第5期成果発表会

ソフトウェア品質保証の肝Part Ⅲ
～わかっちゃいるけどうまくいかない悩み解決します～

2014年11月25日

SQiP品質保証部長の会

第３部（第５期 第２グループ）

メンバー (五十音順)

池上　直之　AJS株式会社
稲冨　秀人　株式会社FAITEC
大石　晃裕　日立製作所株式会社
鎌倉　洋一　株式会社富士通ミッションクリティカルシステムズ
佐藤　孝司　日本電気株式会社
佐野　健士　日本アイ・ビー・エム株式会社
早崎　伸二　株式会社リンクレア
藤川　昌彦　アズビル株式会社

はじめに　～本活動の背景～

- 品質保証活動の肝となる考え方を創出し、整理し、分かり易く伝える資料にまとめたい
  - 3年間の活動の集大成を図る

- 特徴は品質保証業務経験者の生の声であること
  - 本に書かれた内容の受け売りではない

- 品質保証活動の悩みを解決するための肝を伝えることで、品質活動の初級者の育成に役立つ内容を目指す
- 品質保証部門に限らず、プロジェクトリーダーの立場から品質保証を考える人にも役立つ内容にする

「ソフトウェア品質保証の肝」 目次

第１章 「ソフトウェア品質保証の肝」とは

- “品質保証プロセス/仕組みはわかっているが、現場でうまく運用できない”
  - 品質保証部門長の“悩みが尽きない”!
  - “わかっちゃいるけど、うまくいかない”!

◆一方、これらの悩みを、豊富な経験から解決してきた事例も、たくさん持っている

⇒これらの悩みと解決ノウハウを収集し、“肝”として整理した

[解説①]

- “品質保証プロセスに必要な仕組みはわかっているが、現場ではうまく運用できていない”と、悩んでいる品質保証部門やプロジェクトリーダーは意外と多いかもしれません。一方で、これらの悩みは、ちょっとした工夫や考え方の修正で解決できることを経験してきた方も案外多いようです。このような経験を収集して“品質保証の肝”として整理しました。
  - たとえば、品質保証部門の思いが開発現場から乖離していませんか？
  - 開発現場をその気にさせるための本来の目的を確認してみましょう。
  - すると、仕組みを運用する上で、各所に勘所のようなものが存在するはずです。
  - これらの勘所は試行錯誤を積み重ねた経験として暗黙知になっています。
  - 従って、肝とは仕組みを円滑に運用する人の行動に着目した経験ともいえます。
  - また、肝が、品質保証プロセスの改善の機会の原動力になります。
- このようなノウハウは既に著名な方の書物などに言い尽くされているかもしれませんが、少なくともここに記載されたものは、現場を担当している我々の生の声であり、成功体験であり、議論を重ねた結果であることを自信を持ってご紹介したいと思います。

[解説②]

- 結局、“ソフトウェア品質保証の肝”とは何か？
- 品質保証活動における悩みがなぜ生じるのか？
  - 品質保証の仕組みに頼りすぎていないか？
  - 本来の目的を明確にしないまま、管理などの“手段”を押し付けたり、目的化してしまっていないか？

⇒本来の目的を実現するためには、どのような仕組みを使って、どのように運用するかが、悩みの生じるところであり、各々の組織に適切な解をみつけることが肝となる

[解説③]

- 世の中に紹介されている品質活動に関する理屈は正当である。
- しかし、現場に適用しても、うまくいかない。
- 理屈を表面的に現場に持ち込んでも運用できないということ。
- この問題を抽象的にとらえても現場では現実に矛盾が生じているわけなので議論が発散する。
- 例えば、早く判定しないと現場は先に進む、でも、早く判定しようにも実績データが揃わなければ判定できない。
- そこで、理屈ではなく、使い分けのノウハウが肝となると考える。
- 肝は、理屈ではなく、現場感覚で矛盾する事象もバランスよく運用する感覚が必要と考える。肝の意見が異なるということは、それぞれの意見の中に潜む肝で選り分けていくことが必要と考える。

第２章　ソフトウェア品質保証の肝

※３年間の活動で収集した８５件の肝を紹介

※SQuBOK体系で整理

参考

SQuBOK®樹形図 （Guide to the Software Quality Body Of Knowledge V2）

1. ソフトウェア品質の基本概念

1.1 品質の概念
- 1.1.1 品質の定義
- 1.1.2 ソフトウェア品質モデル
- 1.1.3 メンテナビリティ
- 1.1.4 リライアビリティ
- 1.1.5 セキュリティ
- 1.1.6 ユーザビリティ
- 1.1.7 ディペンダビリティ
- 1.1.8 セーフティ

1.2 品質マネジメントの概念
- 1.2.1 品質コントロール
- 1.2.2 品質保証の考え方
- 1.2.3 改善の考え方

1.3 SWの品質マネジメントの特徴
- 1.3.1 プロダクト品質とプロセス品質
- 1.3.2 品質作り込み技術の考え方
- 1.3.3 ソフトウェア測定の考え方
- 1.3.4 ソフトウェア評価の考え方
- 1.3.5 V&V(Verification & Validation)

2. ソフトウェア品質マネジメント

組織レベル
- 2.1 SW品質マネジメントシステムの構築と運用
- 2.2 ライフサイクルプロセスのマネジメント
- 2.3 プロセスアセスメント・プロセス改善
- 2.4 監査のマネジメント
- 2.5 教育・育成のマネジメント
- 2.6 法的権利・法的責任のマネジメント

プロジェクト共通レベル
- 2.7 意思決定のマネジメント
- 2.8 調達マネジメント
- 2.9 リスクマネジメント
- 2.10 品質管理
- 2.11 トレーサビリティ管理
- 2.12 要求管理
- 2.13 構成管理
- 2.14 情報・文書管理
- 2.15 プロジェクトマネジメント

プロジェクト個別レベル
- 2.16 品質計画のマネジメント
- 2.17 要求分析のマネジメント
- 2.18 アーキテクチャ設計のマネジメント
- 2.19 実装のマネジメント
- 2.20 レビューのマネジメント
- 2.21 テストのマネジメント
- 2.22 品質分析・評価のマネジメント
- 2.23 リリース可否判定
- 2.24 運用のマネジメント
- 2.25 保守のマネジメント

3. ソフトウェア品質技術

工程に共通なSW品質技術
- 3.1 メトリクス
- 3.2 モデル化の技法
- 3.3 形式手法

工程に個別なSW品質技術
- 3.4 品質計画の技法
- 3.5 要求分析の技法
- 3.6 設計の技法
- 3.7 実装の技法
- 3.8 レビューの技法
- 3.9 テストの技法
- 3.10 品質分析・評価の技法
- 3.11 保守の技法
- 3.12 運用の技法

専門的品質特性のSW品質技術
- 3.13 ユーザビリティの技法
- 3.14 セーフティ（安全性）の技法
- 3.15 セキュリティの技法

2章の分類に沿って整理

【表記】 赤字：新たな章立て、青字：内容改訂

SQuBOK 体系 2．ソフトウェア品質マネジメント　目次	肝の個数
2.1 ソフトウェア品質マネジメントシステムの構築と運用	4
2.2 ライフサイクルプロセスのマネジメント	2
2.3 ソフトウェアプロセス改善のマネジメント	2
2.4 監査のマネジメント	2
2.5 教育・育成のマネジメント	5
2.6 法的権利・法的責任のマネジメント	1
2.7 意思決定のマネジメント	1
2.8 調達マネジメント	2
2.9 リスクマネジメント	3
2.10 品質管理	3
2.11 トレーサビリティ管理	1
2.12 要求管理	2
2.13 構成管理	2
2.14 情報・文書管理	1
2.15 プロジェクトマネジメント	6
2.16 品質計画のマネジメント	7
2.17 要求分析のマネジメント	1
2.18 設計のマネジメント	5
2.19 実装のマネジメント	2
2.20 レビューのマネジメント	9
2.21 テストのマネジメント	3
2.22 品質分析・評価のマネジメント	9
2.23 リリース可否判定	2
2.24 運用のマネジメント	3
2.25 保守のマネジメント	1
【領域外】サービス品質	6
総計	85

【肝001】組織の品質目標の達成が品質保証活動の目的

- 組織目標が無くて、品質保証活動はありえない
- 品質目標合意→品質計画策定→品質施策実行（監視→対策実行→効果確認→監視）のＰＤＣＡサイクルを回す事
- 品質管理（データ収集）は目的でなく手段。目標達成の手段は様々
- 目標達成による成果（バグが減る等）を実感するまで開発部門は冷たい
- 成果が見えるまでブレなく粘り強く地道な努力が絶対に必要

【肝002】品質保証部門のミッションを明確にして「ブレない」こと

【背景】

- 品質保証部門は、ISO9001、トラブル対応、品質データ分析など、トップから矢継ぎ早に業務を命じられる。何でも屋なのか？どこを向いて仕事をすればよいのか、わからなくなる

【解決のヒント】

- 品質保証部門の業務は多岐に渡り、組織の中の位置づけも各々のケースで決める必要があるため、トップを含めた組織的な合意が必須
- ミッションは、「製品が品質要求事項を満たして必要十分な情報を提供できる状態であることを、事実として証明できる体系的な活動を継続すること」
- 基本的には組織の品質目標を策定し目標達成をミッションとすべき
- 業務の方向性は以下の２種類がある（障害対応系は別として）
  ①品質管理や出荷判定を主導し、“出荷品質”を確保すること
  ②品質の良い開発をするためのプロセス改善を主導し、“強い開発組織”を作ること

【肝003】品証活動は専任者を増やせなければ推進者を増やす

【背景】

- 品証の活動領域は広くて多岐に渡る
- しかし、新設の品証部門の場合、少数の担当しかいない
- 少数の品証担当で、品質活動をするためにはどうすればよいか

【解決のヒント】

- 経営層に品質活動の責任者を置く
- 経営層に働きかけて、各部署に品質担当（ＳＱＡ）を置く
- 品証部門は、各部署の品質担当に対する教育を行う
  - 品質意識を持ってもらう。品質確保の手順を教育する
- その人達を核にして品質活動を展開する
- 品証部門が直接対応するのは、重要問題が発生している部署に絞る

【肝004】品証部門と開発部門はアウトプットで議論する

【背景】

- 開発部門は品保部門のことを「あいつらは監視しているだけで仕事の邪魔ばかりする。」と思っている。
- 品保部門は開発部門のことを「あいつらはすぐに手を抜く。品質の悪いものばかり作っている」と思っている。
- 上記の例えは極端な例だが、Win-Winの関係には遠い。

【解決のヒント】

- プロジェクト開発においては、開発も、ＱＡも、ＰＭＯも役割は違うが目的は同じであることを共有する
- プロジェクトが成功すれば、開発もＱＡもＰＭＯも全てインセンティブが与えられる、だから、互いに協力しあう、という意識合わせが必要
- 品証部門と開発部門との間にアウトプットを置き、品証の指摘は、開発者に対してではなく、あくまでアウトプットに対して行う。アウトプットを良くしていきたい、という思いは、開発者と同じ。

【肝005】品証部門の成果は、出荷後に判断される

- 出荷までに実施してきた品証部門の活動の成果は、お客様先での問題発生の状況により判断される。
- 出荷後に発生した問題は、品証部門自身の活動を振り返る貴重な情報である。
- 出荷後に発生した問題を収集できる仕組みを作っておく必要がある。その際の注意点として以下がある。
  - お客様先では、発生した問題が、現場で片付けば、報告を上げない傾向にある。
  - 問題発生を、責めない・叱らない、という姿勢でないと、正確な情報収集ができない。

【肝006】現場の開発プロセス把握が品質分析の前提

【背景】

- プロジェクトの品質データを横並びで見たり、過去の実績と比較したりすると、データのブレ幅が大きいことがある。なぜだろう？

【肝の説明】

- プログラム設計書が妙に少ない
  ⇒ツールを使った開発が大半で、プログラム設計書を書かない
- ソースレビューの指摘件数が妙に少ない
  ⇒テスト駆動開発のため、ソースレビューは単体テスト後にやるリファクタリングが中心
- テスト件数がやたら多い
  ⇒テスト自動化されているので、過去から累積した項目を全て実施している

★あらかじめ、プロジェクトが採用している開発プロセスを知ったうえで品質分析をしないと、ミスリードしてしまう

【肝007】モデルは現場に合ったテーラリングをしてこそ使えるものに

- CMMI、PSPなどソフトウェア開発プロセスの改善モデルは、自分の組織に合ったテーラリングをしてこそ初めて使えるようになる。
- プロセス改善が定着し、効果を上げるまでに１０年程度の年月がかかるのは、どの会社も似たような「学習」を経ないと、正解を求め続けるという愚から脱することは出来ないからだと思う。
- 学習費用はだいたいにおいて、高くつく。専属の改善担当を配置できるだけの体力のある企業であり、企業のトップが「改善とは時間がかかるものだ」という認識を持ち、改善には１０年というスパンが必要であるという認識があり、かつ、「どこかに本当は王道があるのではないか？」という誘惑に打ち勝つことが出来た場合のみ改善は定着して、効果を発揮する。
- 改善推進者を置いたら、置いたことに対して責任を持とう。
- 改善推進者を置く場合には、次の言葉を考えてからにしよう。
  - 「改善をする”覚悟”が本当にあるか？」

【肝008】ＱＭＳは想い（設計思想）を伝えないと浸透しない

【解決のヒント】

- どんなに精緻でよく設計されたQMSが定義されても、飾ってあるだけでは絵に描いた餅である
- QMSの普及は勿論、QMS自身が本当に役立つようになることも、現場中心のPDCAがなされない限り達成されない
- QMS上で、目標を具体的・定量的に設定し、それに本当に接近しているかを、組織のレベルで組織長が本気になって追求することのみが、品質向上にQMSが役立つ道である
- 組織のトップである経営層が本気で取り組まない限り、組織長も、組織のメンバーも本気にならない

【肝009】監査を技術伝承の場にする

【背景】

- 「監査」は、監査員も、監査される側のイメージも、上から目線になりがちである。

【解決のヒント】

- 監査体制の提案
  - ①嫌われ役の監査員、②監査される側の助け舟を出す監査員で構成する
  - 嫌われ役の監査員は、あえて「嫌われ」役として振る舞う。監査される側の助け舟を出す監査員は、嫌われ役の指摘を緩和することでネガティブな場の雰囲気をポジティブに変える役割をする。
  - ただし、嫌われ役は本人の適性もあることも配慮する。
- 監査される側と監査する側が長い年月をかけて、仕事を続けていく同士であるから、お互いに緊張感を維持しつつ、良い関係を築いていければ良い、と思う。

【肝010】ISO9001は「気づきを得るツール」として使う

【肝の説明】

- 品質向上プロセスととらえ、プロセスに息を吹き込む
  - 形骸化しているプロセスは大胆に除外することを検討する。ＱＣＤに対して意味のないプロセスならば不要。
  - プロセスの弱点を考える良い機会と考える。
  - 品質目標未達成の原因、バグ分析、バグ傾向分析など、改善の機会を組み入れてみる。
  - 自ら積極的に探し出した改善活動こそＱＭＳ活動。
  - ＩＳＯ９００１の不適合を積極的に受け入れること。指摘を受けると作業が増えるので執拗に抵抗するのは本末転倒。

【肝011】ソフトウェア開発するために必要なスキルとは

- ソフトウェアを開発する人は、良い設計や良いプログラムができること。
- ただし、開発者であっても、プロジェクトマネージメントに関する知識（ＰＭＢＯＫ等）や品質管理等の知識もあったほうが良い。
- 開発者全員にＰＭＢＯＫに精通させるのは時間もお金もかかるので、開発対象に適切な組織構成を維持できることが望ましい。

【肝012】品質保証やPMO活動には高スキルが必要

- 考え方のひとつとして・・・
  - 必要なスキル
    - 論理思考、本質把握、全体を俯瞰できる、体系化/抽象化できる、問題の重要度を客観的に位置づけられる→おかしいな、変だな、とわかる
    - コミュニケーション力、挑戦する力、忍耐強さ、負けない根性・・・・
    - プロジェクトリスクの具現化とそれらのリスクアセスメントができること
    - 個々の問題を解決するだけでなく、再発防止を組織の問題として仕組み化できる
  - コアコンピタンス
    - 組織横断的にさまざまな成功事例、失敗事例、課題と解決策を経験している→ 傾向がわかる。過去事例を蓄積。統計分析。データで語れる。
    - 開発技術の改善例もたくさん見て知っている→ 組織内の横展開ができる唯一の部門

【肝013】品質改善推進者の人材確保は経営課題としてとらえよう

【背景】

- トップから「品質改善の成果を見せろ」と迫られるが、何を成果として見せていいのか分からない
- 改善施策が、どれだけ品質に影響を与えたかを定量的に示すことが難しいので、説明に自信を持てない
- 従って、積極的に人材を確保するためのアピールができない

【解決のヒント】

- 現場に理解者を作る（時間がかかる）
  - １０年もすれば、現場の理解者が、裁量権をもった人財に育つ。
  - その間、絶えず改善努力を続ける必要がある。組織の２０%の人の理解を得られれば、改善が停止するリスクは低くなる。
- 組織変更等で、改善が振り出しに戻らないように、組織横断的に理解者を得ておくことが必要
- 改善成果が経営数値に結びついているように見せることが必要。経営層は経営数値が変化しないと納得しない

【肝014】有識者は計画的に育成する

- プロジェクト内で有識者として育成するメンバを計画して動機付けしておく。
  - プロジェクト開始時に十分な有識者がアサインできているケースは少ない。
  - 誰もが有識者として成長する過程が必要である。
- レビューの場は育成の機会と捉える。
  - 勉強会そのものの効果はあるが、レビューの場が育成する絶好の場である。有識者は次の有識者を育てるという意識のもと、レビューの場で考えさせながら指導を行う。
- 組織の支援体制も重要である。
  - 有識者は、原理原則の理解と実践の場（ＯＪＴ）の繰り返しによって育っていく。教育、研修など機会を与える組織的支援が必要である。

【肝015】事業戦略として教育を定着させる

- 経営層に訴える
  - 多くの人が品質教育を受けていない実態を認識してもらう
- 会社制度とリンクさせて受講者を増やし継続もできる
  - 人事部門（昇級）、教育部門（キャリア認定）との連携を行う
- 全てを自部署だけでやろうとしない
  - 品質教育の専門会社を活用する
- 理論だけでなく、手も動かせる教育が望ましい
  - 座学だけでなく、チーム討議や実機演習も取り入れて即戦力を養う

【肝016】開発時のコンプライアンス遵守のために

【背景】

- 開発中に、権利侵害やＯＳＳライセンス違反が発覚すると、手戻りが大きい

【解決のヒント】

- 開発の計画段階で、チェックするルールを設定しておく
  - 品証部門の担当範囲でない場合は、担当部署と連携してルールを作る
- 受託開発の場合は、ＯＳＳの使用を認めないお客様もいるので、要件定義の時点で、お客様に確認しておく

【肝017】意思決定がタイムリーでないプロジェクトは注意する

- 工程会議で、決裁者に報告しないプロジェクトは怪しい。
- マススケジュールは、下記の傾向が見られる。
  1. そもそも、マイルストンが記載されていない。
  2. 自社のスケジュールだけで、お客様側のスケジュールが併記されていない。

⇒このようなスケジュールを見ても指摘をしない決裁者も問題

- 今決めるべきことか先送りしても問題ないかを判断して決める
- 直感でなく直観(＊)で判断する

  ＊:思考や推論などによらず、本質を直接にとらえること

【肝018】発注先の成熟度を観て管理レベルを変える

【背景】

- 国内／国外のアウトソーシングが増えているが、どんな管理をすべきか分からない

【解決のヒント】

- 発注前に押さえる（一括委託契約書等を交付）
  - 品質基準、PJ管理基準を合意する
  - 要所要所で確認ポイント、事前納品も計画する
  - 発注先のスキルを判定できる情報を得ることも重要

- 確認ポイントや進捗管理で課題を見つける
  - 仕様理解が怪しそうなところ
  - 何も質問がなかったところ
  - 工程遅れに対する対策の確認
  - 特に、発注先リーダがメンバの実態を把握できていなさそうなところに注目する

【肝019】オフショア開発は、成熟度や文化が国内と異なることを考慮する

- 発注元の意図が正確に伝わらないリスクあり
- 受け入れ時のレビュー・テストで指摘・バグが多く発生する

従って、

- 国内の開発に比べ、指標値を1.5～2倍に設定する。
- ２回目の発注であっても、慣れたからといって指標値を変えない（人員の流動により、習熟効果が期待できないため）
- 伝わらない部分は、自身の判断で、設計・実装する。

【肝020】リスクは集めることが目的ではない

- リスクは、計画し、監視し、対処することが重要
- リスクを発見した後の計画が重要である。
- リスクを全部取り上げようと思うな。一人の人間が１０時間考えて考え付かないことは、１００時間投入しても無理と悟れ。
- リスク抽出は非情にて行うべし。「ｘｘｘさんは頑張っているから～」とか、「ｘｘｘは有り得ないよねぇ～」とかの油断は禁物。
- 技術的なリスクで崩壊したプロジェクトを見たことがない。社会的なリスクで崩壊したプロジェクトは後を絶たない
- 発見した途端に「解決した」と思いたがる傾向にあるが、リスクは成長（問題化）する。

【肝021】有識者の“直感”は、下手な分析を凌駕する

- どんなにメトリクスを駆使しても、見えてこないリスクがある
- 人間の直感は時として、どんな正確な検出器をも凌駕する
- 人の話は真摯に聞こう
- 嗅覚の鋭い人は、沈没する船のかすかな軋みを敏感に察知する
- 重要なメンバが「逃げ」のモードに入る前に、リスクの根源を絶つ（手段は問わない）ことも必要

【肝022】対処不能なリスクも存在する

- 「腐った政治を下から変更することはできない」（デマルコ氏の名著「熊とワルツを」）
- 特に人的障害がリスクの場合、かつ、それが排除不能な地位に居る人物から発せられている場合、プロジェクトが悲惨な収束に陥らないように、“正しく失敗させる”ことも必要になる

【肝023】手順や帳票フォームは、その意味や目的を伝えないと形骸化する

【背景】

- 複数のチームから構成されるプロジェクトで、品質管理の手順や帳票類が不統一のため、同一観点での横並び評価ができない。
  （後になって、障害票の書き直しをする事態となる）

【解決のヒント】

- 作業開始前に、プロジェクトとしての統一手順、統一帳票を規定しておく。
- 分類コードの意味づけを、プロジェクト全体に周知して、同一観点での品質分析・評価ができるようにする。

（品証部門が、全社共通の手順やツールを提示して、教育することで、プロジェクト側の作業を軽減できる）

【肝024】障害票は品質評価のインプット

【背景】

●障害票をなぜキチンと書かなければならないのか、時間の無駄ではないか、という質問に、どう答えるか。

【解決のヒント】

1.障害票はアウトプットでなく、品質評価のインプットである

- 障害票をインプットとして品質評価ネタのアウトプットが生まれる
- いい加減な障害票の記入は品質評価の質を下げることになる
- 正確に障害票を書けない担当者は一度品質分析を経験してみるとよい

2.品質の基礎データを整理するのは、開発者個人の役割である

●品質管理の基本は自己管理

- 規模、テスト項目数、バグ数を整理することは誰でも出来る
- 難しいのは自己評価。障害票もまともに書けない担当者は問題外である

●品質管理は品質管理者だけが行うものではない

- プロジェクトリーダー、設計担当、製造担当、品質管理者など全員で行うもの

【肝025】コントロール→マネジメントの差を意識して向上を

- コントロール→管理→マネジメント（広義になっていく）である
- 「コントロール可能であること」（可制御）とは、
  - ある入力を与えたら、ある出力があることが分かっている場合、入力をコントロールして、出力を得る行為を「コントロール」すると言う。
  - 例えば、ドキュメント１００ページに、何時間のレビュー工数を使えば、どれだけの指摘が出てくるかが分かっているか？この場合、入力（レビュー工数）と出力（指摘）の関係が分かっている場合のみ、「コントロール（制御）できる」と言う
  - 現実問題として、何の予想もなく、やみくもにレビューに時間を費やすことが多い。
- 「管理できている」とは、
  - 前提条件としては「コントロール（制御）されている状態である」ことが必要。
  - 例えば、レビューを繰り返す時、毎回同じ出力が得られているかを観測する。
  - これには、QC７つ道具の「管理図」が利用できる。異常があれば、特異点として管理図上に現れる。可管理状態とは、このように「異常点」を捕捉できる状態を言う。
  - 現実問題として、何の管理もなく、ただ毎回無意味なレビューが垂れ流される現状がある
- 第三段階が、マネジメントである。
  - コントロールでき、管理もできている、その状態であることが前提条件である。
  - ここまでできることで、初めて大きな視野で品質をマネジメントできる。
  - 全体の最適化に着手できるのは、この第三段階の状況であろう。
- これらの関係性を正しく意識することが重要である。

【肝026】トレーサビリティはプロジェクトの成熟度を考慮する

【背景】

- 開発規模が大きくなると、トレーサビリティ負荷は爆発的に増える。
- 有限な資源で、どこまで実現できるかは、いつも悩ましい。

【解決のヒント】

- 極論すれば、以下のようなトレーサビリティでも良い。
- 要件→試験（妥当性確認）のみを管理して、途中は全部捨てる。
- ただし、以下の紐付けだけは確認する。
- 要件 →システムで実現することを、ステークホルダ間で合意した「システム要件」
- 妥当性→システムが要件に正しく応えている状態
- これだけでも確実に確認できれば、ある意味「大丈夫」ではないか？
- 無理してトレーサビリティを追求して、途中で力尽きてしまっては意味がない
- 頭と尻尾をつなげる。最初はこれだけでもよいのではないか。
- これが出来てから、途中の状況を紐付けていけばよい。

【肝027】すべての出発点である要求管理は大事

- 要求のインプットは要望であり、その元は「欲望」である。
- 欲望は際限がなく、「コントロール（制御）」するのは極めて難しい。
- 従って、欲望の流入量を制御する弁を管理するしかない。
- しかし、無理やり入り込もうとする欲望には、欲望の転換が有効である。
  - 「対象ずらし」は、欲望の矛先を別のプロジェクトに押し付けることである。
  - 「時間ずらし」は、「次回対応します」として、時間を稼ぐ手法である。そのうちに忘れてくれるか、違う欲望に変化する可能性にかける。
- ふざけたことを言っているように聞こえるかもしれないが、世界の一番最初に欲望があり、それを徐々にシステムに変換するという作業は、それほど難しいことであると思う。
- 要求を取り違えてシステムを作成したときの「落胆」が尋常でないことを思えば、多少の荒業を使ってでも、欲望を排除した方が良いと経験的に痛切に思う。
- 制御弁をいかにうまくコントロールするかに、定石は無い。

【肝028】品質は顧客要求との合致である

- 品質は顧客要求との合致なのだから、作り手視点で考えてはダメである。
- 当り前品質（基本要求）は明示されないことがあり、この要求に対する欠陥はクレームに繋がる。
  - 業務担当者に「あってはならないことは何？」のキーワードで聞き出せる。
- 機能毎に、「この機能での致命的な結果は何か？」をユーザと擦り合わせれば、基本要求が見えてくる

【肝029】構成管理ツールは成熟度に合わせて管理レベルを上げる

- 「ツールさえ導入すればうまく行く」と考えてツールを導入した場合は、金も手間も増える。自分たちの能力も知らずに構成管理を全面運用すると失敗する。
- 最初はプロジェクトのポイントを絞った構成管理をする。または、最初は最新版管理だけでも良いかもしれない。最新版管理すらできないのは論外である。
- 構成管理の運用レベルの段階は、以下である。

  ①最新版の管理：ソースやドキュメントの最新版だけは分かる。

  ②バージョン管理：ソースやドキュメントの改版の前後の関係が分かる。

  ③構成管理：過去の構成のシステムを、ソースやドキュメントのツリーを辿って、再構築できる。
- 段階的な適用には「手間がかかる」が、最終的には、「段階的な手順を踏んだほうがやはり一番近道であった」と、悟るものである。

【肝030】構成管理は、極力枝分かれさせないこと

【悩み】

・お客様毎にカスタマイズして提供していくうちに、資産管理がうまくできなくなり、修正提供の失敗が頻発するようになった。

【肝の説明】

１） 管理対象となる世代を極力増やさない指導が必要。

－お客様からの要求により、開発途中品の先行提供の依頼

－お客様毎の要求により、機能の追加・変更

要求を受け入れるたびに、保守すべき世代が増加してしまう。

①開発途中品の先行提供は必要最小限にする。

②機能の追加・変更を行ったら、次期版では、極力汎用機能として、枝分かれ部分を吸収する。

２） ファイル・フォルダによるソースコード管理はミスを誘発するので構成管理ツールの導入を指導する。

※ただし、複数案件の開発期間が重複する場合は、分岐させることも重要

【肝031】文書管理はプロジェクトの身の丈に合わせる

【背景】

- 情報・文書管理の手順書はあるが、守られていない。

【肝の説明】

- 大規模開発のプロジェクトと少人数のプロジェクトとでは、必要な情報・文書管理の仕組みは異なる。
- 人間が覚えられるルールの数はそれほど多くない。
  - 大規模プロジェクトはツールの導入を検討
  - 小人数プロジェクトは人手でできる範囲のルール
- 管理すべき情報や文書の種類は、必要最小限に絞り込む。

【肝032】計画書が更新されないのは、計画書を使っていない証である

【背景】

- 品質保証部門が全てのプロジェクトの計画変更の状況を収集してトップ報告しているが、プロジェクトは計画書を変更しない、提出もしない
- プロジェクトの言い分は、計画書を更新しなくてもプロジェクト内は計画変更の伝達がうまくいっているので、面倒な改版はしない

【解決のヒント】

- プロジェクトの計画変更管理を明確にする。計画書を更新するのか、別の変更承認の手順を認めるのかルール化すること。
- 品質保証部門が計画書を集めてトップ報告している目的を明確にする
  - 例えば、トップはトップ判断が必要な計画変更を把握したいと考えているのであれば、トップへの稟議プロセスを明確にする。投資の総額や増額の上限値、納期遅延の限度等の管理項目を決めて、現状報告や変更時の承認プロセスを決める
- 計画書を活用しながら開発を進めるプロセスにすることが肝要（計画書の目的は額にいれて飾っておくことではない）
  - 工程会議では、まず、計画書を参照して、次に実績を見て、差異を確認する。差異があれば、吟味を行い、必要であれば計画書を変更する等のプロセス。
- 計画書は、最初に全ての項目を詳細に記載する必要はない。工程が進むにつれて追記や詳細化すればよい。

【肝033】他のプロジェクトの開発計画のコピペは怪しい

- この場合、プロマネがプロジェクトの要点やリスクを理解していない可能性がある。プロマネに下記項目を確認する。
  - 計画の各項目の根拠を聞く
  - 全体スケジュールと各工程での実施内容を確認
  - スケジュールに対しての体制と役割

（注）他のプロジェクトと共通の項目（管理方法、作業手順や会議体等）は、違和感がなければ問題なし

【肝034】非機能要件は数値で合意する

- 非機能（性能・可用性、セキュリティ等）は要件定義の時点では、あいまいになりがち。
- ＩＰＡのガイドラインを参考にチェックリストを作成し、要件定義の時点で、明確に定義され合意されているかを確認する。（後工程で、お客様とベンダーの間での齟齬が発覚すると、コストに大きく影響する）
- 非機能要件のグレード分けは意識せず、お客様や自分たちがわかりやすい言葉で目標値を表現する。
- 段階的にしか非機能要件を確定できない場合は、必ず要件数を管理して、いつまでにどれだけの要件を確定しなければならないかを決めておく。（スパイラル開発を利用すると良い）

【肝035】進捗の予定が数値化されて初めて実績と対比できる

【背景】

- 担当者は進捗を%で報告するが、数値の根拠がなく、適当な値と判明

【解決のヒント】

①進捗を示す%の算出方法を定義する
  - 設計工程なら仕様書の作成ページ数/予定全体ページ数
  - コーディング工程なら作成規模/予定全体規模
  - テスト工程なら実施テスト項目数/予定全体テスト項目数
  - １週間以下に作業を細分化したWBSの終わった数/全体数

  ※消費した工数で進捗を評価せず、どれだけの成果物（アウトプット）を生産したかで進捗を評価するべき。EVMは有効。ただし、過信は禁物。

②%の報告だけでなく、中間成果物を必ず出させ、内容をチェックする
  - 全ての内容をチェックすることは無理だが、サンプリングしてでも見る
  - 内容の正しさを見るわけではないので、チェック者は必ずしも技術スキルが高い必要はない

③過去に同様の問題があったチームや人の報告には、集中的に内容をチェックする

【肝036】まずい進捗報告は具体的に指摘する

- 進捗線の書き方
  - 進捗のイナズマ線がない→進捗管理が不在
  - 進捗のイナズマ線が、月１回だけ→報告のためだけの報告
  - 進捗のイナズマ線が、毎週まっすぐ→毎週予定どおりはありえない
  - 進捗のイナズマ線の遅れ項目が　何週間か同じ位置→対策されていない
- あいまいな表現
  - 予定□□□□□□
  - 実績■■■　のような表記
- 予定が、頻繁に書き直される→遅れのつけを先送りしているだけ
- 遅れ、進みを平均化して、予定どおりとしている→遅れ項目を注目すべし
- 工程が１ケ月単位で、ブレークダウンされていない

【肝037】品質報告は根拠の妥当性を見る

１：「問題なし」というトーンの報告の場合

- 「～のため問題なし」という説明に終始する品質報告は信用できない
  - 開発過程で品質問題の発生しないプロジェクトなどありえない
  - 現実を冷静に監視して、問題原因を分析し、対策を実施することが重要
  - 対策に至る経緯、対策の結果を説明したうえで「問題なし」という結論になっているか

２：「言い訳」に終始するトーンの報告の場合

- 「予定よりも少ないが簡単なため問題なし」という結果論の報告は根拠が希薄
  - 「簡単なため」というのは計画時から分かっていること。言いわけにすぎない
  - 計画時の想定はどうだったのか？実績は想定どおりか？想定外か？
  - 計画時の品質ベースラインを正として客観的かつ冷静に実績分析しているか
  - その結果、計画（予想）が間違っていれば言いわけでなく計画変更しているか

【肝038】品質目標はプロジェクトリーダーが良くしたいことを設定する

【背景】

- 品質目標がどんなものかわからない。
- 品質目標が無くても開発できるので、品質目標を立てようとしない。
- 達成度判断が可能な品質目標を設定できない。
- 品質目標が定量的な判定が難しい行動目標になってしまう。
  たとえば、レビューを十分に行う、など

【解決のヒント】

- 最初は「行動目標」であっても良いと割り切る。
  - 次に「行動した結果」として、変化したものを「計測」するようにする。
  - 次に「望ましい」変化を予測することで、その予測値を「目標」にする。
- プロジェクトが成功したかどうかの判断基準を目標にする（ＱＣＤなど）
  - プロジェクトが成功したら関係者全員にインセンティブを与える

【肝039】品質目標を決めないと問題が見えてこない

- 目標値は、当該プロジェクトや類似プロジェクトの過去の実績値をもとに設定するとよい。
  - 実績値が無い場合には、当該部門や会社の実績値を参考にする
  - それも無い場合には、世間一般（IPAのサイト等）から引用した値を参考にする
- 目標値は、開発部門が自ら決めることが望ましい。自力で決められない場合も多いが、上記を参考に手助けすれば大抵は決められる。
- 問題とは、目標と現状とのＧＡＰである

【肝040】プロジェクト目標と全社目標の乖離を評価する

- プロジェクト目標に連鎖しやすい全体目標を設定し、全体目標とプロジェクト目標の連鎖を確認ポイントとする。
- 全社目標、プロジェクト目標共に「目標」なので、達成度が評価できる指標でなければならない。PDCAを回すためにも、達成成否・ギャップを分析する。
- 全社目標はスローガン、プロジェクト目標は必達の値と考える
  - 全社目標：前年比３０%向上や重大障害ゼロなど、スローガンとして掲げる。
  - プロジェクトの目標：各プロジェクト毎の実績に合わせ全社目標を達成するために、必要な値を設定する。
- プロジェクトの目標値は、組織目標値より高い目標を設定しなければ組織目標は達成しにくい

【肝041】品質目標はリアルタイムに評価する

- 目標対象のスケジュールが完了した時点で初めて結果を評価していては遅い。
- 達成のためには目標を細分化して、実行すべき対象組織に割り振る。そして、着実に目標を達成するために、リアルタイムに評価し、目標から乖離が見られたらすぐに対策を打つというＰＤＣＡサイクルを回す事。
- 開発時の工程毎に目標に対する実績を評価することで、早いフィードバックを行う。

【肝042】見積もりの評価はベテランやデルファイ法を使う

- 見積もりが適切かを判断するのは非常に難しい
  - 見積もりの経験豊富なベテランに審査してもらう
  - 複数人で見積もられた合意（デルファイ法）を使う
- 見積りは、金額を見積もる前に工数を見積もる。その工数に　単価をかけて金額とする
- 見積りは、工期も見積りが必要である。工数と工期には関係がある。（COCOMOⅡ、JUASによる工期モデルを参考にする。ただし、プロジェクトが大きい場合には合わない）
- 新規開発と改造では、見積もりの基準値は異なる

【肝043】過去のプロジェクトの失敗を見積もりに反映する

- ＱＣＤで失敗したプロジェクトの原因を分析して見積もり時に反映するとよい
  - システムの特性や、お客様の体質
  - プロマネの資質や、メンバーのスキル

  これらからリスクを想定して、見積もりに反映する
- 計画策定時の見積もり値と、開発終了時の実績との差を分析して、見積りの精度を向上させる
- 工程終了毎に、見積もりと実績の差を確認しておくと、見積もりの弱点が見えてくる

【肝044】品質管理は数値の管理が要、要員数把握は品質計画の基本

1. 計画の策定と展開が遅れるほど痛手が大きく、しかも、かなり後になって問題が顕在化する。大規模プロジェクトでは致命傷となる。計画、説明、実行、監視を徹底し、評価して改善するＰＤＣＡサイクルを常に意識する。

2. 品質悪化プロジェクトは往々にして要員数さえ管理されていない。要員数・品質データは常に計画時から管理しておく。
（要員数、工数、機能数、画面・帳票数、Ｉ／Ｆ数、ﾌﾟﾛｸﾞﾗﾑ／ｓｈｅｌｌ本数・規模、不良数、運用手順書数など）

【肝045】仕様決定者を早く見抜く

- 複数のステークホルダーの中で誰が仕様を決定するキーマンなのかを、要件定義工程の早期段階で見極める。
- 業務有識者かどうかだけでなく、意見を通すことのできる発言力があるかなども見極めの要素である。
- 仕様調整の進め方・段取りを決める。曖昧になりがちなやり取りの内容を見える化し、確認する。
- 顧客の業務仕様を理解する、学ぶ姿勢が大事である。

【肝046】お客様の声を機能へフィードバック

- 使用性に関するお客さまのクレームはスペック（要求、仕様）の曖昧さや未定義など合意できていなかった場合が多い。
- お客さまの声を蓄積して他プロジェクトでも積極的にスペックに流用してお客さまに提案できるようにするとよい。
  - ただし、蓄積したデータを活用しやすくするために、蓄積したデータは具体的な内容であり、他でも流用しやすい汎用性も必要である。

【肝047】基本設計書にお客様の承認印を

- お客様との段階的な合意を得るため、設計書ごとにお客様から承認をいただくことが必要
- 承認をいただくのは難しいが、お客様との設計レビュー会議の議事録を送付しておくだけでもよい

【肝048】設計では、障害時の対策や運用を漏らさないこと

- 設計書のテンプレートにあらかじめ記載必要な項目を設定しておき、必ずテンプレートを使わせる。
- 設計審査のチェックリストに運用設計を考慮した項目を記載し、設計審査時に使用してもらう。
- 品質特性等を利用して、チェックすると効果的。チェックリストを作らず、仕様／設計のテンプレート自体に品質特性を盛り込んだ項目を追加しておくと、書き忘れを防止する効果がある。
  （障害耐性は機能性や信頼性、運用は使用性で設計する）

【肝049】“前機種と同じ”設計は要注意

- 「前機種と同じコードを移植／流用する」場合は危険
- 前機種との些細な違いが、重大な影響を与えることがある
  - 前機種と環境や運用が異なる部分が必ずあるはず。その部分を徹底的にレビューやテストで確認するべき。
- コードクローン検出ツールなど、どれだけ流用部分があるかをチェックするだけでも、影響範囲を特定することが可能

【肝050】設計書はどこまで書くべきかを考える

【背景】

- HW設計図は緻密である、ＳＷの設計書はこれに近づけるか
- ＳＷ開発は人である。設計は“そこそこ”で、“行間”を読む、とか、職人技が大事
- 昔はアセンブラ、メモリ削減、コードパンチ、マシンタイムなど制約が多く、緻密さが必要。今はPCでできる、ソース流用、あふれる情報（インターネット）・手段（OSS)から選択するなど、開発者は、緻密さよりも要領の良さが必要
- 正常系の記述は網羅されているが、異常系の記述が漏れるケースが多い。

【解決のヒント】

- 設計書は次工程へ引き継ぐための情報だと考える。
  - 次工程が作業できるだけの、漏れなく正確に記載されている必要がある。
- 設計書が何を為すべきものかを考える。そうすれば、仕様書に何が書かれるべきで、要求はどこで確定されるべきかがわかる。
- 識別ＩＤを付加したり、図や表で表現したりという工夫が必要。
- 「何がアウトプットされるか」が重要である。「どう作るか」は、標準化として、検討すべき。

【肝051】ツールに使われるな

- ソースコード検証ツールが莫大な量の警告を検出しても、担当者も管理者もやる気が失せてしまう
- ツールの警告の内容を理解し、どの警告に対応すべきかを、あらかじめ合意しておく
- ツールの実行時期は、対象となる設計書やコードが完成した時点では遅く、要所要所で適用すべき

【肝052】コード変更の影響範囲を見誤らないコツ

【背景】

- コードの変更による影響範囲を見積もることができない。
- 変更するたびに、すべてのコードの回帰試験が必要になってしまう。

【解決のヒント】

- 回帰試験項目を事前に決め、変更後は、必ず実施する
- 機能毎にテスト項目のブロック化を行なっておく。最初のうちは、全てのブロックを実施するが、何回か実施して、修正と影響のあるブロックの関係が見えてきたら、修正に対応した実施すべきブロックが見えてくる。そうすれば、テストすべきブロックの強弱がつけられるようになる。（ある場合は影響のありそうなブロックのみ実施。ある場合は全ブロックを実施）

【肝053】機能面以外の設計レビューのポイント

- 設計者本人に説明させる。あいまいな箇所や自信がなければ、しゃべらせるだけで問題箇所がわかる。本人が自分自身で気づく場合もある
- 設計の入力情報も確認する。（旧版や、間違った情報の参照を防ぐ）
- 第三者が理解できない文章は最初から不適合とする
- 書かれていないことを聞く（例：性能，容量）
- 必要なことが書かれていないことが判明したら、レビューを進めてはいけない。追記することを指摘し、レビューで書かれていない内容を深追いしない

【肝054】レビューで問題指摘が無いプロジェクトは怪しい

- 以下のレビュー情報を含むレビュー記録を提示してもらい、内容を確認する。
  - レビュー対象のドキュメント名/量、レビュー時間、参加者名、事前査読時間（参加者毎）
  - 指摘内容、指摘の分類、指摘者名、レビュー回数
- 再レビュー・強化レビューの記録も提示してもらう。この場合は、摘出目標値を設定して行うと、必ず新たな問題が見つかる。

【肝055】レビュー状況の評価は早めに行う

- レビューが全て完了してから、レビュー記録を一括して受領し評価していたが、問題を指摘しても、次工程で改善すると逃げられてしまう場合が多い。
- レビューの途中でも、レビューの終わった部分から提示してもらうようにして、レビュー票の記載不備を指摘しながら評価するようにした。早めの問題指摘ができるようになり、開発側はレビュー票の書き直しの手戻りが減った。

【肝056】レビュー会議を効果的に運営するコツ

【背景】

- 「多くの人にレビューしてもらいたい」という思いから、レビュー対象に関係の無い人が、レビュー会議に招集される。
- 「レビューは教育の一環だ」という認識から経験の浅い人が召集される。（教育効果があることは否定しないが、時と場合による）
- レビュー工数の水増しに利用される可能性がある。

【解決のヒント】

- 事前レビューの目的、対象を明確にする。
- レビュータイプを予め宣言する（例えば、“今日はチームレビューを行い、ウォークスルーではない”など）
- オブザーバーは議長（モデレータ）からの指名制にする。

【肝057】レビュー会議を効率良く運営するコツ

【背景】

- レビュー対象物が事前配布されていないケースもあり、レビューアがレビュー会議の場ではじめて対象物に目を通す場合もある。
- レビュー対象物が事前配布されていたにも関わらず、レビューアが事前査読していないため、会議のかなりの時間が、対象物への質問や確認に浪費されてしまう。

【解決のヒント】

- 事前査読できていない場合、レビュー会議の開始前に査読時間（短時間）を設ける。（査読時間はレビュー会議時間に含めない）
- 事前査読できていない人はレビューアにカウントしない、レビュー会議で発言しても記録しない。

【肝058】レビュー会議で検討会を始めない

【背景】

- 指摘するたびに、指摘に対する検討を始めてしまう。
- 検討会のために、レビューの時間が浪費されてしまい、検討会の時間もレビュー時間として計上されてしまう。

【解決のヒント】

- 検討会は別途分科会を結成して行なってもらうと宣言して、検討を中断させる。（モデレータの力量に依存する）

【肝059】レビュー結果の評価方法

【背景】

- レビューの評価に何を使っていいか分からない。
  - レビュー指摘密度の分母は規模か工数か？分子は指摘件数か？
  - レビュー指摘とは何か？質問は含むか？合意事項は含むか？アクションの発生しない指摘を含むか？
  - レビュー指摘密度でレビューを評価するレベルにあるかの判断基準は何か

【解決のヒント】

- 複数の指標を使い総合的に判断する。（１つの指標では判断できない）
  - 密度（指摘数/規模（KLOCや要件数））
  - 作業充当率（レビュー工数/該当プロセス工数）
  - 作業実施率（レビュー工数/KLOC）
- 指標だけでは判断できない。定性的な観点でレビュー指摘の内容が枯れてきたかを判断する。
- レビューで指摘すべき内容であるかを確認する。（誤字・脱字の指摘ではなく、本来指摘すべき内容になっているか）

【肝060】レビュー会議では目的を唱和するだけでもずいぶん違う

- レビューの目的は、「レビュー対象から、欠陥を除去すること」。
  - 教育的な価値や、メンバ間の情報共有もあるが、二次的な目的である。
  - レビューアがレビューアを「演じきれるか？」が重要であり、レビューの目的と自身の役割を忘れてはならない。
  - 多くのレビューは、ウォークスルーであり、「作成者」VS「レビューア」である。モデレータ不在の場合、レビューが設計検討会になるケースも多い。
- レビュー会の開会宣言に「本日のレビュー会の目的は、欠陥除去であり、設計検討は別途行いたいと考えています」と宣言するだけでも、随分と違うレビュー会になる。
  - 宣言の有効期間は、レビューの間だけ、一時間程度が限度である。
  - 別のレビュー会や、時間が延長された場合は、宣言の呪文は効力を失う。
  - よって、レビュー会が開催される度に、開会宣言を行う必要がある。

【肝061】レビューアには心得が必要

- レビューは、「活発な議論がしやすい雰囲気」を築くことにより、「リスクを検知しやすい環境」の形成に繋がる。
- レビュアーは単に質問を投げかけるだけでなく、「隠されたリスク」を顕在化させる問い掛けやファシリテーションの工夫を図ることが求められる。（以下の心得）

心得の七柱	具体的な内容
「サービス業」の意識	● 「何をしたら担当者の役に立つのか・プロジェクトが健全に進むのか．」それぞれの立場でそのプロジェクトのために**何ができるかを考えて，臨むこと**が重要である．
「レビュー」≠「業務監査」	● **準備と生産性とのバランスを重視し，プロジェクトレビューを最小の手間で実施する**ことも心掛ける．作り過ぎた資料と確認行為よりも，どのようなレビューであったか？という中身を重視するように進めていく．
「指摘」＜「質問」	● 担当者から状況・事実を把握するために，**「こうしなさい」の前に先ずは「どうなっていますか？」**と質問する．
「問題」＜「課題」	● 「問題」という言葉には，自分たちの手に負えなくなったというネガティブかつ主観的な物差しが入る．「課題」という言葉には，解決するために行動を起こすことを意志表明したポジティブな印象がある． 【レビュアーからの質問例】 ○：**「現時点の課題について教えてください」** ×：「何か問題はありますか？ないですか？」
「完璧ではない」という自覚	● 経験や感覚に頼るシーンが多いレビューだが，決して指摘の網羅性や深度が完璧であると自信過剰にならないこと．必ず，**チェックリストを用いて「経験以外の視点」からもレビューを行う**．
「根拠」と「合意」	● 「みなし」「決め付け」「思い込み」の落とし穴を見つけるため，根拠（**「どうしてこうなったのか？」** **「その事実関係は取れているのか？」**）や，合意（**「関係者の確認は済んでいるかか？」**）に着眼する．
これから先への「How」	● 資料の確認だけではなく，スキルや性格といった人的要素にも目を向けながら案件の進め方についてアドバイスする必要がある．**「これからどのように進めていくのか？」**を聞き，計画へのアドバイスを示唆する．

【肝062】出荷検査で品質の実態を現場に示す

- レビュー記録票や故障票の分析で、品質の課題を指摘しても、状況証拠なので、品質の良否についての説得力に欠ける場合があるため、出荷検査で、バグや仕様書の誤りを見つけてこそ、開発者に対する影響力が大きい。
  - 上流工程のレビュー指摘に対して、開発部門は「レビューでの指摘は障害ではない」という認識が強いので、上流での指摘に対して危機感が薄い。よって、出荷検査の段階で「要求」や「仕様」の不適合を開発部門に示すことで、どれほど手戻りが多いかを認識してもらうことが有効。レビューの意識が薄くても、試験で検出された不適合に対しては、非常に関心を示す。

【肝063】テストの見積もりのコツ

【背景】
- どれくらいのテスト項目数が必要かわからない。
- どれくらいのテスト工数が必要かわからない。
- 従って、テスト工期の判断ができない。

【解決のヒント】
- 過去プロジェクトやIPAの資料を参考に、規模あたりのテスト項目数の基準値を設定する。
- 工数も同様にテスト生産性から算出する。
- テスト工期もテスト生産性と規模と人数をもとに算出する。
- テストの１項目の考え方を定義しなければ比較できない。せめて画面系・操作系と制御系を分けるとよい
- 結合テスト以降は、プログラムやシステムの特性によりテスト項目の粒度が異なるため、一律な基準を設定することが難しい。従って、対象物の過去の開発時のデータをもとに見積もる。

【肝064】テストのフェーズの区切り方

【背景】

- テスト工程の区別（単体、結合、総合など）がつけられない。
- 区別がつかないので、渾然一体となったテストを実施する。
- どのテストフェーズで検出すべき不適合かを分析しないので、テストフェーズを後戻りする必要があるかどうか判断できない。
- テストフェーズの指針はあるが、抽象的すぎて分からない。

【解決のヒント】

- Wモデルを利用する。（設計書を書いたタイミングで、テスト仕様書を書けば自ずと、テストのフェーズは区切れる）
- テストの目的（開発のアウトプットの何を確認するのか）を把握しておけば単体・結合・総合といったテストの区切りにこだわらなくてもいい。

【肝065】工程の移行判定では、未完了項目の管理方法を明確にする

- 工程完了項目と未完了項目を明確に把握できれば、工程を条件付きで移行させても良いと考える。
- 未決定な要件や、未確定な仕様がある場合は、スパイラル開発を採用するなどして、進める。
- 未完了項目を定期的にフォローする会議を設ける
- ただし、工程移行不可となる必須条件を決めておく
- 未完了（課題）管理のコツ
  - 解決期限を明確にする。期限を何回も見直させない、期限が過ぎたままの状態にしない
  - 担当者を割り振る。
  - 課題をブレークダウンする。未完了なものは、より詳細化することで、進むところと進みにくいところを分解する
  - 残課題の数を管理する。設計前半時点では残課題が無いのはおかしい、テスト後半で残課題が増加傾向になるのはおかしい、と考える

【肝066】計測できないものは品質向上できない

- 品質が良くなったかは、計測して定量化しないとわからない。
- 計測するには、開発過程でデータを採取しなければならない。
- データ採取するには、何を採取するか決っていなければならない。
- 採取したデータをどう評価するかの尺度が決っていなければならない。
- 評価したら、その結果をどう生かすかアクションが決っていなければならない。
- 「GQM」を使って、目的と手段がリンクしていなければならない。これを怠ると、手段が目的化してしまい、陳腐化し、計測行為のみが惰性化するか、計測が消滅する。

【肝067】納得されるデータ分析のコツ

- 他組織とデータ比較をしても納得性が薄い。組織や開発の性格の違いは測れない。
- 同じ組織や似た開発案件の過去との比較で乖離がどれほどあるかを把握する。比較してこそ、どれだけ違うかの量がわかる。
- データを平均値で見ない→個別の異常な“外れ値”を探す
- データ上、何の問題もなし→逆に怪しい。成果物から実態を探る
- データの推移が、数値が上がったり下がったり、その場しのぎ　→怪しい。管理された状態ではない
- データの使い方を段階的に進化させる。

  ①：　測った結果を「理解」できること。（理解＝分析）
  ②：　理解した結果を使って、現状の説明ができること。（説明＝対策）
  ③：　対策を繰り返していける仕組みを作れること。（仕組み＝管理）
  ④：　管理した状態を標準化できること。（標準＝定着化）
  ⑤：　標準を改善できること。（改善＝CMMIのレベル５）

【肝068】品質指標の設定のコツ

- 品質指標はあまり欲張らない。
  - 上流（要求、要件、仕様、設計）：　レビュー指標
  - 中流（制作、単体試験）：　レビュー＋試験指標
  - 下流（結合、総合、評価試験）：　試験＋バグ指標
  - 全体（進捗（WBS）、リスク）：　WBS消費率、リスク発現率
    この程度で、まずは十分。
- 簡単なメトリクスから始める。全員が納得できる目的を決め、その目的の達成状況を説明するためだけのメトリクスを１、２個程度用意する。
- 複雑なメトリクスは、理解してもらうための障害になる。
- １０年計画くらいの長期計画で、５ステップくらいに分けて段階的にステップアップする。

【肝069】基準値は閾値、アクション無ければ意味なし

- メトリクスは、データを収集するだけではなく、実績値が基準値を超えた場合に、どのようなアクションを取るかまで、ルールを決めておくこと。基準値は閾値と考え、閾値を超えた場合には何かのアクション（分析と対策の実施と報告）をとらなければ意味がない

- 基準値という言葉が、現場の抵抗がある場合には、「参考値」と置き換えると良い。
参考なので、強制力は無いが、現場が自らアクションを取る方向に誘導する。（品質保証部の腕の見せ所）
自分で決めたアクションならば、現場も達成しようと頑張る。

【肝070】バグ分析で品質を見極める

- 開発終盤に発生したバグを分析し、なぜ、この時期に検出されたのか、これまでの弱点が何かを見極める。
  - 発生条件を見れば、もっと早い段階で見つけられるべきバグかどうかを判断できる。
  - 特に一つの工程を飛び越えて検出されたバグは、飛び越えてきた工程の運用自体に欠陥を抱えている可能性が高いので要注意。（例： コーディングで仕様のバグが見つかるなど）
- バグ分析ができないとは、弱点を見つけられない/論理的な説明ができないということ。品質の疑いが強くなる
  - バグ分析の悪い例は、“確認不足”、“漏れていた”、“思い込み”、“経験不足”。なぜ確認が不足したのか、なぜ漏れたのか、なぜ思い込みしたか、を追求しないと、見直す対象が絞れず、全てを見直すことになる
- バグ分析の専門チームを準備する。
- バグ分析しやすいように　ツールで解析、自動化を図る

【肝071】タイムリーな工程完了判定のコツ

【背景】

- 工程の作業終了後に、完了判定会議のためにリーダーが品質分析を行っている間、担当者は次工程の作業を開始してしまう。そのため、次工程がなし崩しに開始され、次工程作業と並行した是正策しかできないため、抜本的な対策が実施できない。

【解決のヒント】

- 工程完了判定に向けた品質分析は、工程の終盤から着手し、担当には必要な是正策を指示することで、なし崩しの次工程着手を防ぐ。

【肝072】データに溺れない

- 品質分析は、必要な量のデータに、必要なだけ分析し、適切なタイミングで、適切な人が評価する。分析「しすぎ」は禁物。
  - 品質分析には、「必要十分か？」を常に意識する必要がある。
  - 品質分析の対象データのほとんどは代替指標である。品質を直接表すことは少ないため、何らかの変換と、推定が必要になる。
- 品質分析の改善は、まずは、「現状に＋１（プラス、ワン）だけで良い」。
- 一度に複数の解を求めないこと。分析や施策がぶれる可能性がある。
  - レビュー記録が無ければ、レビューの質だけでも問いましょう。
  - 試験が十分でないと思ったら、試験の十分性だけを問いましょう。
- 分析手法に凝り始めるのは、３年くらいの実績を積んでからにしましょう。まずは、後戻りのない分析を定着させることが重要。

【肝073】データ収集の前に考えておくこと

- データ収集の前にデータの使い方を考えておかないと、無駄なデータ収集になる。
- データ収集してからを分析方法を決めると、足りないデータがあった場合、手遅れとなる。
- データ分析報告書は、読み手の視点で「目的と判断事項」を想定しておかないと、無駄なデータを生み出してしまう。

【肝074】テストの終了判断のコツ

【背景】

- いつテストを終了すべきか分からない。
- テストが十分かどうかも分からない。
- 不安なので、期限ぎりぎりまでテストをする。
- 期限になったら、「やるだけのことはやった」とテストを終了する。

【解決のヒント】

- 基本は、V字開発プロセスにより各設計仕様書を入力情報としたテスト項目を抽出し、全てのテスト項目を消化すること
- テスト進捗とバグ摘出状況のグラフから成果物の安定度合いを判断する
  - ただし、信頼度成長曲線は、総合試験などの運用を想定した試験で使って初めて意味がある。機能の確認の試験では、意味がない。
- テストカバレッジを測る

【肝075】リリース判定では、リリース後の準備状況も確認する

- リリース判定では、以下のリリース後の準備状況も確認する
  - ①本番移行の準備状況、②業務運用の準備状況、③保守体制の取決め状況、④本番トラブル発生時の取決め状況
- 「③保守体制の取決め状況」は以下を確認する
  - 本番稼働直後のフォロー体制や役割りを利用部門と合意しているか？
  - リリース直後のフォロー体制から通常保守体制への切り替え日時は明確か？
  - 保守体制／役割り／内容は明確か？
  - 保守費用についての具体的な取り決めは完了しているか？
- 「④本番トラブル発生時の取決め状況」は以下を確認する
  - 事故発生時の報告ルート、責任体制は明確か？
  - システム障害対応、誤操作対応、復旧対応の手順や実施体制は明確か？
  - 対応について瑕疵の切り分け方法、切り分け費用は合意しているか？

【肝076】システムを活かすも殺すも運用次第

- 運用を無理に強制しても放置しても、いけない
- システムの運用で重要なことは以下である
  - ①無理をさせない、②勝手にさせない、③①と②を時と場合で使い分ける
  - 特に③が重要。①と②は、運用管理者の性格によっては、極端な方向に振れることがある。従って、③のバランス感覚が重要になる。
  - ③をバランスよく配分するには、天賦の才能と冷静な分析が必要

- 従って、①コマンダーな資質を持つ人間と、②分析者の資質を持つ人間をペアで用意し、このマッチングをうまく取り、適材適所に当てはめるのが、重要なポイントである

【肝077】運用手順の遵守の“幅”や“深さ”は担当者の意識で変わる

- 運用メンバーが「自発的な行動」をするためには、その行為が「職場の常識」として共感されていなければならない。
- 従って、リーダーシップ，コミュニケーション，モチベーションなどを工夫して組織風土（＝職場の常識）を醸成する必要がある。このような取組みを「仕事」と認識しているか否かで、風土の良し悪しは決まる。
- 組織のパフォーマンスは、その水面下に隠れている組織風土の浮力に支えられている。プロセス，ツール，手順などが実装されるだけでなく、それらを動かす「人」の集団となる組織（プロジェクトやチーム）のパフォーマンスを良くしていこうとする取り組みがプロジェクト運営上の重要なポイントとなる。

【肝078】運用リーダーの発言力が重要

- あるタスクをメンバーに指示する際、いきなり「計画」から語り始めてないだろうか？ 何か新たな仕事を遂行する時には、そこに「目的」（ビジョン）があるはずであり、またリーダーとして「こうありたい」と望む「理念」があるべきである。業務指示や進捗管理だけではない「何か」をメンバーに発信していく行為も、リーダーの仕事となる。

＜頭の中の「見せる化」＞

- リーダーは自身の「判断結果」だけでなく「判断基準」もメンバーと共有する。メンバーも、リーダーの「内側」を知りたがっている。

＜計画をストーリー仕立てに＞

- リーダーは時に「理論」だけではない「ストーリー」を語ることも重要。ヒトが心を動かされるのは、頭の中でイメージがはっきりした瞬間。

＜作業≠仕事＞

- 理念やビジョンを問わない（「なぜ、これをやるのか？」を問わない）行為は「作業」となり、問い掛けて行う行為は「仕事」となる。

【肝079】トラブル情報は記述情報を知識化した記録にする

- トラブル情報は、“記述”情報を知識化した“記録”情報にしないと使えない
  - 記述情報：事象－経緯－推定原因－対処－総括
  - 記録情報：事象－【背景】－経緯－【原因】－【対策】－総括－【対策後状況】－【知識】
- トラブルの重大さを「物作りの視点」で判断してはダメ、上長に即連絡する
  - 「報告したくないなぁ～」と思った事象は報告事項！ 1秒でも早く報告する
  - 本番対応は時間軸を変える！製造業の製造ラインの単位である秒で考える
- 本番操作を１人でやってはいけない、複眼が基本である
  - 充分に注意をしていても、うっかり、ボンヤリ、勘違い、間違い、思い込みからは逃げられない！
- 手順書やチェックリストは「平常時の品質を保つツール」であり、想定外の事態では無力であることを肝に銘じる
  - 手順書やチェックリストは繰返し行うことの漏れや間違いを正すものであり、これから外れた場合は「非常事態に陥った」という検知が重要！

【肝080】お客様と納得のいくＳＬＡを設定するコツ

- SLAはお客様との“契約”として合意し納得いただくべきであるが、欧米と異なり日本のIT業界の習慣では、“緻密な契約”や“契約を無視してお客さまが追加要求してくる”ことがある。（日本のお客様は品質に対して厳しい）
- それでも、サービス提供側として以下の工夫するとよい。
  - お客様はSLAを正確に理解できていない場合が多い。お客様が本当に欲しているものを引き出してSLAにするのがよい。お客様の対応者（情シス部門、経営層等）によっても意見は異なることを理解しておく。
  - そのために、“どういう状態が困るのか”、“最悪の状態は何か”を具体的に引き出せるとよい。
  - SLAで設定した値（稼働率99.99%など）の意味をお客様に理解いただくこと。“本当に必要なのか”、“コストとの関連も理解されているか”、等

【肝081】お客様の緊急変更依頼を低減するコツ

- お客様の緊急変更依頼のフォーマットを決め、①背景、②目的、③理由/根拠等を必ず記入いただく。
  - お客様は深く考えずにリクエストしてくる場合もあるため、変更用のフォーマットを決めることで、要求内容が明確になり、緊急性や重要性を冷静にご判断いただけるようになる。
- 緊急の定義、優先度付けのルールを決め、変更依頼の一覧を見える化し、処理可能容量と処理状況をお客様と共有する。また、エスカレーションルールを決めることで、お客様の経営層にも認知いただく。
- 変更依頼に対して、緊急性や重要性をリスク判断することにより、変更しない期間を設ける。（ディレードオペレーション）

【肝082】組織の風土がサービス品質に与える影響を測定する

- 測定手法は、発生確率と影響度を各3段階評価指標に区分けした９象限マトリクス表を作成してリスク管理を行う。
- ヒューマンエラーを防ぐには風土は大事。チームが一丸となって自律してお互いのメンバを補完し合える関係に醸成していくことが必要。

【肝０８３】“出来て当たり前”を期待されるサービス運用者の真のモチベーション向上の方法

- 定期的な会合等で、お客様からサービス状況に対してお言葉をいただき、お褒めの言葉も含めて、現場の運用者と共有する。
- 必要以上に分厚くきめ細かな手順書は、逆に運用者にとって心理的な抑圧になりエラーを誘発する場合もある。
  - 手順をできる限りシンプルにする
  - レアケースや複雑な手順や専門家の判断を強いるような場合を全て手順に網羅するより、手順から外れたケースが発生したら全てエスカレーションする、くらいの手順のほうが間違いが発生しにくい場合もある。

【肝084】プロセスを重要視し、作業品質を確保する

- 日常の作業で使う帳票やツールの中にルールや手順を組み込み、ルールや手順を別の参照資料にしない
- ルールや手順は守れる程度をよく考えること（過剰に盛り込み過ぎない）
- チェックリストは、運用オペレーションの実行結果のように愚直に確認すべきものと、設計レビュー時の留意点のように開発者本人の気付きを促すものを使い分け、後者は多くなり過ぎないようコントロールする
- 第三者による監査（作業監査）の結果を大事にする

【肝085】成果品質だけでなくプロセス品質も大事

- 成果品質（ＱＣＤ）のみ注力してプロセス品質を疎かにすると、顧客満足でなく自己満足に陥る。
  - プロセス品質＝正確性・迅速性・柔軟性・共感性・安心感・好印象
- 「顧客の期待」に合致したサービスを提供しないと折角の行為が「余計なお世話」や「迷惑行為」に勘違いされる
- 実績評価が「顧客の期待」を上回れば「リピート客」となるが、イコールでは、競合が良ければ取引きが途絶えてしまう

第３章　肝が生まれるメカニズムについて

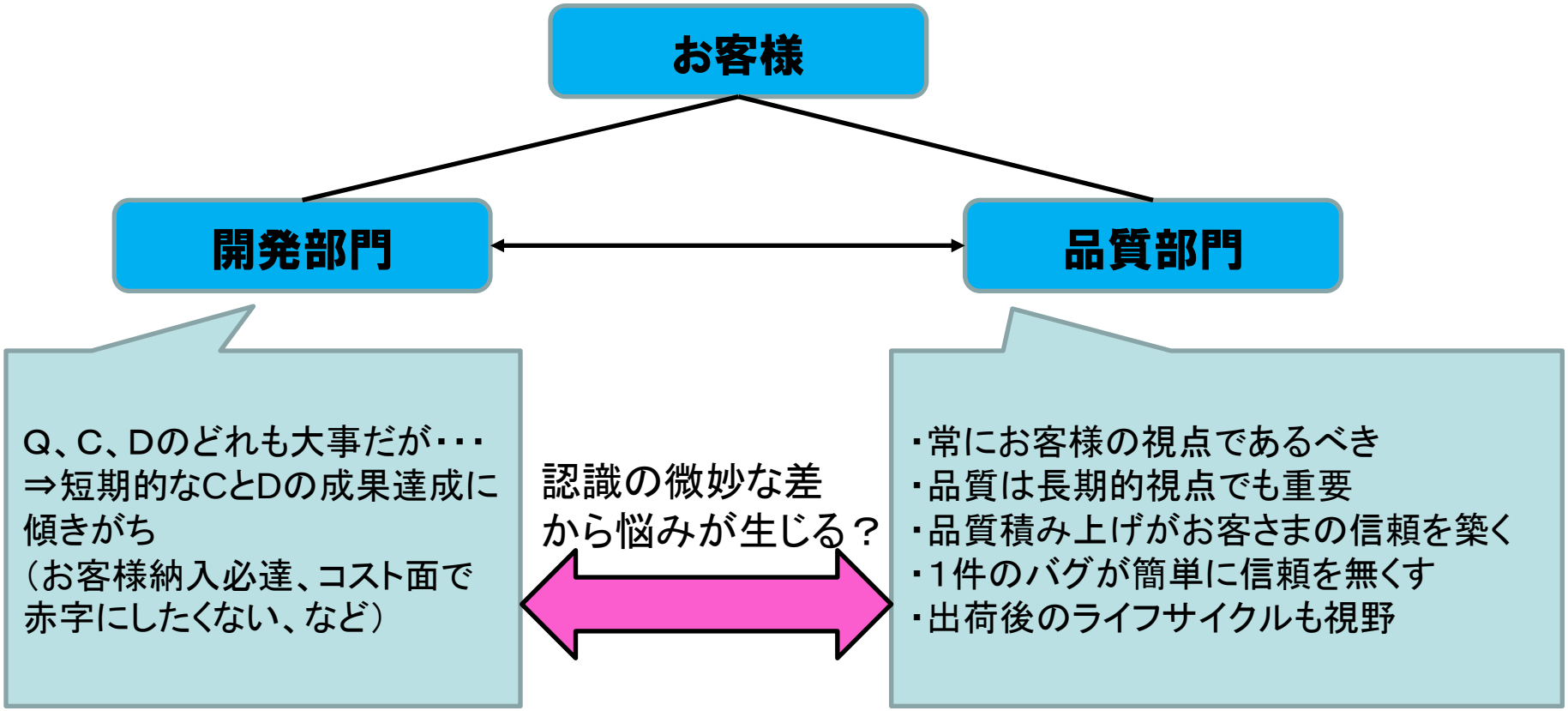
3.1 品質保証の視点・役割と、開発の視点との差から肝が生じる

3.2 PDCAサイクルの中で肝がステップアップの役割を果たす

3.3 開発組織の成熟と品質保証活動との差から肝が生じる

3.1 品質保証の視点・役割と、開発の視点との差から肝が生じる

お客さまに対する視点が、開発部門と品質部門で、ずれることがある



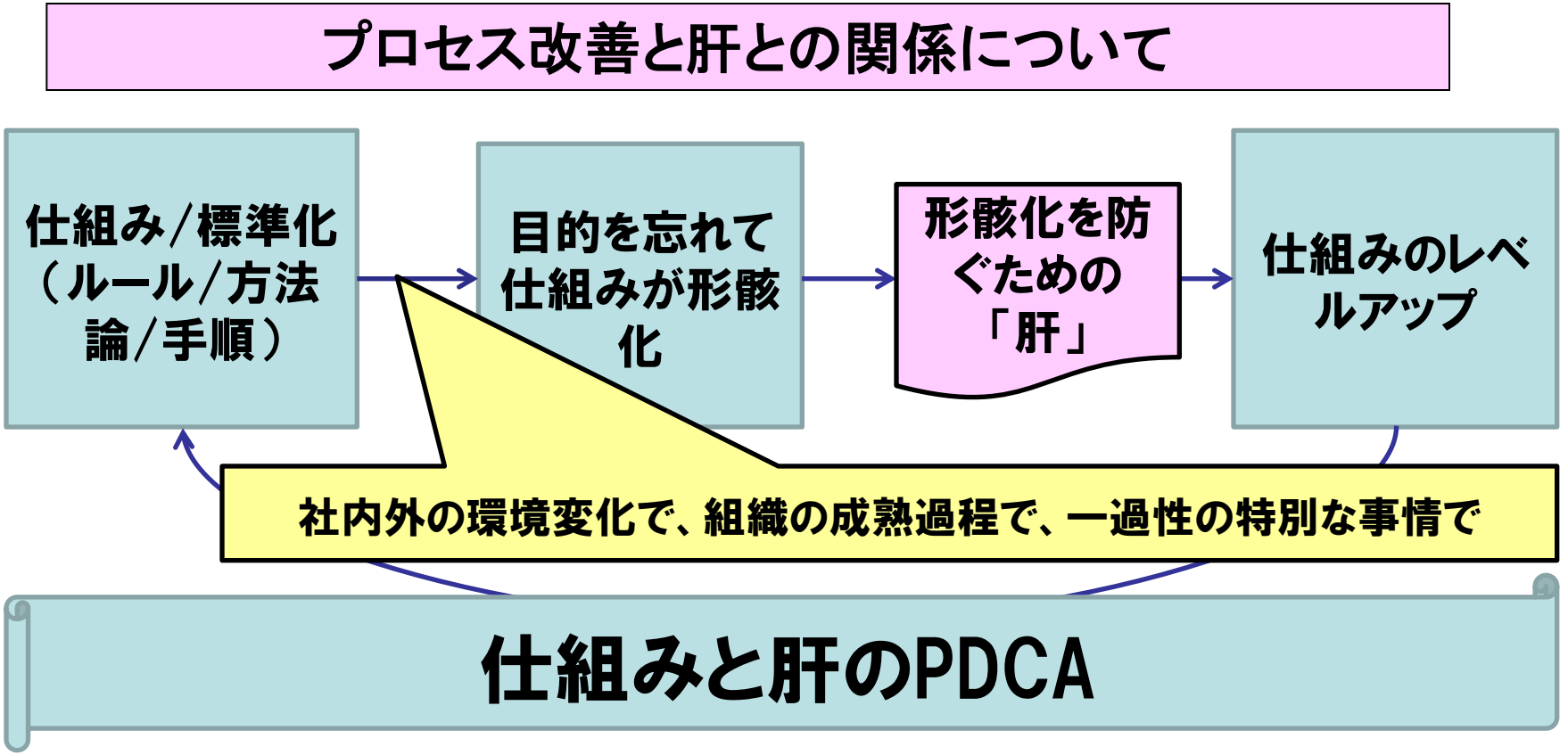
①開発部門と品質部門のゴールの共有、双方の立場の相互理解が重要

[解説]

「肝」が、なぜ発生するのかなぜ、必要となるのか、について掘り下げてみました。

- 先ず、開発部門とやりとりする際の、品質部門の種々の悩みがあり「なぜそのような悩みが出てくるのか」について考察してみました。どうやら、開発部門と、品質部門とでは、お客さまに対する視点に微妙なずれることがあることが原因であろう、という考えに至りました。
- 品質部門は、品質を常にお客様側の視点で考えており、かつ、長期的な信頼関係、という視点で見る傾向があります。一方、開発部門は、短期的なコストとデリバリに視点がいく傾向にあります。
- この認識の違いが、根底にあることにより、双方のフラストレーションがたまり、悩みが発生するのではないでしょうか。
- この悩みの解決策としては、開発部門と品質部門のゴールの共有と双方の現場を相互に理解するということが重要である、と考えます。どちらもお客様に向かって努力しているところは変わりません。

3.2 PDCAサイクルの中で肝がステップアップの役割を果たす



②肝が必要になる理由

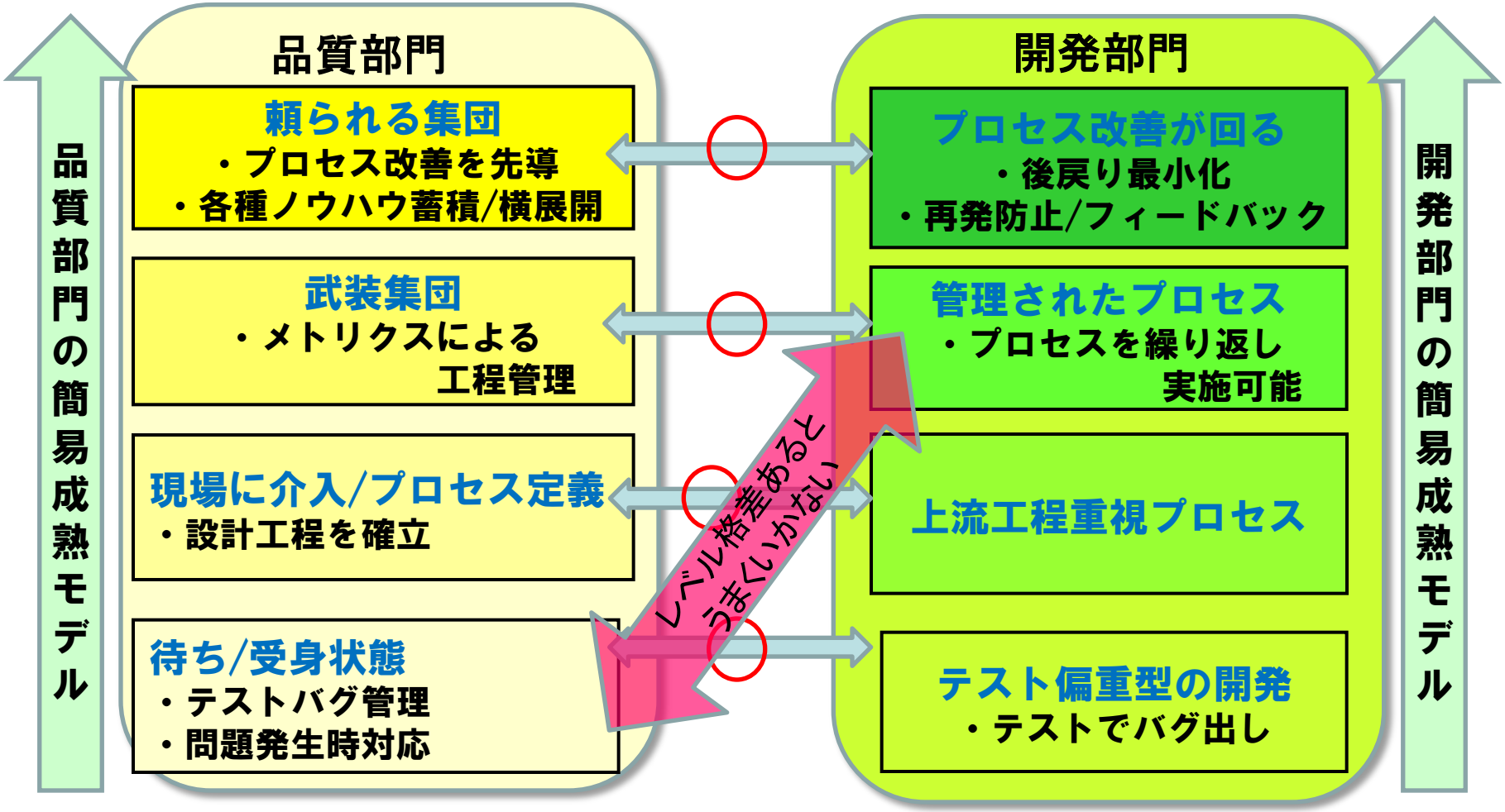
- **開発部門と品質部門の相互の期待の不一致**
- **部分最適と全体最適のサイクルにずれが生じる**

[解説]

- なぜ、肝が必要となるのかについて
- 組織で導入された仕組みやルールは、時間が経つにつれて、その導入の背景や目的が忘れられ、陳腐化したり、形骸化することがあります。これは、社内外の環境が変化し、導入時とは状況が変化したり、組織そのものが成熟して、変化が発生したりして、ある意味、避けることのできないことです。あるいは、目の前のお客さまやプロジェクトの特別な事情や状況が理由で、標準的な仕組みやルールをそのままに適用することが不適切な場合もあります。
- このような形骸化などによる、仕組みやルールの運用不全を回避するために、先人の経験に基づく「肝」が活用されます。そして、この肝が次第に仕組みやルールに取り入れられて、仕組み・ルールの修正やレベルアップが行われます。
- 私たちは、このような仕組みと肝のPDCAが存在するのではないかと考えました。
- １）開発部門と、品質部門の相互の期待の不一致の解消
- ２）部分最適と全体最適のサイクルのズレを解消
- これらのために、肝が必要となると私たちは考えました。

3.3 開発組織の成熟と品質保証活動との差から肝が生じる

品質部門と開発部門→レベル差がGAP（悩み）を生む？



開発部門の成熟レベルに最適なレベルの品質保証対策を講じる

[解説]

- 品質部門の成熟レベルについて
- 品質部門と開発部門それぞれの成熟レベルとその対応関係をひとつのものの見方として、絵にしてみました。左側が、品質部門の成熟レベル、右側が、開発部門の成熟レベルです。
- 開発部門の成熟度は、「テストでひたすらバクだしする」というテスト偏重型の開発のレベルから、しだいに、レベルアップして、プロセス改善が回るような進化したレベルまで、レベル分けできます。
- 一方、品質部門は、テストバクの管理や問題発生の都度対応するといった、待ち／受け身状態のレベルから、品質を作りこむ、つまり、プロセス改善を先導したり各種ノウハウを蓄積／横展開できる、進化した品質部門のレベルまで、レベル分けできます。
- 悩みが生じるのは、品質部門と開発部門とで、成熟レベルにギャップがある場合で、相互の期待値がズレている場合に悩みが発生します。開発部門の成熟のレベルに合わせた品質対策を講じないと、意識の齟齬が発生します。品質部門も、自身で、成熟レベルを上げる努力を行うことで、開発部門をリードしていかなければなりません。双方の切磋琢磨が必要です。
- 気をつけるべきことは、「片方だけ、自分だけ、レベルアップしても、それが自己満足になる可能性もある」ということです。

第4章　品質保証プロセスの肝

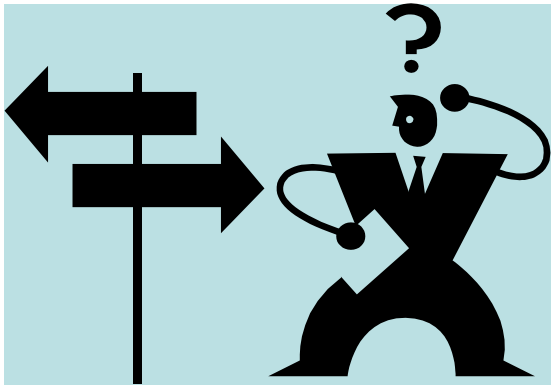
4.1 品質目標の運用プロセスの肝

4.2 メトリクスの運用プロセスの肝

4.3 テストにおける品質判断プロセスの肝

4.1 品質目標の運用プロセスの肝（１）

- 悩み
  - 品質目標がどんなものかわからない
  - 品質目標が無くても開発できるので、品質目標を立てようとしない
  - 品質目標が行動目標になってしまうので、定量的な判定が難しい
  - そもそも、品質データが集められない



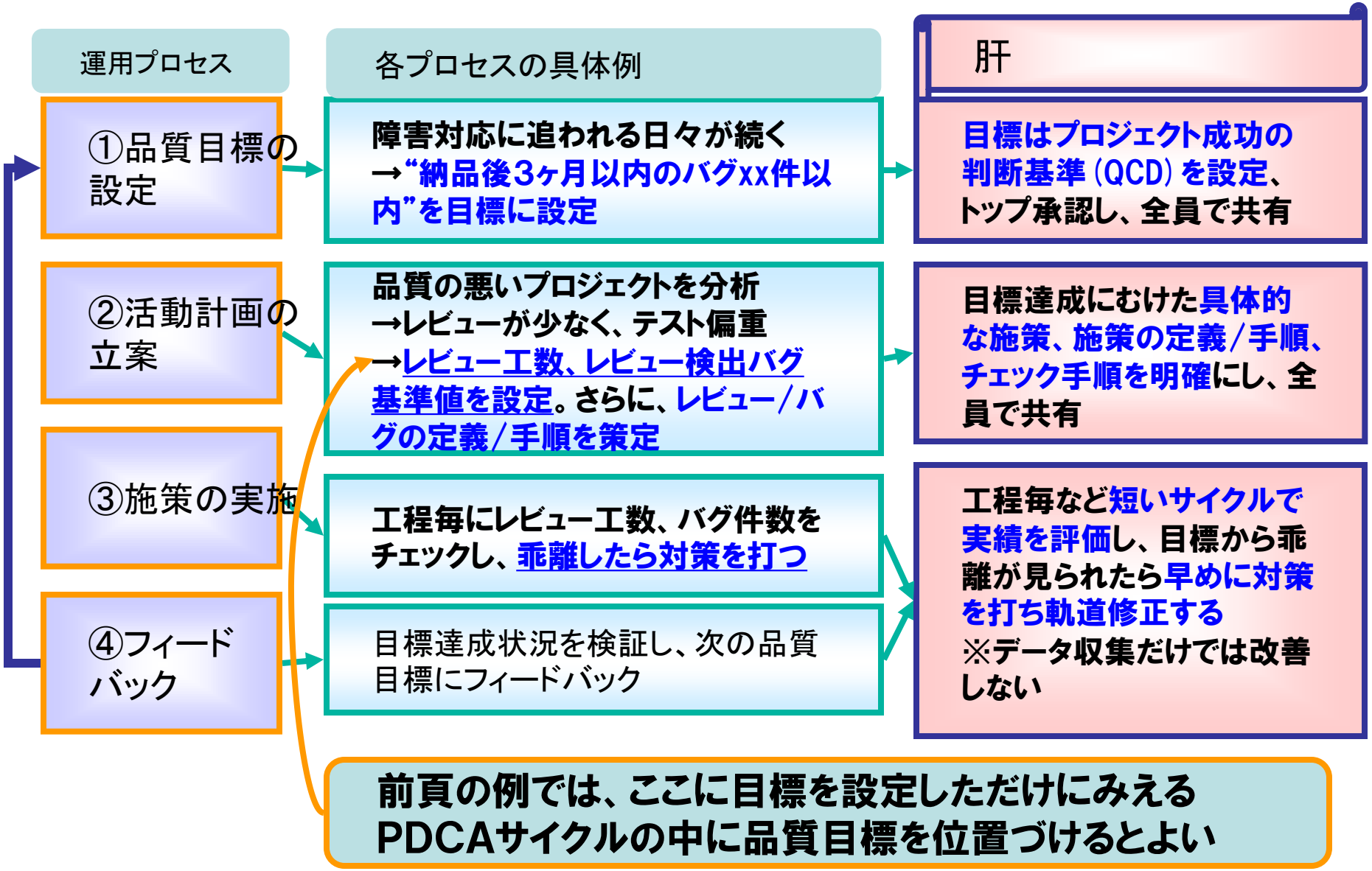
■例えば・・・

- **「レビュー工数は XX人H/ページ以上」、「バグ摘出数を XX件/規模以上」を目標と決めているが、うまくいかない**

→品質目標の運用プロセスに、問題がないだろうか？
→次頁に事例を紹介

4.1 品質目標の運用プロセスの肝（２）

- 品質目標の運用プロセスの事例



4.2 メトリクスの運用プロセスの肝（１）

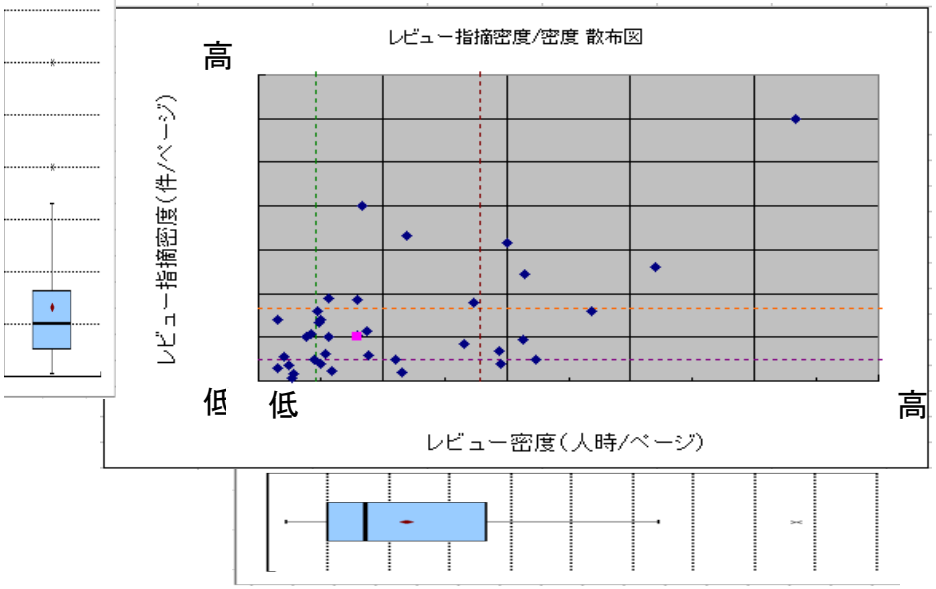
- 悩み
  - メトリクスとして何を測ればよいか？
  - 収集したデータをどう評価すればよいか？

◆肝

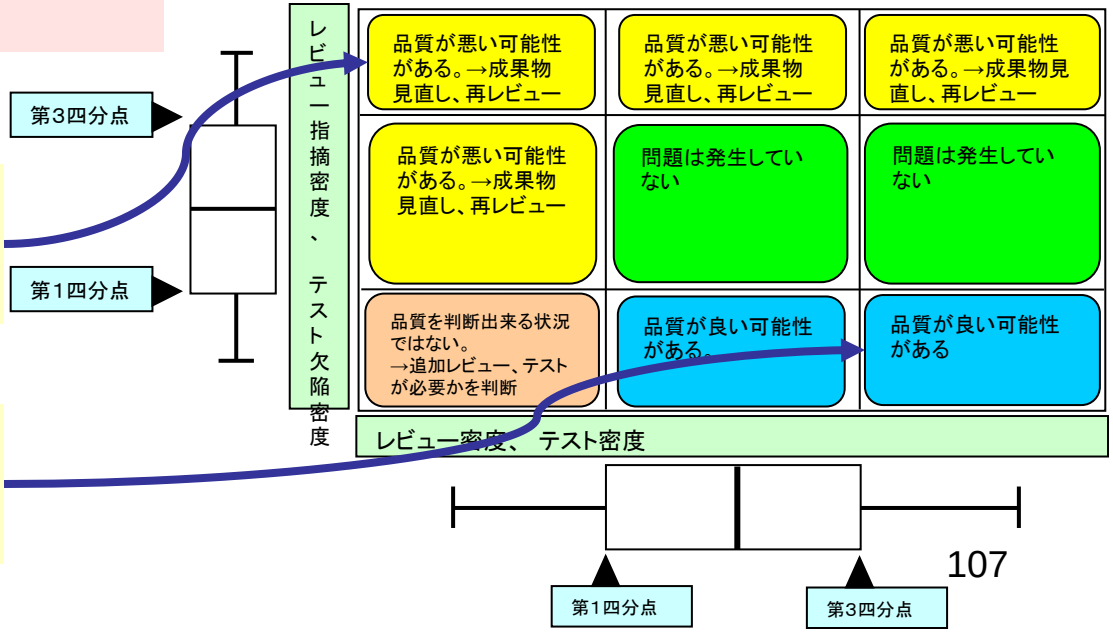
- **「データを平均値だけで見ない」**
  →個別データに“はぐれ値・外れ値”などの異常を探す
- **「データ上、何の問題もなし」**
  →逆に怪しい。成果物をみて実態を探る
- **基準値との比較だけではわからない場合が多い**
  →複数のデータを組み合わせた分析が必要
  ※次頁に事例を紹介

4.2 メトリクスの運用プロセスの肝（２）

- レビュー量が多いこと、レビューの指摘数が多いことを、ばらばらに評価しても品質を判断できない



◆ **レビュー量と、指摘数との関係を組み合わせて品質を判断**



レビュ密度：小 ＆ 指摘密度：大
→品質が悪い可能性あり

レビュ密度：大 ＆ 指摘密度：小
→品質が良いといえる

4.2 メトリクスの運用プロセスの肝（３）

◆ **バグの作り込み工程と発見工程のマトリクスから、当該工程でどれほどバグを摘出したか、次工程以降にどれほど流出しているかを把握し、どの工程の品質が悪いかを判断する**

		作りこみ工程							
		要件定義	仕様作成	システム設計	プログラム設計	コーディング	見逃し率(%)	妥当性(%)	飛び値有り
発見工程	要件定義	0					－	－	－
	仕様作成	14	14				26.3	50.0	－
	システム設計	0	5	24			0.0	82.8	－
	プログラム設計						－	－	－
	コーディング						－	－	－

メトリクス	内容	評価参考値
レビュー妥当性	当該工程のレビューで発見すべき指摘を当該工程で摘出した率	80%以上
レビュー見逃し率	当該工程のレビューで発見すべき指摘が次工程に流出した率	20%以下
試験妥当性	当該工程の試験で発見すべき指摘を当該工程で摘出した率	80%以上
試験見逃し率	当該工程の試験で発見すべき指摘が次工程に流出した率	20%以下
指摘飛び値有無	該当する工程よりも、2つ以上先で発見された欠陥の有無	無いこと

4.3 テストにおける品質判断プロセスの肝（１）

悩み

- テストを終了してよいかの判断が出来ない
  - テストで摘出すべきバグを見積もれないので、いつテストを終了してよいか分からない
  - テストが十分か過剰か分からない
  - 不安なので、期限ぎりぎりまでテストをする
  - 期限になったら、「やるだけのことはやった」とテストを終了する

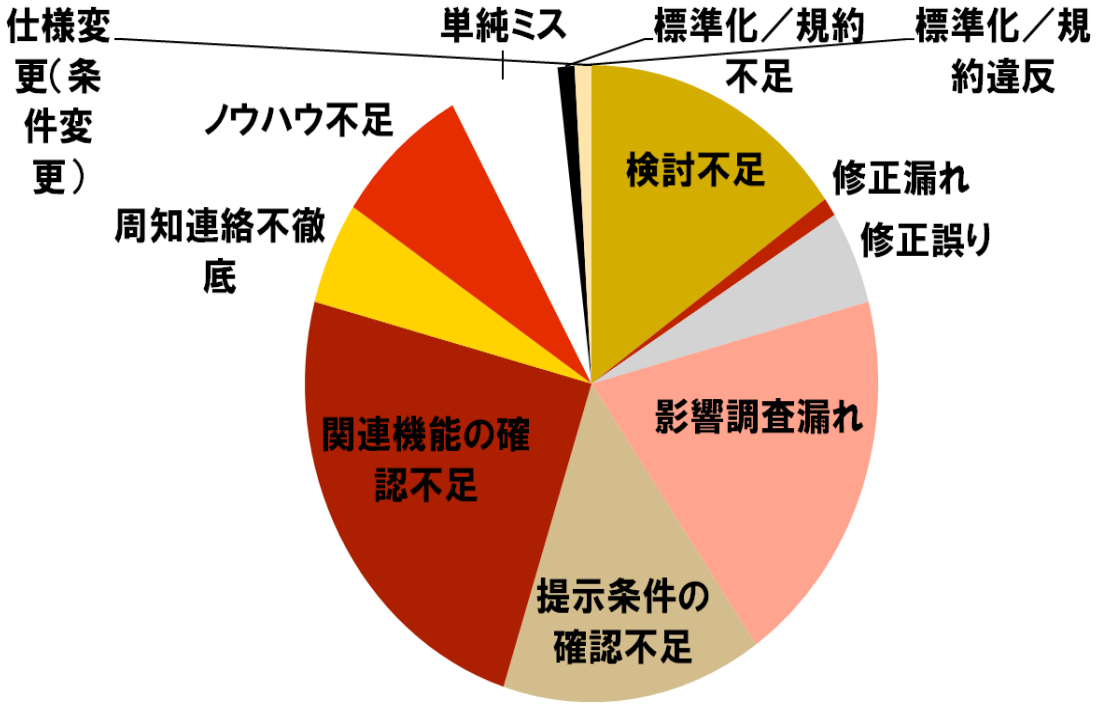
肝

- どれだけテストをするのかは、V字プロセスにより各設計書を入力としたテスト項目を作成し、全てのテスト項目を消化することを最低限とする
- さらに、バグの収束状況から、テストが十分か、追加施策が必要かを判断する
- ただし、品質の判断は単純ではなく、多次元の組み合わせ分析で判断する

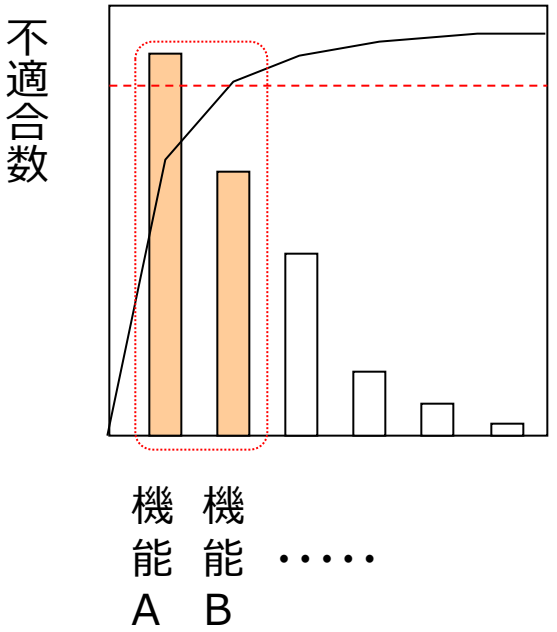
→次頁に事例を紹介

4.3 テストにおける品質判断プロセスの肝（２）

- バグの傾向分析により、品質の弱点箇所を判断する
  - バグ原因別分布、機能別分布、人別分布などに詳細化
    - 特定の箇所に集中している場合、該当箇所を再点検
    - 特定箇所がなく、全般的に同様の傾向の場合、プロセス全般を再点検



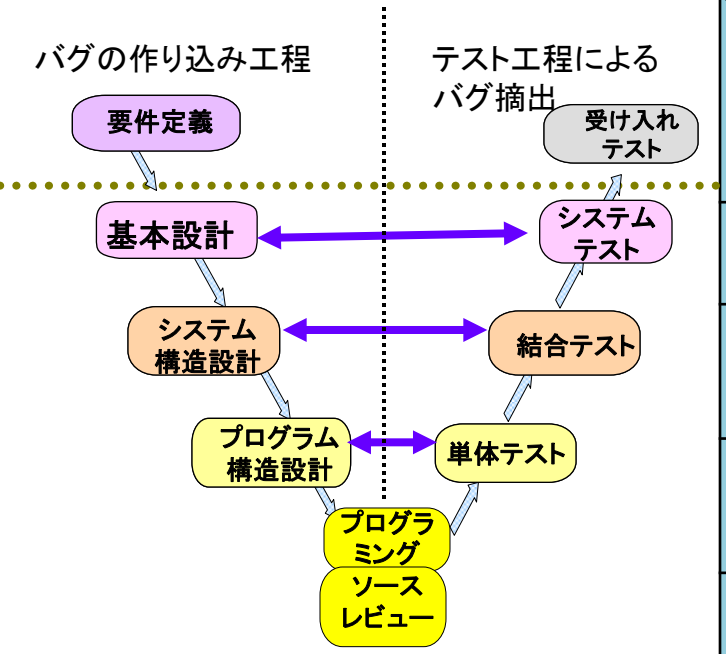
バグ原因の分布による傾向分析



機能別分布による傾向分析

4.3 テストにおける品質判断プロセスの肝（３）

- テストフェーズと、バグの作り込み工程の相関表で評価する
  - 作り込み工程に対応したテスト工程で摘出されているか？→①
  - 該当テスト工程にふさわしくない、作り込み工程のバグはないか？→②
  - レビューの方が、テストよりも多く摘出しているか？→③



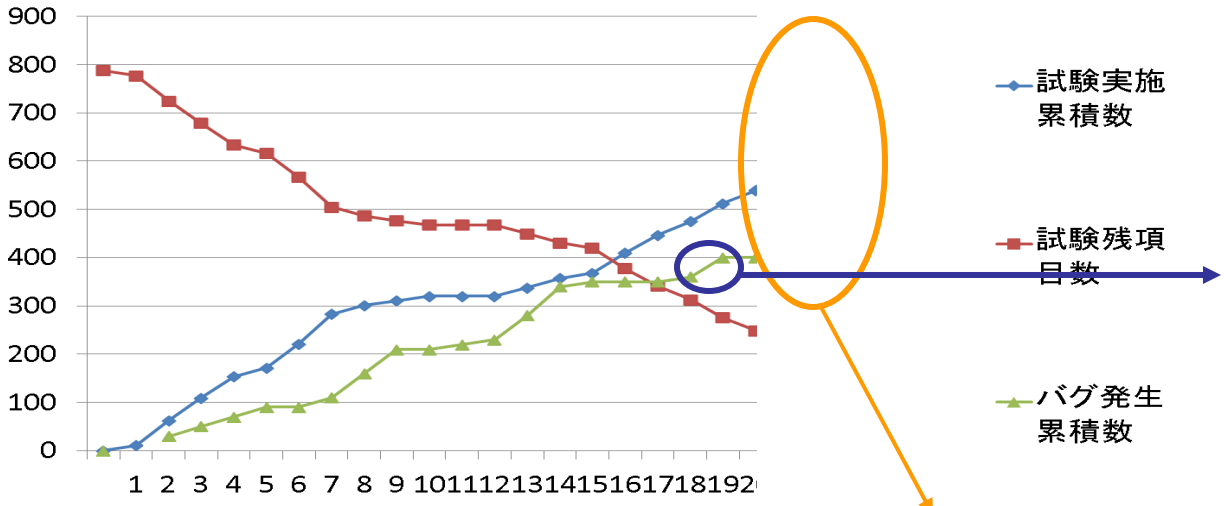
摘出工程 / 作り込み工程	レビュー ③				テスト ③		
	基本設計	システム構造設計	プログラム構造設計	プログラミング	単体テスト	結合テスト ②	システムテスト ②
基本設計 ①	75件 ①	5件	0件	0件	0件	0件	20件 ①
システム構造設計		160件	8件	2件	0件	20件	10件
プログラム構造設計			160件	40件	80件	15件	20件 ②
プログラミング				320件	180件	80件 ②	5件 ②

②の例；「システムテスト」には、作り込み工程が「プログラム構造設計」や「プログラミング」のバグがでている。当該工程のレビューや単体テストの不足が考えられる

4.3テストにおける品質判断プロセスの肝（４）

- テスト項目の消化と、バグの発生件数の収束状況を評価する
  ただし、グラフだけで判断しない
  ⇒テスト終盤で発生しているバグの内容を確認する
  なぜテスト終盤で出たのか？、なぜこれまでのレビューやテストで見つけられなかったのか？を分析して、弱点箇所を探し、対策を打つ

テスト項目消化数とバグ累積数のグラフ



経験的には、予定テストを消化し、バグの対処が全て完了しても、さらにテストを追加し、もうバグが摘出されないことを確認する

■ **開発終盤で発生したバグの内容を確認する**

① メインルートのバグがシステムテストで検出されていないか？
⇒重大なテスト漏れの懸念

② 同様な勘違いを複数のプログラムでしていないか？
⇒仕様書に欠陥があるかもしれない

③ バグの内容と、原因の分類に矛盾がないか？
⇒バグ票の信憑性に問題があるかも

※数字には表れてこない問題が見えてくる

第５章　品質保証部門の肝

5.1 失敗事例から学ぶ

5.2 品証部門と開発部門の対立事例

5.3 品証部門はどうあるべきか

(付録)品質部門の役割の違いについて

5.1 失敗事例から学ぶ（事例１）

「品質データを収集できない」 ⇒３年がかりで収集できるように

1年目

・品質データの申告に対応してもらえず。
・進捗会議では、次回は申告するよう、注意するばかり。
・開発部門からは「こんな数字遊びがなんの役に立つんだ」

ここで、メゲない

2年目

・開発部門から、少しずつ品質データが申告され始めた
・品質問題と、プロセスデータが、繋がる事例が出てきた

さらに、継続

3年目

・レビュー不足やバグ検出が低い指摘で、見直し作業
（品質データ収集と分析、品質見直しフィードバックが定着）
⇒出荷後のバグ数が減少し、開発作業のストップが減少
⇒上流工程のレビューが、自らが楽になる作業という実感

（相互の期待の不一致）
・開発部門：短期的な成果　　・品質部門：品質向上は長期戦

（肝） 品質部門は目的を明確にして、
地道な作業をブレなく言い続けることが大事

5.1 失敗事例から学ぶ（事例２）

「品質目標はお仕着せではうまくいかない」

品質指標設定時には、過去の統計値を参考にするよう指示

開発に入ると、この指標値と実績値が、毎工程乖離してしまう

品質部門は、毎工程で乖離の理由を開発部門に求めるし、
開発部門は、毎工程理由を考え報告（作文に近い）

開発部門、品質部門の双方で無駄な作業ではないかと、疑問

⇒品質目標の設定を、下記に変更

①バージョンアップの場合は、前バージョンの実績値をもとにする

②前バージョンがない場合は、類似の開発体制の実績値をもとにする

③それもない場合は、過去の統計値をもとにする

乖離が発生しても、その理由がキチント説明でき、対策が取れる。

品質部門は「全社統一でなければ」という発想になっていた。

（肝）開発部門、品質部門とも、品質目標の意味を理解して運用

5.1 失敗事例から学ぶ（事例３）

「予防処置やプロセス改善は報告したら終わりがち」

改善定着や、達成評価するまでに、１年、２年と時間がかかる。
⇒活動報告してトップが納得したら、そこで活動が停止

・プロセス改善は、報告後に継続する工数のほうがかかる
・トップが納得しても関係者全員への納得は難しい場合がある
・全体最適策を個々に適用するのは、なかなか進まない

トップの入るプロセス改善会議等で実行が完了するまで、アクションアイテムに残し、しつこくフォローすること

（肝）プロセス改善リーダーは、施策をやりきること、効果を確認するまで1年も2年も続けること、これを繰り返し組織文化にすることが最終ゴールであると、強い意志を持つこと

5.1 失敗事例から学ぶ（事例４）

「レビュー状況の評価は早めに行う」

品質部門は、レビューの評価を「工程終了時」に実施
レビュー票の記載十分性や、指摘分類の不適切さを指摘

開発部門は、「今更、レビュー票の修正は、時間がもったいないので、次回からきちんと書きます」との回答

次工程で、レビュー不足やレビュー指摘の取り違えのバグが多発し、結局、レビュー強化した

⇒レビューの評価を、下記に変更

①記録票のフォーマットは、開発の初期段階で確認

②記録票は、工程中盤になったら、確認を開始し早めの指摘

（肝）　工程終盤の指摘では、開発部門に押し切られるか、手戻り作業を強いるだけ。問題点を早めに指摘することで、工程内でのフィードバックが可能になる

5.1 失敗事例から学ぶ（事例５）

- 「納得されるデータ分析手法」
  - 品質部門は、対象製品毎に品質分析のやり方を考えていた
    - 品質分析の担当者毎に、指摘観点が異なる、分析に時間がかる、報告書のレビューにも時間がかかる、という問題があった。

⇒品質分析の手法を下記の流れと観点に標準化

①指標値分析（指標値との乖離度合で評価）

②傾向分析（指摘内容の集中傾向で評価）

③起因・摘出工程分析（障害の作りこみ工程と摘出工程のバランスで評価）

④テストの網羅性分析（テスト項目の網羅度合を評価）

⑤障害収束状況分析（テスト期間内で、収束傾向にあるかを評価）

⇒同じ手法・レベルの分析により開発部門から納得性の高い評価を得た

標準化により、新たなメンバーでも一定水準の分析が可能、
品質報告書の作成やレビュー時間が短縮、
同じ手法なので、複数の案件比較や、前回との比較が可能

5.1 失敗事例から学ぶ（事例６）

- 「バグ分析したらプロセス改善の方向をミスリード」
  - バグ分析し、経験の浅い開発者が構成管理ツールの使い方を誤ったため、新人でも間違えないように作業手順を明文化し徹底するプロセス改善を報告
  - トップ上司からダメだし
    - バグの発生個所からコードレビューで見つけるべきと判断できる
    - プロセスデータでは、レビュー工数もテスト項目数も少ない
    - このバグも、基本的なテストケースで発見可能である
    - やるべきことができていないことを予防すべき
    - ツール使い方のプロセス改善など、どうでもよい
  - 品質部長の反省
    - バグ分析を、狭い視野で既存のパターンにあてはめて結論しようとした
    - “やるべきこと”ができていないことを見抜く力が大切

問題の本質を見抜く力、ブレない客観的な分析が大事

5.2 開発/品質部門の対立事例（１）

「次工程へ進む前に品質強化をするかどうか」

品質部門⇒次工程に進む前に品質強化をやるベキと主張

開発部門⇒工程遅延につながるので、次工程に進むと主張

（対立の背景）

品質部門⇒後工程や、納入後の品質を懸念

開発部門⇒目先の「コスト」「納期」に意識が集中

（対処の「肝」）

品質部門は、より多くの品質データ、個々の障害の内容まで踏み込み、具体的な指摘を行うことを心がける

⇒今、対策を実施することのメリットが理解できるように協議

5.2 開発/品質部門の対立事例（２）

「プログラムレビューをやるかどうか」

-品質部門は、品質確保のためプログラムレビューやるベキと主張

-開発部門は、ペアプログラミングなのでレビューも兼ねてると主張

（品質部門は、どう対処したか）

⇒品質部門は、ペアプログラミングが成立する組み合わせなのか
また、できているのかを確認してから判断し、できているなら指摘を撤回し、次工程で効果を確認する

（対立の背景）（対処の「肝」）
品質部門は、先ず、建前から入る
開発部門は、どうやれば出来るか、という現実路線で考える
⇒落とし所を協議することで、解決へ導く

5.2 開発/品質部門の対立事例（３）

「詳細設計工程をやるべきかどうか」

-品質部門は、詳細設計をやるベキと主張

-開発部門は、前工程の構造設計が詳細設計を兼ねると主張

（品質部門は、どう対処したか）

⇒品質部門は、構造設計書の詳細度合を確認してから判断する。ある程度プログラムロジックまで規程可能な粒度の記載か確認し、できているなら、指摘を撤回し、次工程で効果を確認する

（対立の背景）（対処の「肝」）
品質部門は、先ず、建前から入る
開発部門は、どうやれば出来るか、という現実路線で考える
⇒落とし所を協議することで、解決へ導く

5.2 開発/品質部門の対立事例（４）

「品質部門の指摘に対応するのは負担と感じる」

-品質部門は、品質データを分析して問題指摘するのは当然

-開発部門は、品質データを整理し提出、指摘に対応するのが負担

（品質部門は、どう対処したか）

⇒開発部門への教育を行い品質意識を改善する。

予め、開発部門に品質部門への対応工数を提示しておく、

品質確保のための、必要工数を見込んでもらう

（対立の背景）（対処の「肝」）
品質部門は、何を、いつやるのか、予め開発部門に提示し、作業内容や量を開発部門に見積もってもらう。（特にSI開発の場合）
開発部門は、品質確保の作業内容を理解しておく
⇒やるべきことを品質部門、開発部門の相互に認識しておく

5.2 開発/品質部門の対立事例(5)成功事例

品質部門と開発部門は対立していない。
⇒品質を良くしたい意識は同じと感じている。

（対立しない背景）
品質部門⇒作業計画は、開発部門の責任者と合意をとる
開発部門⇒責任者は、品質の重要性を一番認識している。
ただし、開発部門の責任者も、納期と品質をてんびんにかけて、悩む場合もある。

（対処の「肝」）
品質部門は作業計画（目的や判断基準）を、予め開発部門の責任者に提示して、合意しておく。
⇒開発部門の責任者が、品質部門の言い分を理解して尊重する文化を創っておくことが肝要。

5.3 品質部門はどうあるべきか(1)

品質部長の信念/ポリシーを集めました

お客様先で品質問題を出さない（基本事項）

常にお客様視点で考え、判断する

品質について広く深く誰よりもわかっている

品質のエバンジェリスト（社内、社外（お客様含め）どちらも）

お客さまをバーチャルに想定できる

是正処置と予防処置（基本事項）
是正⇒製品の品質を磨く
予防⇒プロセスにフィードバックし組織を強くする

行動パターン⇒ブレない、地道、愚直、真摯

5.3 品質部門はどうあるべきか(2)

品質部門のスキルとコアコンピタンスは何だろう？

必要なスキル

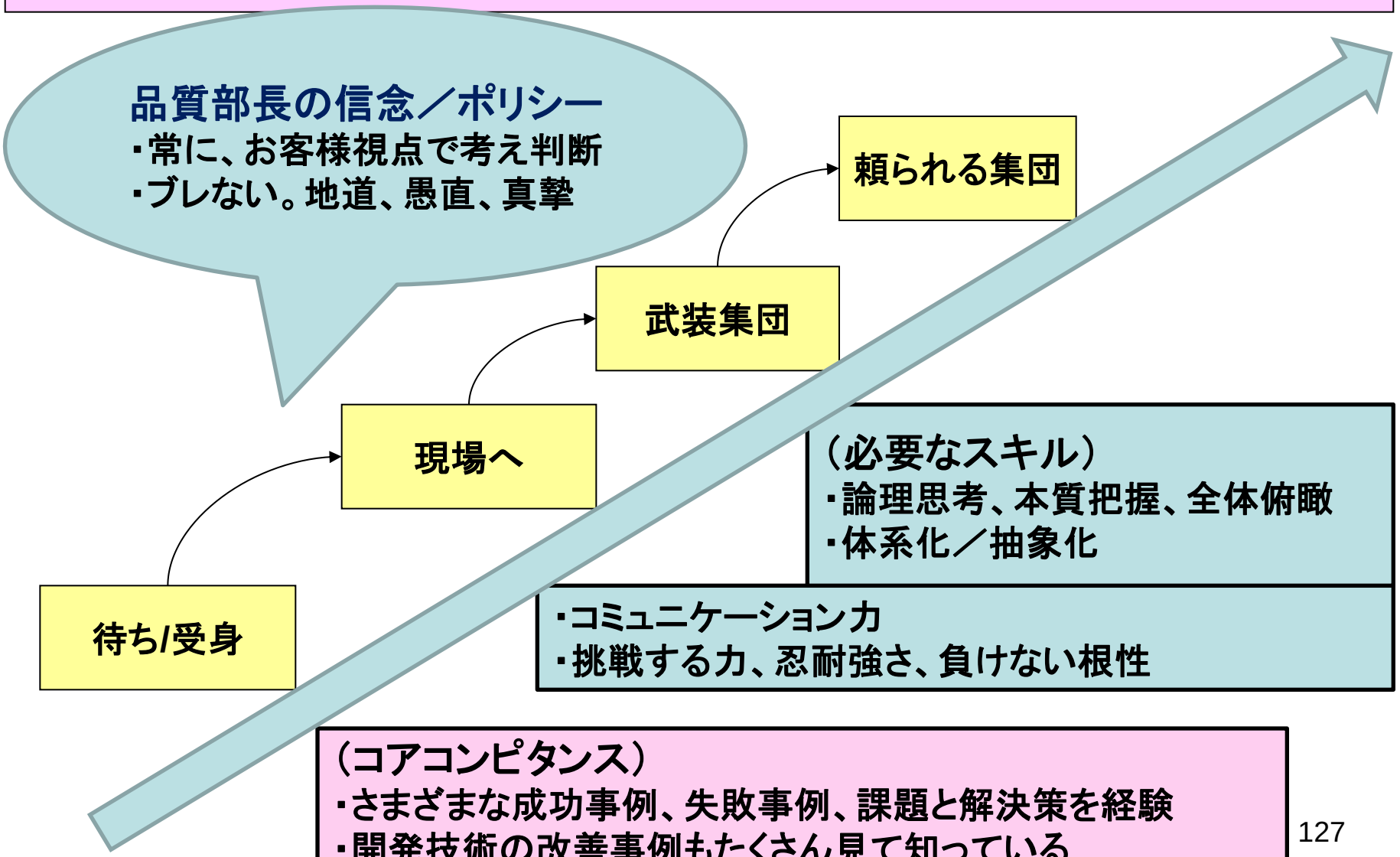
- 論理思考、本質把握、全体を俯瞰できる、体系化/抽象化できる、問題の重要度を客観的に位置づけられる
  →“おかしいな”、“変だな”、と判断できることが大事
- コミュニケーション力、挑戦する力、忍耐強さ、負けない根性

コアコンピタンス

- **さまざまな成功事例、失敗事例、課題と解決策を経験**
  → 傾向がわかる。過去事例を蓄積。統計分析。データで語れる。
- **開発技術の改善例もたくさん見て知っている**
  → 組織内の横展開ができる唯一の部門

5.3 品質部門はどうあるべきか(3)

品質部門の成熟モデルを考える



5.3 品質部門はどうあるべきか(4)最後に

- 品質保証業務ってセンシティブかつ創造的な仕事？
  - 何をやっても確からしいことがなかったり、開発部門から疎まれたり、不安にかられることが多い
  - なぜなら、
    - 目に見えない敵（品質）を相手にしている
    - 人間を相手にしている
    - “これをやれば絶対”はない
  - だから、形式的な対応は失敗のもと
    - 人間は、保守的なもの。変化には抵抗がある
    - 人間（開発者）の心に響く働きかけがなければ変化は起こらない

悩むのは止めて、行動をおこしましょう
そのとき、「ソフトウェア品質保証の肝」を右手に。

(付録)品質部門の役割の違いについて（１）

1）プロダクト事業

開発体制
・同じ体制で継続して開発

開発費用
・何本売れたら、で設定

検査部門
・プロダクトの品質を評価
・費用は製品原価に含む

・メリット：製品に対する検査部門の貢献度が分かり易い（出荷後バグの減少、等）
・デメリット：検査部門のバグ出しに頼る傾向。出来上がってからの品質向上になりがち。

2）SI事業

開発体制
・案件の度に、発足・解散

第三者監査部門
・プロジェクトの成否（QCD）を評価
・費用は全社間接費用

開発費用
・お客様との契約で確定

お客様が品質を評価

・メリット：上流工程からQCDについて口出しができる。
・デメリット：第三者監査部門の貢献度が分かりにくい。開発部門から見ると他人行儀。

（付録）品質部門の役割の違いについて（２）

	プロダクト事業の例	SI事業の例
会社文化	品質問題が発生すると、購入ユーザ全てに対して修正処理を行わなければならず、その手間を考えると、品質優先の文化が根付いている	品質問題が発生した場合でも、基本的に一品物である為に、対処療法的な対応が多く、真因の追究が浅い。新技術への対応やｺｽﾄが重視される傾向がある
要求仕様	事業主が市場等を調査の上、仕様を決定しており、品証部門も当初から仕様検討に係っている事から製品仕様の理解が深い	直接のお客様からの引合、仕様決定、受注迄は開発部門中心で進められる為、品質部門の仕様の理解が浅い。開発部門の技術ﾚﾍﾞﾙによってお客様の要求把握に差が発生する
納期	直接のお客様契約をしているわけではないので、納期については多少の融通がきく。	直接のお客様との契約の為、納期に対しては厳しく要求される。
検査	（比較的）検査工程は確立され、製品知識、ノウハウを持った検査員によって蓄積された手法に基づき実記にて検査が実施される	基本的にお客様要求書や設計書を検査員が理解した上で検査を実施する（検査はあくまでプロセスが手順通りに実施されたかの確認のみの場合もある）
品質分析	定量的なデータを用いて、組織全体の平均との差や製品横並びの比較、過去との比較や、多くの開発部門との経験などから、現在の品質を判断して提示することを期待されている	定量的のデータは取得しているものの作番による品質にばらつきがある為、有効なデータとして活用できていないケースが多く、一般的な傾向を述べるにとどまる
顧客対応	品質問題発生時には、品質部門が先頭に立って解決する	品質問題発生時には、開発部門が先頭に立ち解決を行う。品質部門は解決の支援のみのケースも多い

第6章　品証カルタ

- ソフトウェア品質保証の肝をカルタで表現してみました。
- 短い文で言いたいことを伝える難しさを感じながらも、品証部門長の本音の“あるある感”を軽い遊び心も織り交ぜて作成しました。
- 皆さまの部署において、参照したり、自部門でもカルタを作成するなどにより、教育や実務にお役立ていただければ幸いです。

品証カルタ①

い	• いけません　後出しじゃんけん　倍返し
ろ	• ロジックの　シンプル故に　バグ無しに
は	• パレート図　眺めて対策　優先度 • はずれ値に　真の課題が　隠れてる
に	• 日本式　体力勝負じゃ　もう限界 • 日本式　こだわり品質　ガラパゴス
ほ	• 本当の　品質良くする　現場の声 • 本当の　品質良くする　トップの背
へ	• 変更に　良かれと応え　悪化させ
と	• 取り返せ　なぜあの品質で　出荷した

品証カルタ②

ち	• 力ない　技術者レビューで　成長し
り	• リスクだけ　集めて満足　大失敗 • 「良心」も「高品質」の　一要素
ぬ	• 抜きん出た　不具合の数　誰入れた
る	• 累積の　バグ収束は　もう一歩
を	• 置き去りの　バージョン管理　致命傷

品証カルタ③

わ	• わかってる つもりだけでは 役立たず • 忘れるな PDCA 回すこと
か	• 限りなく テストは続く いつまでも • 隠しても EVMで 全てバレ
よ	• 世の中の 技術の動き 我知らず
た	• 体力で 工程回復 皆ダウン
れ	• レビューした 問題点は なんだった？ • レビューの場 仕様検討 徹夜した
そ	• 想定外 リスク管理で 想定内

品証カルタ④

つ	• 鶴一声　出荷はしたが　バグ多発
ね	• 寝ても夢　品質炎上　プロジェクト
な	• なぜなぜを　5回もやって　本質は？
ら	• ランダムに　起きる事故にも　類似性
む	• 無理難題　これの解決　SE冥利 • 無視するな　予兆を示す　異常値に

品証カルタ⑤

う	• うれしいな　今日もトラブル　コールなし
ぬ	• 急いでた　プロセス飛ばし　後戻り
の	• 野放しの　工程管理で　納期遅延
お	• 多すぎる　チェックリストで　バグ減らず
く	• 苦労する　ツール導入　３年越し • クローンの　なれのはてが　スパゲティ
や	• やる気だけ　分析無しで　空回り
ま	• またも事故　再発防止　繰り返す

品証カルタ⑥

け	• 軽微ミス　客先流出　大トラブル
ふ	• 複雑な　プロセスそれは　何のため • プロジェクト　ピンチは改善　チャンスだよ
こ	• これからの　経営左右　クレーム費
え	• エラそうに　品質問題　語ってる
て	• テストした　不具合つぶした　なぜ悪い

品証カルタ⑦

あ	• ありえない　上から目線の　アドバイス
さ	• さあ稼働　でもその前に　再チェック
き	• 気を付けて　分析大切　メトリクス
ゆ	• 勇気だし　ダメ出ししたら　総スカン
め	• メトリクス　何のためかと　問うてみる
み	• 魅力的　品質求めて　日々努力
し	• 信頼は　日々の活動　積み重ね • 失敗の　もとをたどって　未然予防

品証カルタ⑧

ゑ	• 絵に描いた　ルール･基準値　価値は無し
ひ	• 品質の　本質 顧客の　満足度
も	• モデル見て　そのまま適用　甘いです
せ	• 先人の　知恵を生かして　高品質 • 設計を　より良くするのが　レビューです
す	• スミマセン　謝るだけでは　進歩せず
ん	• んーまだだ　納得いくまで　追究を

E N D