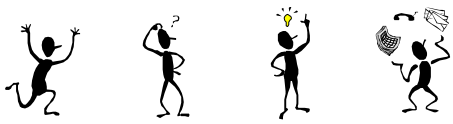
JaSST札幌

テスト･エンジニアほど素晴らしい仕事はない！

●知的で、創造的な世界へようこそ●

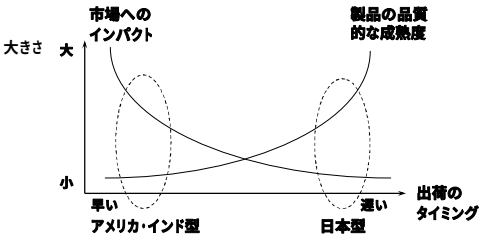
2006年10月17日
東海大学情報理工学部
山浦 恒央 (YAMAURA, Tsuneo)
yamaura@keyaki.cc.u-tokai.ac.jp

第1部： ソフトウェア開発を取り巻く背景



1. 日本型 vs アメリカ型の製品出荷戦略

1.1 従来の日本型



早期出荷でシェアを確保するため、品質が未成熟のまま出荷

コスト<品質<納期

未成熟製品出荷による手戻り作業やコストを削減するため、品質の成熟を待って出荷

コスト<納期<品質

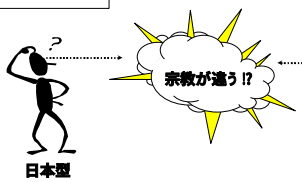
1. 日本型 vs アメリカ型の製品出荷戦略

ソフトウェア開発で優先順位が違う。

「日本型」 対 「アメリカ･インド型」=「品質」 対 「市場性」

品質こそ全て ！

品質は製品の一部。
売れなきゃ意味がない



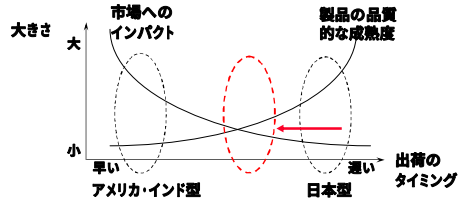
日本型

アメリカ・インド型

1. 背景その1： 日本型 vs アメリカ型の製品出荷戦略

1.2 これからの日本

･マーケティング戦略の変化：重厚長大から軽薄短小へ (数を売るには、販売タイミングが非常に重要。例えば、ボーナス商戦に間に合わないと売れないない等)
･顧客の強い要求 (例：会計年度に間に合わせる)

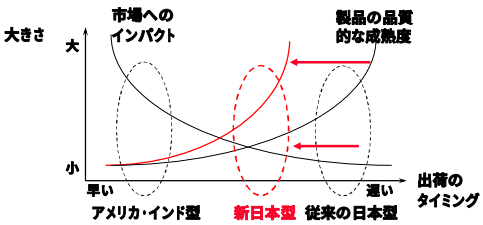


コスト<品質<納期

1. 背景その1： 日本型 vs アメリカ型の製品出荷戦略

1.2 これからの日本

･リリース時期を早めつつ、品質を上げねばならない。



コスト<品質＝納期

2. 背景その2： ソフトウェア開発方式の変化

・コスト削減、納期短縮のため、全て自社開発するのではなく、既に市場に存在する製品を組み込む。

→ 組み込む製品にバグがあっても、タイムリーに修正してもらえない。

→ 「バグ」を仕様としなければならない場合が多い。

・技術の特殊化、特化により、他社に、製品の一部の開発を依頼せざるを得ない(例えば、携帯電話用のブラウザ)。

→ 製品の総合的な品質は、他社発注製品の品質次第。

→ 自社の品質保証方式を強制できない (強制しても、やってくれない)。

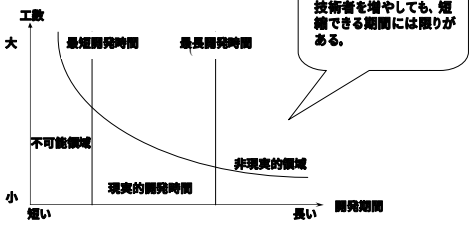
7

3. 背景その3:. ソフトウェア開発関係式

ソフトウェア開発での工数、開発期間、機能総量、生産性、品質の関係式

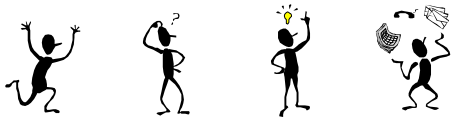
$$\left(\frac{\text{工数}}{\beta}\right)^{\frac{1}{3}} \times (\text{開発時間})^{\frac{4}{3}} = \frac{\text{機能の総量}}{\text{生産性}}$$

β＝定数



8

第2部： テストのモデル



9

1. テストとは何か？

・テストは、コーディング終了後、ソースコードに対して実施するだけではない。全ての開発フェーズで必要なプロセス。

・ソフトウェアが正しくないことを証明することは非常に容易だが、正しいことを証明するのは、非常に困難。

⇒この困難さを工学的に解決する方法がソフトウェア工学

10

2. テストとデバッグの違い： そっくりだが、全く別物

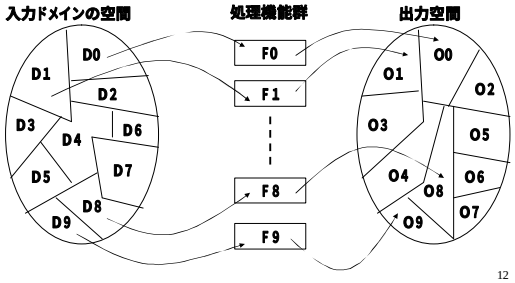
テスト
・ソフトウェアが正しくないことを証明すること。
・重大な不良があれば、以降のテストを中断してよい。
・入学試験のようなもの。
⇒足切り点に達しないと、その時点で不合格。以降の試験は受けられない (受けても意味がない)。

デバッグ
・ソフトウェアの正当性を証明すること。
⇒これは、非常に困難！
・全ての機能をチェックする必要あり。
・模擬試験のようなもの。
⇒全種目を受験し、自分の長所、弱点を洗い出し、合格できるレベルにもっていく。

11

3. プログラム処理の概念モデル

バグは、次の2通りしかない。
●処理機能が正しくない。
●入力ドメインの境界が正しくない。



12

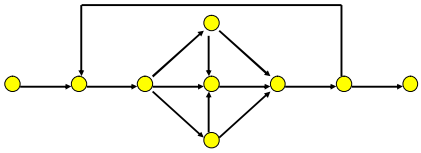
4. 入力ドメイン空間とテスト空間の大きさ（テスト可能性）

●単純な構造のプログラムでも、全入力データの全組み合わせ（全パス数）は、天文学的な値になる（下記の例では、1件1秒で実行できても、100万年かかる）。

●有限のリソースで、全組み合わせを検証するのは不可能。

$$5+5^2+5^3+\cdots+5^{18}=4.77\times10^{12}$$

1～18回ループ



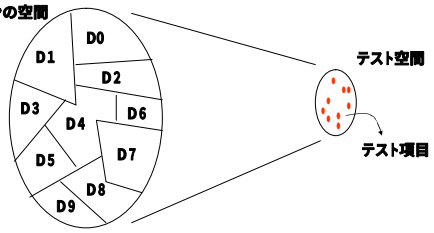
13

4. 入力ドメイン空間とテスト空間の大きさ

入力ドメイン空間に比べると、テスト空間は、圧倒的に小さい。

⇒ 太陽の質量と、素粒子の質量ほど違う？

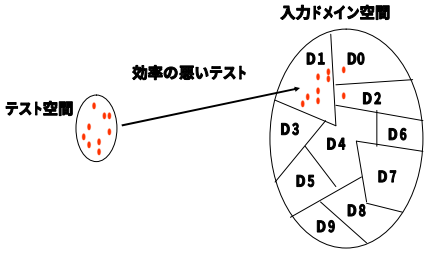
入力ドメインの空間



14

4. 入力ドメイン空間とテスト空間の大きさ

完全なテストができないのなら、テスト空間をどのように「入力ドメイン」に割り当てて、効率の高いテストを実施するかが非常に重要。

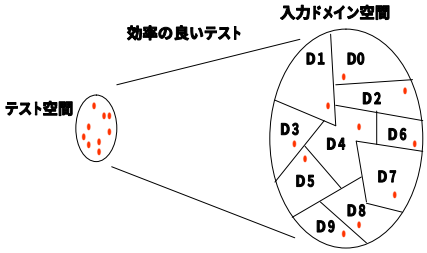


15

4. 入力ドメイン空間とテスト空間の大きさ

●全ての入力ドメインをカバーしなければならない。

●バグのありそうな箇所を重点的にテストする必要がある。



16

4. 入力ドメイン空間とテスト空間の大きさ

●テストだけで、全てのバグは取れない。

⇒テストの時間、人員は有限なので、100%のバグ除去は不可能。従って、バグ摘出率の良いテスト項目が必要。

⇒製品にバグが残ることを前提に、リスク管理的に、重大事故につながるバグを重点的に摘出する必要あり。

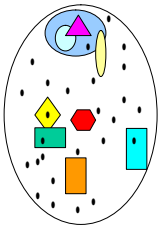
⇩

●仕様、設計段階から、バグを作りこまないために、開発ライフサイクルを通じての品質確保プロセスが必要。

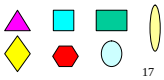
この品質管理プロセスを監督するのがQAエンジニア。

QAエンジニアは、品質の番人。

入力ドメイン空間

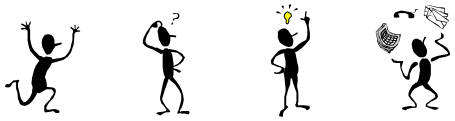


バグ



17

第3部： テスト・プロセスの進化



18

１．従来のテスト・プロセスの進化過程

第１世代：プログラマが自分でテストする。

第２世代：設計部内にテスト・チームがある。

第３世代：社内に、設計部とは独立したテスト・チームがある。

19

2. 従来の品質保証の進化過程 (その1)

第1世代:プログラマが自分でテストする。

・「自分のプログラムは正しく動くはず」症候群

・単なるデバッグの延長

●デバッグとテストの違い

デバッグ:ソフトウェアが正しいことを証明すること。

・テスト項目を全件実行させ、検出したバグを修正し、修正を確認する。

テスト:ソフトウェアが正しくないことを証明すること。

・テスト項目を1件実行させ、致命的なバグがあった場合は、「品質未成熟」とみなし、以降のテストを打ち切ってよい。

20

2. 従来の品質保証の進化過程 (その2)

第2世代:開発チーム内にテスト・チームがある。

「自分のプログラムは正しく動くはず」症候群は、ある程度、除去できるが、…

・テスト担当者(もとは、開発者)にテストの技術や能力がない。

・開発チームの意見に影響され、やはり、「プログラムは正しく動く」と思い込む。

・品質に問題があっても、開発チームの意見や、顧客の要請に押し切られる。

・開発者だけでなく、テスト担当者に、「テストは、ソフトウェアを開発する上での必要悪」との考えが広がる。

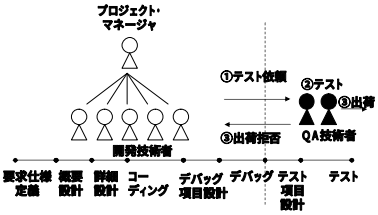
→ いわれなき誤解による、テスト担当者の士気の低下

21

2. 従来の品質保証の進化過程 (その3)

第3世代:開発チームとは独立したテスト・チームがある。

現在の、テスト・プロセスの最先端。



第3世代の品質保証プロセス

22

2. 従来の品質保証の進化過程 (その3)

第3世代のテスト・プロセスの利点

・開発チームとの「親密な関係」がなくなり、「性悪説」にもとづいて、テストを実施できる。

・テスト専門の高度な技術集団なので、効率よくテストを実施できる。

・品質が不十分であるのに、出荷時期が迫った場合でも、開発チームや顧客と対等の立場で(あるいは、それ以上の立場で)、談判できる。

23

2. 従来の品質保証の進化過程 (その3)

第3世代のテスト・プロセスの欠点

・テスト部門が、開発部門とは独立にテスト項目を設計し、実施するため、開発側が同じテスト項目を作り、テストを実施していても、有効利用できない。

・テスト部門は、開発側が実施するデバッグと類似の作業をすることになるため、人と時間が重複することになる。

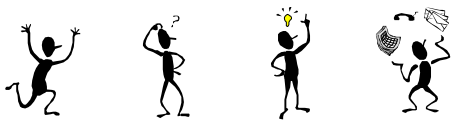
・開発とテストを組織上、明確に分離するため、「他人のソフトウェアのテスト」というテスト部門の行為は、開発側の「プログラム作成」に比べると、新規性に欠け、創造的でないとのメンタリティが、テスト担当者と設計者の両方に醸成されている。 ⇒ テスターの士気の低下

・テスト部門が蓄積してきた対象プログラムに固有のテストのノウハウ、技法、着眼点を開発側と共有することが難しい。

・開発部門に、品質管理をテスト部門に依存する傾向が出る。

24

第4部：テスト･プロセスの課題と解決案



25

1.現状の品質管理の課題

(1) ＱＡエンジニアの地位が低い？
- ＱＡは、創造的な仕事ではない？
- ＱＡエンジニアはプログラマより能力が低い？
- ＱＡエンジニアは、プログラマの尻拭い？
- ＱＡエンジニアは、システム開発能力がない？

(2) ＱＡエンジニアの士気が低い？

(3) 設計部とＱＡ部は、同じ製品のテストをするが、テスト項目は全く独立に設計・作成し、実行するので、時間と資源の無駄？

(4) 「早い検査」の実現

26

2.テスト・エニジニアに対する不当な誤解

- テストは、創造的な仕事ではない？
- テスト・エンジニアはプログラマより、技術力、能力が低い？
- テスト・エンジニアは、プログラマの尻拭い？
- テスト・エンジニアは、システム開発能力がない？

- テスト・エンジニアは、プログラマより、はるかに高い技術力、能力が必要。
- 非常に創造的な仕事。
- システムを容易に開発できる能力がないと、テストは無理。

27

2.ＱＡに対する誤解

- 例えば、原子力発電所を設計するのは簡単だが、安全性を実際にテストするのは、極めて困難。この困難を可能にするのが、高度な技術力を持ったテスト・エンジニア。
- 例えば、期末テストで、テストを受ける生徒が開発エンジニアであり、テスト問題を作る先生は、テスト・エンジニア。
- 例えば、予備校の先生が、①生徒の長所、短所を見つけ、②弱い科目、苦手な項目を指摘し、③時系列的に弱点が是正される状況をモニタリングし、④志望校に合格させるというプロセスは、まさに、テスト・プロセスそのもの。

28

3.「現状の品質管理の課題」の解決案

(1) 品質開発マネージャ制度の導入による早期開発
(第４世代のテスト・プロセス)

(2) 設計部とＱＡ部の資源共有・再利用によるテスト工数削減
(第４世代のテスト・プロセス)

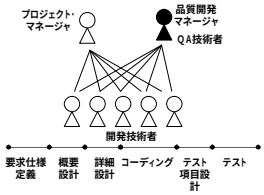
(3) 資格として、「品質エンジニア」を新設し、
ＱＡエンジニアの士気を向上。

29

4．品質開発マネージャの導入によるテスト・プロセスの改善

(1) 第４世代のテスト・プロセス

- 第3世代のテスト･プロセスを改善し、品質保証の効率と独立性を上げるために、「品質開発マネージャ」を導入する。
- 「品質開発マネージャ」を中心としたテスト・プロセスを確立する。
- 品質開発マネージャが、開発プロジェクトの品質を全般的に指揮・制御する。



第4世代のテスト･プロセス

30

４.品質開発マネージャによるテスト・プロセスの改善(続き)

(2) 「品質開発マネージャ」が担当するテスト・プロセス

・品質管理の最高責任者として、開発プロジェクトに関与する。

・プロジェクト・マネージャは製品開発の設計、工程管理、品質管理(デバッグ)を実施。品質開発マネージャは、プロジェクトの品質管理(テスト)の責任を持つ。

・品質開発マネージャは、品質に関し、プロジェクト・マネージャを指揮する(会社と経営コンサルタントの関係)。

・従来のテスト・エンジニアは、受身であったが、品質開発マネージャは品質確保に向けて、指揮権を発動し、積極的に関わる(QA関連の技術者の品質保証技術、モラル、地位を向上させる)。

４．第４世代のテスト・プロセスの概要

(1) 開始条件
開発プロジェクトを開始した時点。

(2)終了条件:
以下を考慮し、ソフトウェア製品が出荷品質に達したと品質開発マネージャが判断した時点。

・対象ソフトウェアの分野で要求される品質基準
・リスク
・不良発生の分析結果
・リリース日程

４．第４世代のテスト・プロセスの概要

(1) ソフトウェア開発プロジェクトに、QA部門から派遣された品質開発マネージャを一人設置する。

４．第４世代のテスト・プロセスの概要

(2) 最終製品のテスト

●第3世代のテスト・プロセスでは、開発側とは独立に、テスト・チームが最終製品のテストを実施

●第4世代のテスト・プロセスでは、開発側の技術者がデバッグとテストを実施し、品質開発マネージャがそれを指揮する。

・プログラマが作成したデバッグ項目の内容をテストの専門家としての視点で精査し、デバッグ項目の追加、変更、削除をプログラマに指示する。

・プログラマのデバッグ項目消化状況、バグ発生状況を分析し、品質を予測する。これをもとに、デバッグ項目の追加、変更、削除を指示する。

・デバッグが終了した時点で、品質開発マネージャは、最終製品の品質を評価する。これが、事前に設定した終了基準に合致している場合、意図した品質が確保できたとみなし、従来のテストを実施することなく、出荷する。

５．資格、「品質エンジニア」の新設

情報処理技術者の上級資格として、「品質エンジニア」を追加する。

(1) テクニカル・エンジニア
・エンベデッド・システム
・システム管理
・データベース
・ネットワーク
(2) アプリケーション・エンジニア
(3) プロジェクト・マネージャ
(4) システム・アナリスト

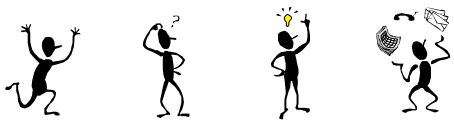
(5) 品質エンジニア
・入社10年目程度の経験と技術力があること。
・QA対象システムに最適のQA方式を導入できること。
・システムの正当性、信頼性、機密性、安全性、可用性等を検証、分析する。

５．資格、「品質エンジニア」の新設（続き）

新資格、「品質エンジニア」の追加により、期待できる効果

(1) 日本のQA技術者が、共通の用語や概念を共有 (現在、開発技術の進歩は業界全体で進んでいるが、テスト技術は、組織内で進み、他社との技術交流がない)。
(2) テスト技術者の大幅な地位向上。
(3) テスト技術者のプロ意識の育成・醸成。
(4) テスト技術者の士気向上。
(5) テスト技術者の技術力のスタンダードを設定。
(6) 会社や組織のテスト能力、技術の客観的な尺度。

第5部： 日立ソフトでのテスト・プロセス



37

1.日立ソフトにおける品質データ:

開発フェーズごとの摘出バグのパーセンテージ

机上 デバッグ	単体 デバッグ	連動 デバッグ	統合 デバッグ	QAチーム のテスト	顧客 サイト
21.5%	35.1%	28.0%	6.1%	9.3%	0.02%

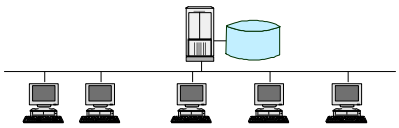
10,000 件のバグのうち、顧客サイトで出るのは2件。

38

2.日立ソフトにおけるQA方式

特別な方法は使っていない。以下のように昔からの方法を踏襲。

(1) プログラマは、テスト項目を設計し、テスト項目を作成し、実行する。
(2) ソフトウェア製品のライフサイクルを通じて、バグの統計データを収集する。
(3) プロジェクト・マネージャは、バグの統計データを分析し、プロジェクトを管理する。
(4) 設計部と独立したQAチームが製品をテストする。



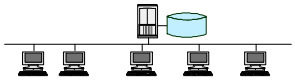
39

(1) プログラマは、テスト項目を設計し、テスト項目を作り、実行

a. プログラマがコーディングを終了すると、テスト項目の設計を開始。

b. テスト項目は全てドキュメント化する。ドキュメント化をすると、以下のような利点がある。
- テスト項目の設計は、仕様を別の視点から見直すことと同じ。この見直しで、プログラム不良だけでなく、仕様関連の重要バグも見つかる。
- テスト自体の品質を測定できる。
- 「テスト件数／KLOC」の計測により、開発者の機能理解度を測定できる。
- 同じテストを何回も繰り返し実行できる。
- テストの実行は、別の担当者にまかせられる(プロジェクトの日程遅延を回復するために投入された「素人エンジニア」にも、テストの実行可能)。

c. テストの自動実行化(テスト・スイート)を目指す。



40

(1) プログラマは、テスト項目を設計し、テスト項目を作り、実行 (続き)

テスト項目設計のガイドライン

a. 基本は、ソースコード 5 - 15行で テスト項目1件
オンライン・ネットワーク系 : 5-10 行で1件
言語プロセッサ系 : 8-12 行で1件
バッチ系 :10-15 行で1件

b. 正常ケース : 60% 以下
異常ケース : 15% 以上
限界・境界 : 10% 以上
環境(別OS等) : 15% 以上

41

(1) プログラマは、テスト項目を設計し、テスト項目を作り、実行 (続き)

マトリックスによるテスト項目の設計

入場料: 6歳以下: 無料, 7歳～12歳: 500円,
13歳～18歳: 800円, 19歳以上: 1,000円

テスト項目 ID		AD001	AD002	AD003	AD004	AD005	AD006	AD007	AD008
年齢	ZZ	X							
	-1		X						
	0			X					
	6				X				
	12					X			
	18						X		
	19							X	
	999								X
入場料	0円			X	X				
	500円					X			
	800円						X		
	1000円							X	
	エラー	X	X						X
分類		異常	境界	正常	正常	正常	正常	正常	異常
机上テスト		5/05	5/05	5/05	5/05	5/05	5/05	5/05	5/06
マシンテスト		6/01	6/01	6/02	6/02	6/02	6/02	6/03	6/03

42

(1)プログラマは、テスト項目を設計し、テスト項目を作り、実行 (続き)

a. マトリックスによるテスト項目の利点
- 入力条件の組合せを網羅できる。
- 機械的にテスト設計が可能

b. マトリックスによるテスト項目の欠点
- テスト項目数が増える(特に、シンタックス・チェック)。
- 複雑な条件の記述が難しい(自然言語によるテスト項目記述も必要)。

テスト項目 ID		A0001	A0002	A0003	A0004	A0005	A0006	A0007	A0008
年齢	ZZ	X							
	-1		X						
	0			X					
	6				X				
	12					X			
	18						X		
	19							X	
	999								X
入場料	0円			X	X				
	500円					X			
	800円						X		
	1000円							X	
	エラー	X	X						X
分類		境界	境界	正常	正常	正常	正常	正常	境界
机上テスト		5/05	5/05	5/05	5/05	5/05	5/05	5/05	5/06
マシンテスト		6/01	6/02	6/02	6/02	6/02	6/02	6/03	6/03

(2) ソフトウェア製品のライフサイクルを通じて、バグの統計データを収集

プログラマは、全てのバグを文書でプロジェクト・マネージャに報告する。

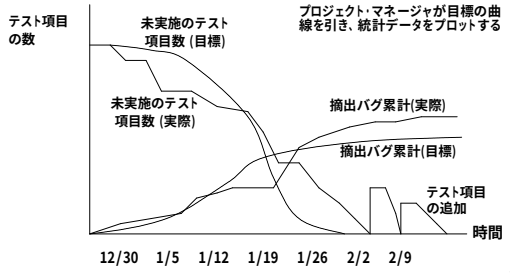
a. 報告開始時期
コーディングが終了し、コンパイル・エラーがなくなった時点(机上デバッグに入る条件が整ったとき)

b. 報告内容
- バグを摘出したとき
ID番号、バグの症状、バグの摘出者、摘出年月日、バグの重要度等

- バグを修正したとき
バグがあったモジュール名、バグの原因、修正者、テスト担当者、修正前コーディング、修正後コーディング等

(3) プロジェクト・マネージャによるバグ・データ分析

- 品質向上をするための最善の戦略を決定する。
- 製品のリリース日を予測する。

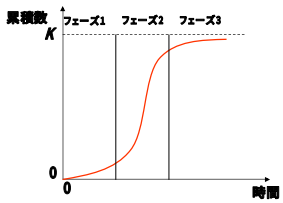


(3) プロジェクト・マネージャによるバグ・データ分析 (続き)

分析する内容

(1) モジュールごとの不良分析 (90%の不良は、10%のモジュールに集中する傾向にある)
(2) 機能固有の不良解析
(3) 個人ごとの不良分析 (個人の不良作りこみの癖を分析)

その前に、Gompertz Curve による予測の課題 (その1)

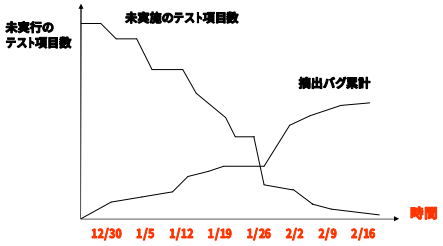


事象の成長過程 (例えば、人口増加) をユニバーサルなS字曲線として表現し、数式化したもの。
- フェーズ1:胎動期 (準備段階)
- フェーズ2:成長期 (急激に成長する段階)
- フェーズ3:停滞期 (成長が止まり、漸近線に近づく段階)

その前に、Gompertz Curve による予測の課題 (その2)

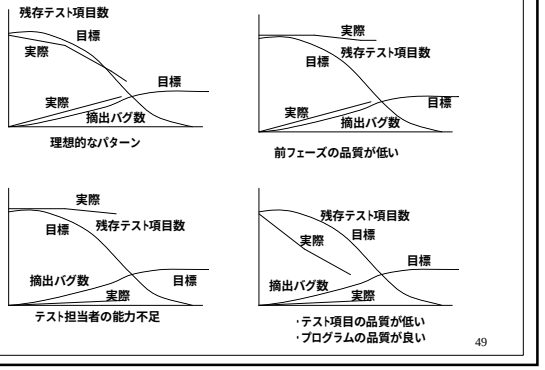
単位時間設定が難しい

⇒ 1日に1時間しかテストを実施しない日と、18時間も実施する日を同じ「1日」としてプロットすると、正確な予測ができない。
⇒ 例えば、2時間単位で、消化したテスト項目、摘出したバグ数をプロットするなど、「正規化」が必要だが、実際に、きちんと決めるのは、困難。



(3) プロジェクト・マネージャによるバグ・データ分析 (続き)

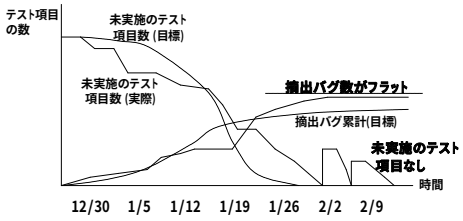
統計データのパターンによる問題点の分析



49

(3) プロジェクト・マネージャによるバグ・データ分析 (続き)

製品のリリース(ＱＡ部への)条件
(1) 未実施のテスト項目がない。
(2) バグの摘出が頭打ちになった。
(3) バグが全件、修正されている。
(4) 48時間耐久テストに合格した。



50

番外編： プロセスの落とし穴@ソフトウェア能力成熟度モデル (CMM)

成熟度の5つのレベル

成熟度低 (芸術的)

- ●レベル1： initial
  - ⇒ 開発プロセスとして、初期段階 (プロセスがない)
- ●レベル2： repeatable
  - ⇒ 開発プロセスを反復可能
- ●レベル3： defined
  - ⇒ 開発プロセスを組織で定義している
- ●レベル4： managed
  - ⇒ 開発プロセスを管理している
- ●レベル5： optimizing
  - ⇒ 開発プロセスを最適化している

成熟度高 (工業的)

51

番外編： プロセスの落とし穴@ソフトウェア能力成熟度モデル (CMM)

質問： CMMレベル5を取得した会社は、インドが世界で圧倒的に多い (インドのソフトウェア開発会社の自慢)。インドのソフトウェア品質は、現実に、それほど良いと思えないが？

答え： CMMは、「品質を向上させるプロセスがあるかどうか」をチェックしているだけで、品質の絶対的な尺度は考慮しない。
例えば、リトルリーグの野球チームでも、投手のローテーションを考慮し、定期的に選手の健康管理を実施すれば、CMMレベル5を取得できる。一方、大リーグの最強チームでも、選手の管理を何もしていなければ、CMMのレベルは1。
CMMでは、チームの絶対的な投手力、打撃力(絶対的な品質)と、チーム内の選手管理(プロセス改善能力)は、別物と考えている。

52

まとめ

ソフトウェアのテストは、知的で、創造的な仕事！！

- テスト・エンジニアは、プログラマより、はるかに高い技術力、能力が必要。
- テストは、非常に創造的な仕事。システムを容易に開発できる能力がないと、テストは不可能。
- 品質は、エンジニアの良心であり、誇り。テスト技術者は、品質の守護神。
- テストだけで、品質保証は不可能。開発のライフサイクル全体を通した品質制御プロセスが必要（模擬試験で、自分の欠点を克服するイメージ）。

53