字違うと、検索に引っかからない。



FTAの例

発想が妨げられる。

分析/抽象化

そこで、

過去の不具合を分析/抽象化し、レビュー観点として再利用する。

次に、Aシステムで発生した不具合事例を挙げてみよう。

39

Aシステムで発生した不具合事例

現象：画像データが流出してしまった。
発生条件/手順：クロスサイトスクリプティングによる被害
原因：実装漏れ(特殊文字のサニタイジング(無害化)漏れ)
要因：考慮不足
動機的要因：(1)仕様書に記載されておらず、検討漏れ
(2)フレームワークで実装済みと、思い込み
作りこみ工程：詳細設計
見逃し要因：テスト観点欠落(フレームワークで実装済みと誤解)

では、

どの情報をレビュー観点として再利用すれば良いのか？
(つまり、どのような観点が漏れたのか？)

40

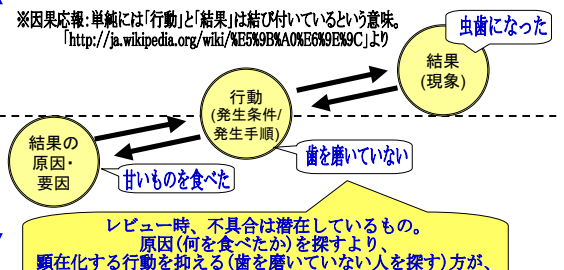
原因から結果までの経過を考察

身近なことわざに、「因果応報」がある。

※因果応報：単純には「行動」と「結果」は結び付いているという意味。
「http://ja.wikipedia.org/wiki/%E5%9B%A0%E6%9E%9C」より

顕在化した不具合

潜在する不具合



レビュー時、不具合は潜在しているもの。
原因(何を食べたか)を探すより、
顕在化する行動を抑える(歯を磨いていない人を探す)方が、
手間は少ない。

「発生条件/発生手順」がポイントと考える

JaSST'09 Kansai (y.kadoguchi)

41

不具合を、レビュー観点として再利用可能なように分析するには

不具合事例
- ・現象
- ・発生条件/手順
- ・原因
- ・要因
- ・動機的要
- ・作りこみ
- ・再発防止

発生条件/手順から、不具合を顕在化させるために行ったテストケースを分析する。

分析

レビューに再利用
【不具合の発見・予知】

どのテストケースで顕在化するのか?

テストケースを抽象化し、概念化

上位分類して整理

あるプロジェクトで、
不具合を顕在化させた手順を分析すると、
8~9割は、レビュー観点として
再利用することが可能であった。

※「発生条件／手順」をバグ管理システム上で明確にすることで、デバッグ時および修正確認の精度が向上する、相乗効果も期待できる。

42

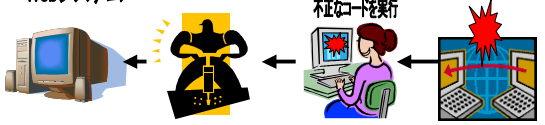
エラー推測のための、不具合分析例

不具合を、上位分類と結びつけて、蓄積。

④着目点　③上位分類　②顕在契機　①現象

Webシステム ← 脆弱性 ← クロスサイトスクリプティング ← 不具合

Webシステム　脆弱性　クロスサイトスクリプティングで不正なコードを実行　画像データ流出



43

上位分類からの、エラー推測例

蓄積した上位分類から、不具合が複数連想される。

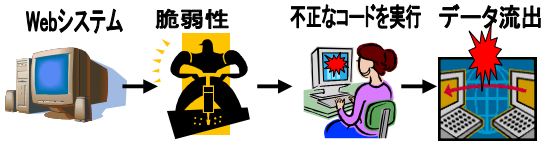
（アナロジーを活用するための道具として、利用可能）

①着目点　②上位分類　③顕在契機　④現象

Webシステム → 脆弱性 → クロスサイトスクリプティング → 不具合1

脆弱性 → インジェクション → 不具合2

Webシステム　脆弱性　不正なコードを実行　データ流出

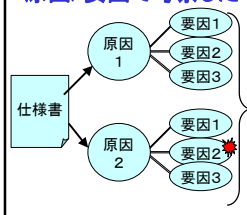


※筆者としては、発生した現象（不具合）と発生契機から、原因箇所の特定へ結びつけるような、トラブルシューティングにも応用することが可能と考える。

44

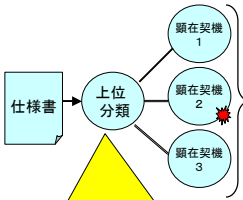
不具合の上位分類化による効果について

原因/要因で考察した場合



原因／要因数に比例して、確認数が増大

上位分類でエラー推測した場合



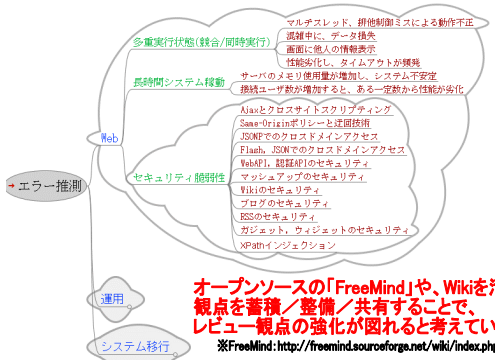
少ない観点で、多くの不具合にたどり着く

上位分類は、用途（レビュー目的）に合わせて、類推辞典（シソーラス）として整理するのが良いと考える。

■参考
※「基本から学ぶソフトウェアテスト（日経BP社）」の巻末付録に、400種類以上の「よくあるソフトウェア不具合」が記載されており、参考となる。

45

不具合事例をマインドマップで整理した例



オープンソースの「FreeMind」や、Wikiを活用し、観点を蓄積／整備／共有することで、レビュー観点の強化が図れると考えています。

※FreeMind：http://freemind.sourceforge.net/wiki/index.php/Main_Page

JaSST'09 Kansai (y.kadoguchi)

46

アジェンダ

- レビューポイントの考察
- レビューにおける改善課題
- 優れたレビュアーのノウハウについて
  - ①怪しいポイントを押さえるコツとは？
  - ②的確に不具合を摘出できるコツは？
  - 仕様の矛盾・抜けを見破るコツは？
- レビュー向け発想法について
  - ①定型連想型
  - ②エラー推測型
  - ③自由発想型
- 「レビュー眼」の効果と、今後の展望

47

③自由発想型発想法について

発想法の例：飛び石

着目点（キーワード）をもとに、発想を飛躍させる。

誇張：極端な条件や状況を設定する

願望：現実や実現可能性などを一切考えずに理想形を思い浮かべる

歪曲：組織や人、モノなどの関係をねじ曲げてみる

逆転：関係を逆転させる

着目点	考察	推論
もし、○○が□□を	××すれば△△だったら	どうなるの？

参考文献：日経ITプロフェッショナル（日経BP社）, 2004/05号

48

Aシステムにおける、「飛び石」での発想例

誇張:極端な条件や状況を設定する

→ システム利用者が10倍になったら…

→ 10年後も正常稼動させるには…

願望:現実や実現可能性などを一切考えずに理想形を思い浮かべる

→ 誰でも、ワンボタンで操作可能な、使いやすさ

歪曲:組織や人、モノなどの関係をねじ曲げてみる

→ 利用者が、他社のスパイだったら…

逆転:関係を逆転させる

→ アップロードができるなら、ダウンロードは？

※筆者としては、「願望」×「逆転」で、「反目標」を検討するなど、制約条件についても応用を利かせれば良いと考えている。

49

その他、使えそうな、自由発想例を取り上げる。

■情報の構造化

「分類」：類似点を分類整理する

「関連」：相反、相互作用、影響範囲

「配列」：時間や量で並べる

キーワードを、分類・関連付け・並べかえることで、何かが見えてくる。

■ペルソナ

仮想ユーザを定義し、それぞれの仮想ユーザの立場から、利用方法（用い方）や、利用される場面を検討する。（例：日立花子 25才、うっかりミスが多い、etc…）

■発想を変える4つの視点

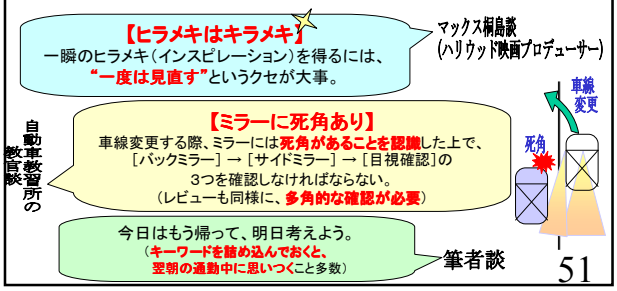
視点（立場）	いまの位置・立場そのままでなく、相手の立場、他人の視点、子供の視点、外国人の視点、過去からの視点、未来からの視点、上下前後左右、表裏等々
見かけ（外観）	見えている形・大きさ・構造のままに見ない、大きくしたり小さくしたり、分けたり合わせたり、伸ばしたり縮めたり、早くしたり遅くしたり、前後上下を変えたり等々
意味（価値）	分かっている常識・知識のままに見ない、別の意味、裏の意味、逆の価値、具体化したり抽象化したり、まとめたりわけたり、喩えたり等々
条件（状況）	「いま」「ここ」だけでのピンポイントでなく、5年後、10年後、100年後、1000年後あるいは5年前、10年前、100年前、1000年前等々

参考文献：日経ITプロフェッショナル（日経BP社）, 2004/05号

50

レビュー向け発想法のまとめと、心得など

定型連想型	正常、異常、境界、限界など、定型的な単語と組み合わせて考える。
エラー推測型	着目点から想定されるエラーを推測し、該当するか否かを、確認する。
自由発想型	ある程度の方向性を与えつつ、自由な発想で考える。



51

アジェンダ

- レビューポイントの考察
- レビューにおける改善課題
- 優れたレビュアーのノウハウについて
  - ①怪しいポイントを押さえるコツとは？
  - ②的確に不具合を摘出できるコツは？
    仕様の矛盾・抜けを見破るコツは？
- レビュー向け発想法について
  - ①定型連想型
  - ②エラー推測型
  - ③自由発想型
- 「レビュー眼」の効果と、今後の展望

52

◆レビュー眼による改善効果（1）

課題①:優れたレビュアーのノウハウを一般化

怪しいポイントを押さえるコツ

・よく観察して「的確に情報をキャッチ」

・レビュー時は「かもしれない」推論

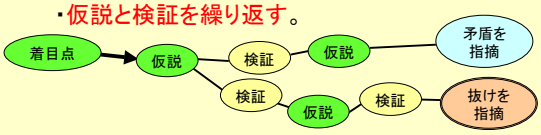
的確に不具合を摘出できるコツ、仕様の矛盾，抜けを見破るコツ

先入観を取り除いて、

・多角的に仮説を考察する。そして、

・仮説と検証を繰り返す。

いわゆる「洞察する」ということ



53

◆レビュー眼による改善効果（2）

課題②:着眼漏れや、思い込み／思い違いを低減し、蓄積された勘や経験を活かす

レビュー向け発想法（3種類の発想形式）で考察することにより、「着眼漏れ」「思考の偏り(思い込み／思い違い)」に起因するエラーを低減させつつ、蓄積された情報を有効に活用することが可能。

レビュー向け発想法に求めた要件	定型連想型	エラー推測型	自由発想型
対象物を見る目（視点）の体系化	○	○	○
思考の偏りをなくす（陥りやすい盲点をなくす）	○	○	○
蓄積された勘や経験を、スムーズに引き出す	△ 現在の知識以上のものは得られない	○	※ 既成概念に捕らわれない発想が可能
的確な仮説と推論を導き出す	○	○	△ 「反目標」を検討することで、的確性が増す

※レビュー観点にも死角があることを認識した上で発想形式を使い分け、多角的な検討を行うことが大切。

54

◆「レビュー眼」今後の展望

レビュー眼
（心得、観点、発想法）

まだまだ狭い範囲での検討結果であり、これからもブラッシュアップが必要。

TEFなどのコミュニティの場で、
より良いものを議論していきましょう！
技法をオープンソースのように、磨いていこう！

■参考■
TEF(Testing Engineer's Forum: ソフトウェアテスト技術者交流会)ホームページ
http://www.swtest.jp/wiki/index.php?swtest.jp/wiki/forum
※TEFへの参加は無料です

55

◆最後に

我々の発見力には、どんな最新ツールもかなわない！

・各種テスト技法や、高額なツールも、“観点”が無ければ実施されないし、不良も発見されない。何ごともレビュー眼のような“観点”を持つことが大切と考える。

JaSST'09 Kansaiサブテーマ
「きて、みて、つかんで、you get a chance！」

・どんなにコンピュータが発達しても、何かを創造できるのは、人間だけの取り柄。
新たなチャンスを作るのも、我々次第！
今一度、ものづくりの大切さ・素晴らしさを振返ってみよう。

レビュー眼で、チャンスを発見しましょう！

56

謝辞

・当プレゼンテーション資料を作成するあたり、JaSST実行委員ならびに、弊社の諸兄方に多大ならぬインスピレーションを頂きましたことを、この場をお借りして、御礼申し上げます。

・また、予稿集作成 ならびに シンポジウム開催準備，会場設営にあたられた方々にも、御礼申し上げます。

57

END

ご清聴、ありがとうございました。

（現場で活用する際に参考となる情報を、付録に記載していますので、合わせて参照ください）

58

付録1：レビュー技法について

レビュー技法の一例

技法	概要	主な目的
インスペクション	専門チームによるレビューを行う（または、レビューを受ける）	対象の合否判定とプロセス改善
ピアレビュー	「同僚によるチェック」の意味合い。 ※幅広い意味があるが、ここでは狭義に捉える。	不明点や、間違い/勘違いの発見
ウォークスルー	役割を決めて、実際の処理流れに沿って、抜けや検討漏れがないか確認する。	成果物のエラーの発見
パスアラウンド	「回覧・回し読み」などで、たくさんの人の意見を集める。（多重同時進行）	多数の意見を収集

参考文献:「ソフトウェア品質保証入門(日科技連出版社)」、「ピアレビュー(日経BP社)」

59

付録2：レビューツールについて

レビューツールの一例

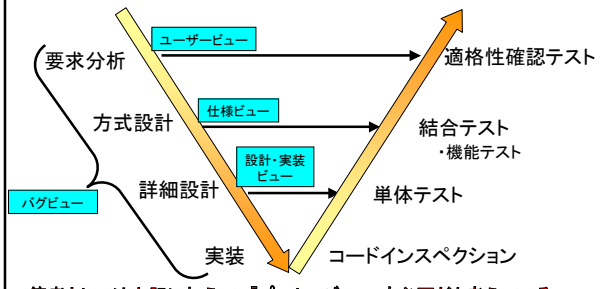
レビューツール	概要
ブレーンストーミング	テーマに基づく、複数人数での自由発想により、思いつきや気になる事項を挙げていく。
5W1H	Why,Whatなどの質問で詳細を具体化する
テンプレート／ワークシート	品質特性や、過去事例など、テンプレートになるものを活用する。
チェックリスト	想定する出来事や過去の経験を一覧化しておき、対比させる
マインドマップ	連想するキーワードを増やす、メモ技術
FMEA/FTA	不具合の可能性から仕様を検討する

60

付録3：レビューの視点について

Ostrandの4つのビュー

ユーザービュー、仕様ビュー、設計・実装ビュー、バグビュー



筆者としては上記に加えて、「プロセスビュー」も必要だと考えている。

付録4：その他、使えそうな定型連想パターンの詳細（1）

■テンプレート

規格、規則類
類似システムなど、過去の成果物
観点リスト など

テンプレートに当てはめたり、何かと比較をすることで、新たな気付きを得たり、検討漏れを防止することが可能。

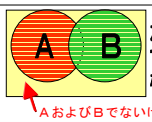
■トランザクション

追加（作成）　競合
更新　排他
参照（検索）　中断
削除
復元

データ操作の定型的パターン。住所録やスケジュール帳など、データを扱うケースに当てはめることで検討漏れの防止につながる。

■データ組合せ

- AはBと同じ
- AにBを含む
- BにAを含む
- AとBの一部が重なっている
- AとBは一致しない



左記はデータ組合せのパターンであるが、マトリクス表で組合せを網羅することで、検討漏れを防止することが可能。

Aおよびでないけれど、Cとは限らない

付録4：その他、使えそうな定型連想パターンの詳細（2）

■ソフトウェア品質特性

ISO/IEC9126(JIS X 0129) ソフトウェア品質特性

特性	品質副特性
機能性	合目的性, 正確性, 相互運用性, セキュリティ, 標準適合性
信頼性	成熟性, 障害許容性, 回復性, 標準適合性
使用性	理解性, 習得性, 運用性, 魅力性, 標準適合性
効率性	時間効率性, 資源効率性, 標準適合性
保守性	解析性, 変更性, 安定性, 試験性, 標準適合性
移植性	環境適応性, 設置性, 共存性, 置換性, 標準適合性

参考URL：http://ja.wikipedia.org/wiki/ISO_9126

■非機能要件のパターン

非機能要件（10個の主特性）

機能性，信頼性，使用性，効率性，保守性，移植性，障害抑制性，効果性，運用性，技術要件

詳細は、JUAS(日本情報システム・ユーザー協会)発行の「非機能要求仕様定義ガイドライン」で述べられている

■システムテストのカテゴリ

機能部分テスト，ボリューム，ストレス，有用度（ユーザビリティ），セキュリティ，効率（パフォーマンス），ストレージ，構成，互換性/構成/変換，設置，信頼性，回復，サービス性，文書，手続き

参考：Glenford J. Myersは、著書「ソフトウェア・テストの技法 第2版」で、システムテスト実施内容について詳細を述べている。

付録5：レビュー観点の、現場での活用例

レビュー観点なし	レビュー観点あり
レビュー担当者は、A機能は○○くん、B機能は△△くん。よろしくね。	○○くんは、この観点で、△△くんは、こちらの観点で、機能を横断的に見てね。
・知識,経験が少ないと、指摘できない問題あり。 ・仕様書に書いてあることしか、思い浮かばない。 ・機能ごとに、レビュー結果のバラツキあり。 ・機能間にまたがるような問題は気付きにくい。 ・観点漏れによる見逃しリスクあり	・知識,経験が少なくても、観点に基づく質問や仮説で、問題を摘出しやすくなる。 ・仕様書に書いていないことも、見破りやすい(洞察しやすい) ・機能ごとの、レビュー内容が一定する。 ・機能間にまたがるような問題も発見しやすい ・観点が明確。(見逃し漏れが少ない)

なお、以下のいずれかが不足しても、レビュー効果を十分に発揮できないと考える。

(1) レビュアーに与える情報　→入力される情報の精度を向上させること
(2) 経験（蓄積された知識）　→適切なレビュアーを割り当てること
(3) 多角的な観点　→複数人(2~4名)でレビューを実施すること

付録6：設計者が涙する言葉とは？

ちまたで語られる「設計者が必ず涙する言葉」を紹介する。
(TEFメーリングリスト[swtest 8138]より引用)

「ほか無いん？」

「それだけなん？」

「ほんま，それだけ？」

「それだけで ええの？」

涙する言葉を、嫁さんの声で録音するとか、上司の声で再生すると、効果倍増…という話もある。

※筆者としては、レビュー時は仮説に基づいた問いを行うと、より良いと考える。
（根拠の無い質問は、迷惑がられる）

付録7：

QCD、どれが大事？

Q：Quality（品質）　例）うまい
C：Cost（費用）　やすい
D：Delivery（納期）　はやい

Case	品質	費用	納期	事例	評価
1	△	○	○	費用の限界と、納期が近づいたので、ある程度の品質（最低限の動作が可能）状態で、完了させた。	必ずしも、お客様にとっての品質が良いとはいえない。納品後のクレーム対応の損失あり。
2	△ ↓ ○	○ ↓ △	○	納期直前で品質に問題があることが判明。短期間で品質を確保するために、人員をかなり投入した。	プロジェクトとして、必ずしも成功とはいえない。
3	× ↓ △	×	×	品質が悪く、デグレードが多発。後戻り作業が多く、修正／見直しに人員と期間を費やした。	最悪の事態
4	◎	○	○	品質が早期に確保され、後戻り作業や追加作業が無く、予定通りの費用と納期で完了した。	理想形

結論として、「品質第一」とすれば、費用・納期も成功へ導くことが可能。
逆説的に、品質を疎かにすると、費用・納期が危うくなる。
「品質」だけを、後から付け足すようなことはできない。

付録8：交通安全⇒プロジェクト安全

自動車教習所で必ず学ぶ、
安全運転のコツとは？

①的確に情報をキャッチ

②「かも知れない」運転

③ミラーに死角あり

④ 交通ルールを守る

ソフトウェア開発においても、
プロセス(作り方)を遵守することは、とても大事。

※レビュー眼と、各種技法やプロセスを
組み合わせることで、より高い相乗効果を期待できる。