マンガでわかる
ソフトウェア開発
データ分析

データ分析事始め

データ分析FAQ

(参考)アジャイルメトリクスFAQ

超合本版**38**編

独立行政法人情報処理推進機構

データ分析事始め　目次

データ分析基礎編

分析データ観察編

特別編

画像提供：いらすとや

データ分析FAQ　目次

基本的な質問

詳細な質問

(参考) アジャイルメトリクスFAQ　目次

アジャイルメトリクスについてよく寄せられる質問を記載していますが、その回答に対して、アジャイル開発に関わる方々からのご意見をお願いします

aa

マンガでわかる
ソフトウェア開発
データ分析
事始め

ソフトウェア開発分析データ集2022
の正しい読み方と賢い使い方

【解説】データ分析

データ分析はソフトウェア開発の道標になり、信頼性向上や生産性向上に役立ちます。データ分析はこれらのすべての始まりになります。

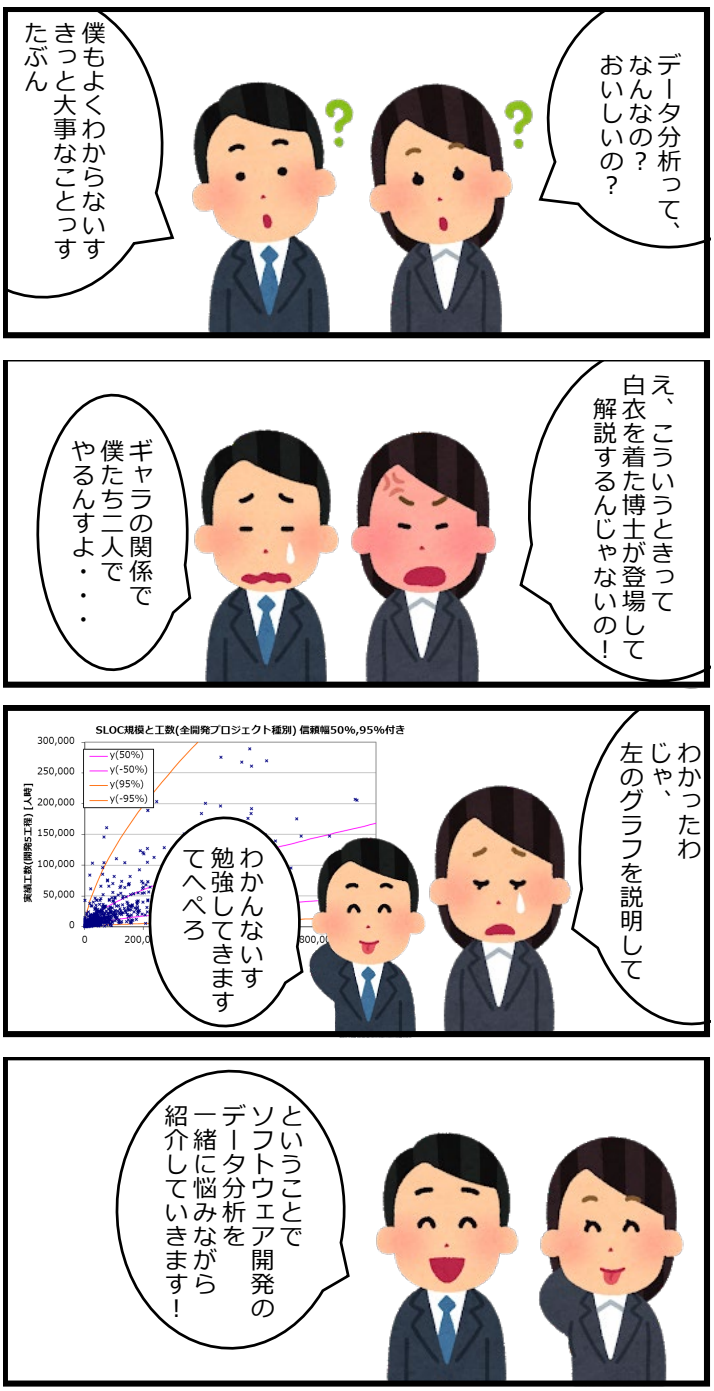
しかし属人的側面が大きいソフトウェア開発にあってはデータ分析はうまく行かず、またデータ分析の結果と同じようには、ソフトウェア開発が進まないことが多いです。

そして影響する要因が多いソフトウェア開発において、データ分析そのものは難しいものになります。得てして間違った分析をしてしまうことがあります。そしてその分析結果を信じてしまうことがあります。

また悪いことにデータ分析には多くのコストが掛かります。特に開発現場のコスト負担が大きくなります。

ここではこのコストを上回る効果を得られるようにソフトウェア開発のデータ分析をしていくようにします。

低コストで高効果なデータ分析を目指していきます。



・てへぺろ
てへっと照れて，舌をぺろっと出すこと。

【解説】信頼幅

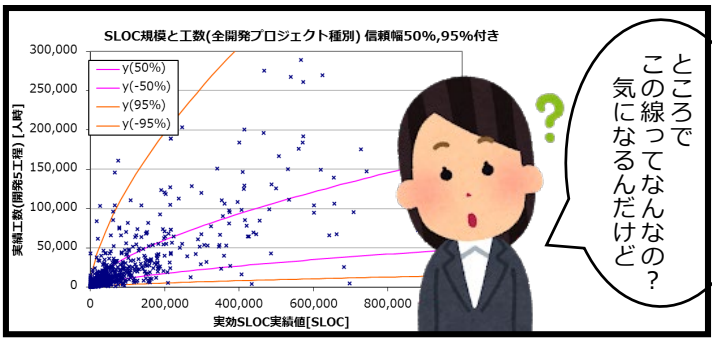
データ分析を有効に使うものとして、リスク対策があります。そのときによく使われるのが信頼幅です。

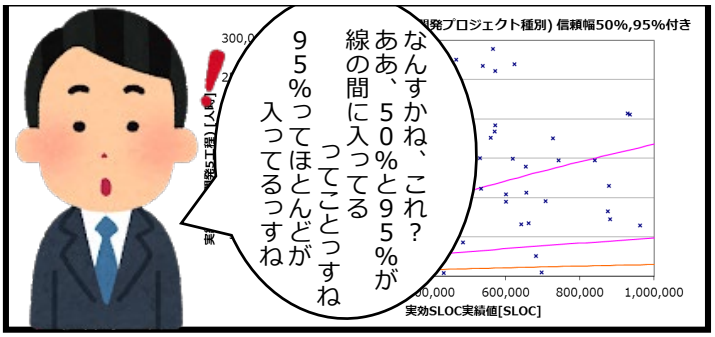
よく使われる信頼幅の線は50%と95%があります。50%の信頼幅の線の中には50%のデータが入っています。95%の信頼幅の線に入っていないデータはたったの5%です。

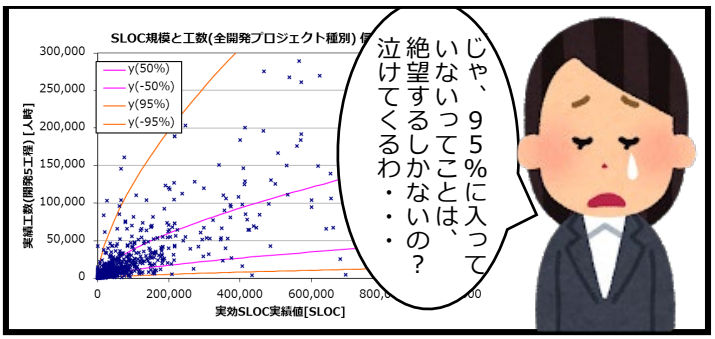
もしあなたのプロジェクトのデータが、この5%になってしまったら、どこかにリスクがあるとして、原因を追究した方がいいでしょう。そしてその原因が妥当なものかどうか、許されざるものかを判断することになります。

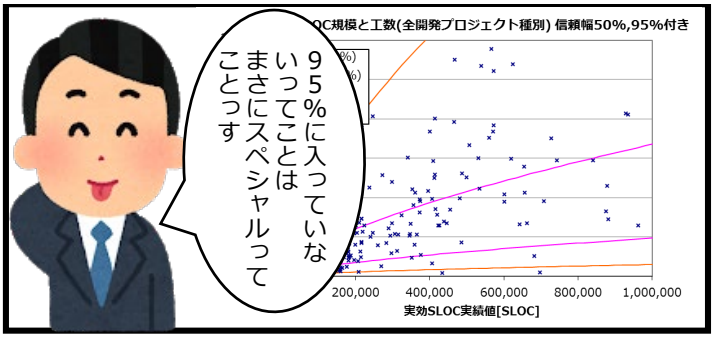
逆に信頼幅の中に入っていたとしても、それは偶然かもしれません。おかしな兆候があれば、信頼幅に入っているかどうかだけでなく、ヒアリングなどをした方がいいでしょう。

信頼幅の線、気になる









【解説】箱ひげ図

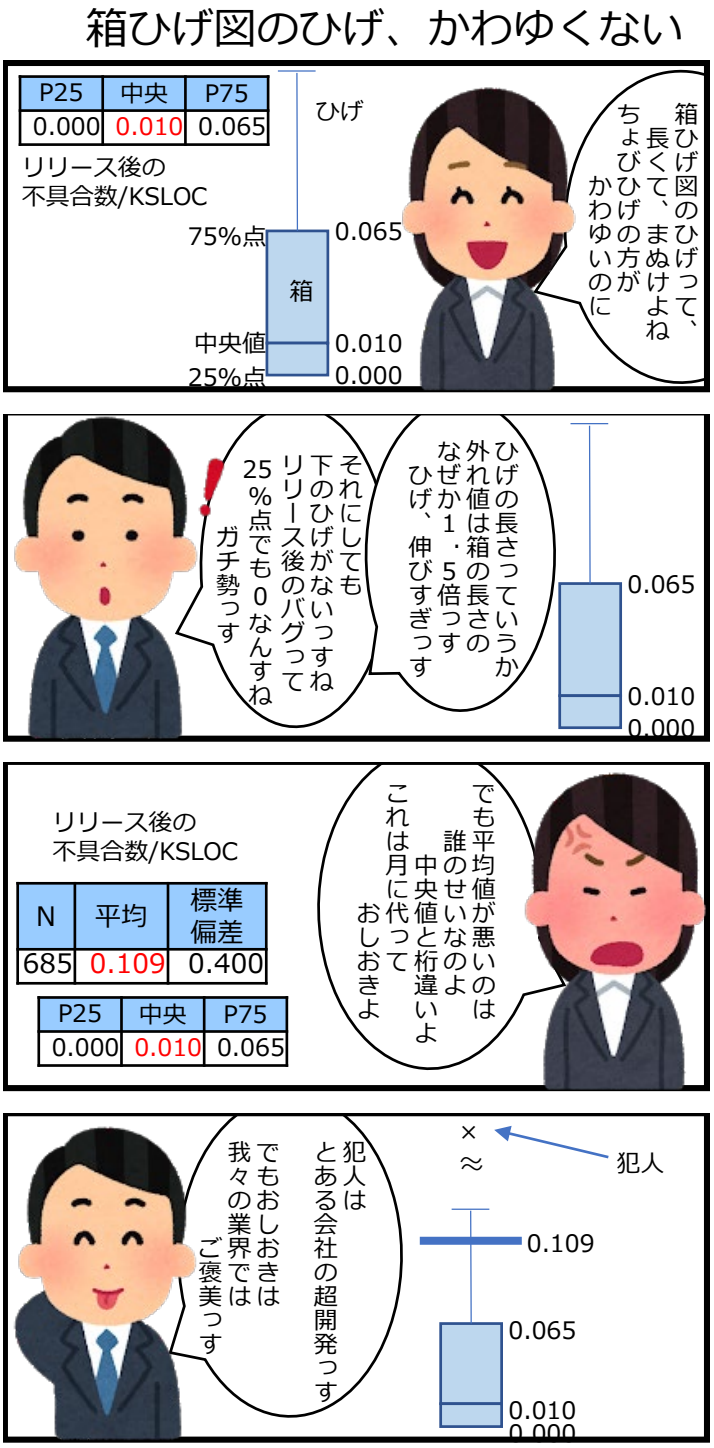
データを順に並べて25%地点(P25)の値から75%地点(P75)の値までを箱と表現し、中央値に線を引きます。箱の長さの1.5倍以上を外れ値として、外れ値でない最大値と最小値から箱までをひげで表現します。この箱ひげ図で分布の概観が一目でわかるようになります。

またP25、中央値(P50)、P75の3点で分布を4か所に分類できますので、3点の値が書かれた表を四分位表と呼びます。

右のマンガで表した四分位表と箱ひげ図は、実際のリリース後の不具合数/KSLOCのデータです。これは対象年度が全年度で、新規開発でのSLOC規模での不具合密度です。

これから中央値では0.010件/KSLOC、つまり10万行で1個のバグが出ていることがわかります。平均値では1万行に1個です。これをバグの相場観として覚えておくといいでしょう。

バグは平均値と中央値が大幅に違います。これはバグはいくらでも多く出ますが、少ないときは0個よりも小さくならず、不均衡になります。このようなときは中央値を用いてベンチマークするのがいいでしょう。



【解説】散布図と箱ひげ図

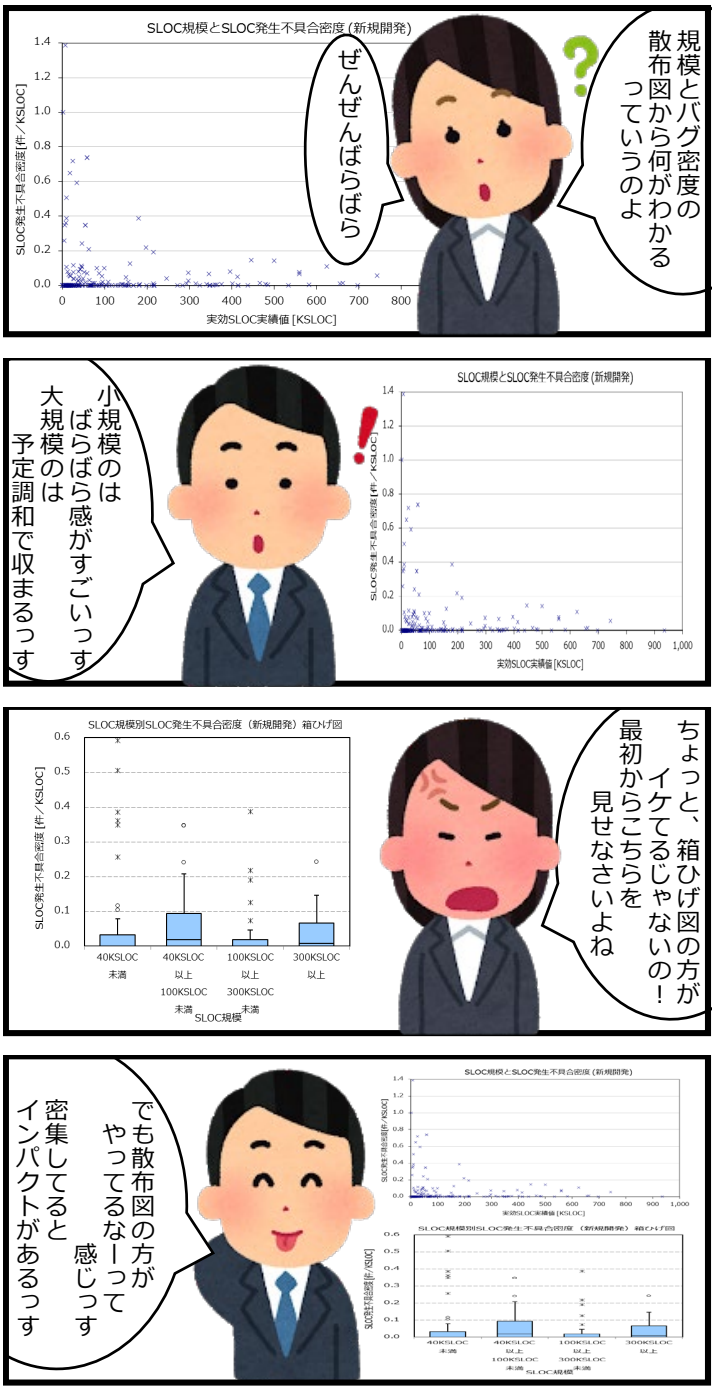
SLOC規模あたりのバグ密度（リリース後の発生不具合密度）は分散が大きく、きれいな正規分布ではありません。マンガのように散布図を見てもばらばらであることがわかります。

それでも強いて特徴を挙げるとすれば、散布図と箱ひげ図からは、小規模のSLOCのときは分散が大きく、大規模になれば分散は小さくなっているように見えます。

これは大規模のときは、リリース後にバグがいっぱい出ると大惨事になりますから、リスク対策として、バグ密度が一定の範囲になるようにコントロールされ、一方、小規模のときはリリース後にバグがいっぱい出ても小規模だからカバーできるということかも知れません。

また散布図と箱ひげ図では同じデータであっても、見え方と感じ方が違ってきます。概要を掴むには散布図で、統計値をわかりやすく見るためには箱ひげ図がいいでしょう。

散布図はぜんぜんばらばら



【解説】中央値と平均値

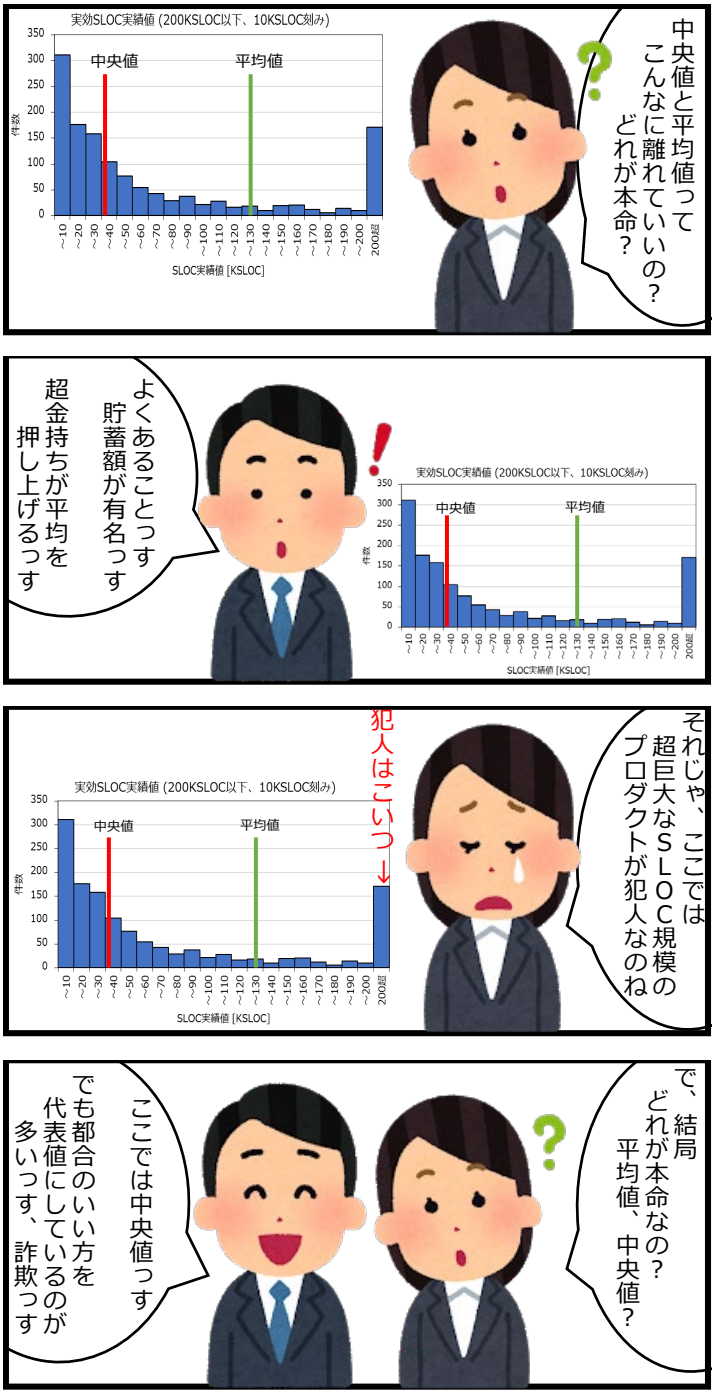
データ全体の特徴を代表する値として、平均値や中央値、さらに最頻値などがあります。

正規分布をしているデータであれば、これらの代表値は一致しますが、マンガのような分布であれば、これらの値はばらばらになる可能性があります。

マンガの分布はソフトウェアプロダクトのSLOC規模の分布ですが、一方向に大きく延びる分布で、巨大な特異なデータがグラフの右端にあります。これらがデータ全体の平均値を押し上げていて、中央値との差を大きくしています。

今回のような分布のときは、特異なデータによって引き上げられる平均値よりも中央値を代表値とする方が、データ全体の特徴を表しています。

どれが本命なの？



【解説】生産性

ソフトウェア開発のデータ分析をするときに、一番、気になるのが生産性です。

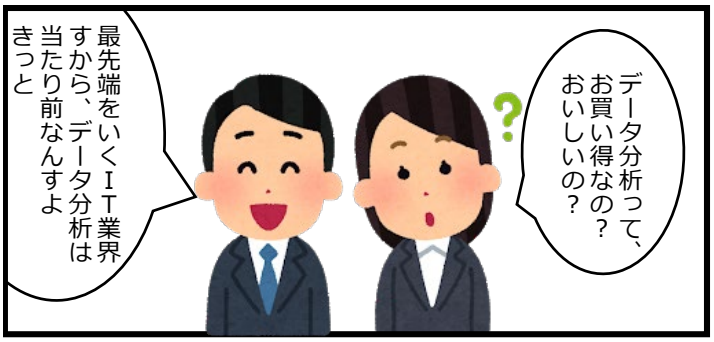
ソフトウェアの生産量の定義は困難ですが、ここではプログラムのSLOC (ソース行数) 規模を生産量としてみます。

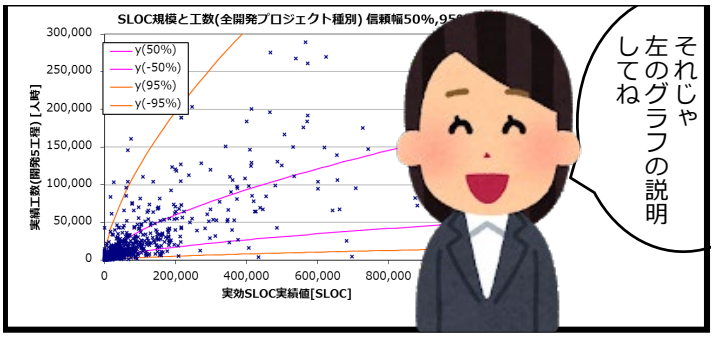
そこで実際のソフトウェア開発現場から生産性のデータを収集すると、右のグラフのように分散が大きくなります。まさに個々のプロジェクトによって、生産性は大幅に違い、生産性に影響する要因は多数あり、これがデータ分析を難しくしています。

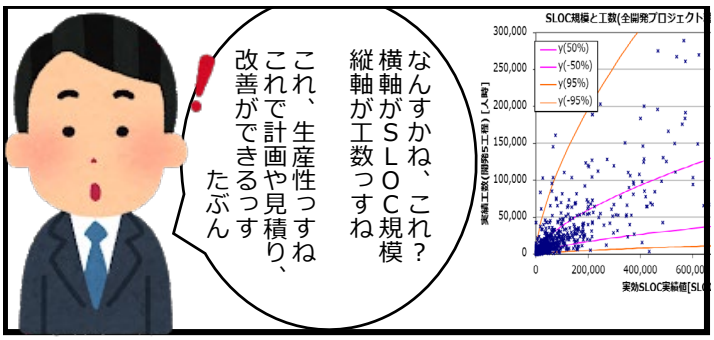
しかしそれでも何らかの分析はでき、その結果を基にソフトウェア開発に役立てることができます。これこそが定量データによる分析の役目です。

ここではまず、生産性の分散は大きいということと、それでも生産性の分析を利用することができるということを感じてください。

生産性は性癖が出る？







【解説】信頼性（不具合密度）

リリース後の不具合数/KSLOC（不具合密度）は信頼性の評価で非常に気になる数字です。これが信頼性評価の中心となる数字です。

バグゼロはリリース後の不具合数がゼロという意味です。さらに言えば、未発見のバグでさえ、ないということを期待している夢の言葉です。

しかし右のマンガの四分位表にもあるように、数は少ないですがリリース後にもバグはあります。バグゼロではありません。

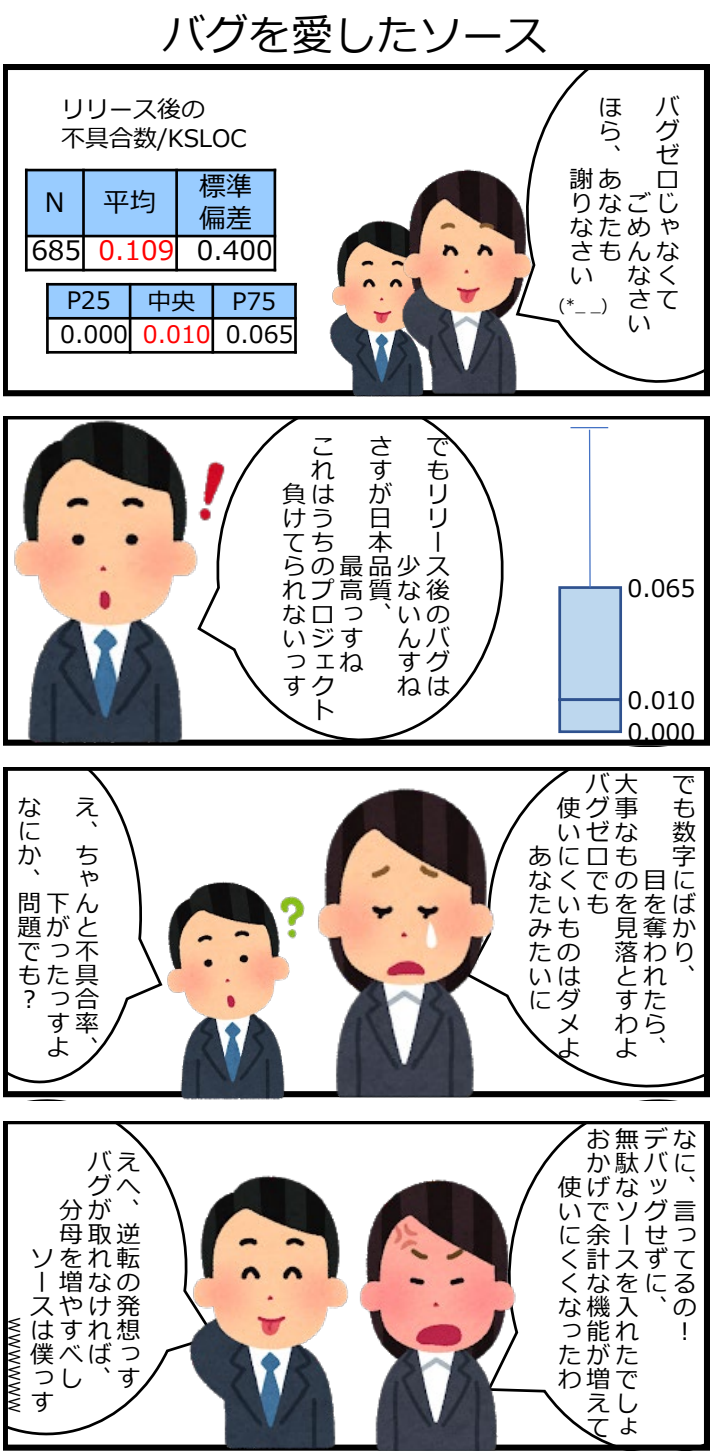
これを厳然たる事実として踏まえて、ソフトウェア開発と、そのソフトウェアの利活用をするようにします。

マンガのように余計なソースコードを追加して不具合率を下げるのは論外ですが、数字に捉われ過ぎるとこの悲喜劇が演じられるようになるかも知れません。

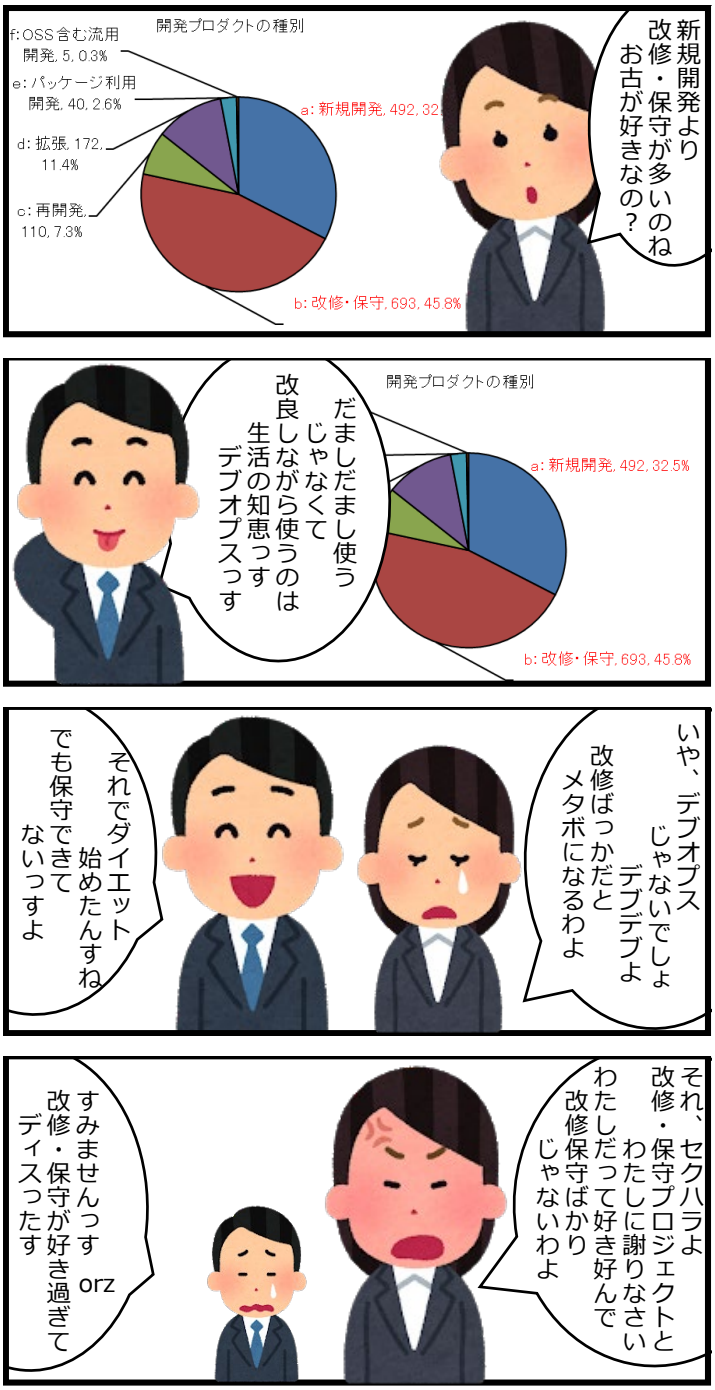
不具合率ばかりに目が行ってしまって、使いにくいシステムになってしまっては、本末転倒です。

使いやすさは不具合率よりも重要な指標です。

定量的データの分析は大事ですが、それればかりに捉われず、使いやすさなど定量評価が難しいものも含めて、総合的に評価していくことが大事です。



改修・保守が好き過ぎる



【解説】開発プロダクトの種別

ソフトウェア開発は新規開発のものより、改修や保守、再開発、拡張するものが多くなっています。逆に新規開発はそれほど多くありません。

マンガにあるように改修・保守ばかりをしていると、ソフトウェアプロダクトは肥大化する傾向にあります。それこそリファクタリングでダイエットする必要があります。しかし改修が多いのが現実です。

データ分析をするときも、対象が新規開発ばかりでは、現実的な分析とは言えません。しかし改修・保守では、既存のプロダクトが変動要因となり、新規開発よりも要因数が大きくなります。その結果、分析は難しくなり、分析結果も有意なものが少なくなります。

最近は新規開発でも、OSSなどの利用などでノーコード開発やローコード開発のように既存コードが増えてきて、分析が難しくなってきています。

これからソフトウェア開発のデータ分析はこれらを踏まえて、新規開発も改修・保守も分析していくことになります。険しい道ですが、一緒に苦労していきましょう。

規模はアンバランスでアンビバレント

【解説】ソフトウェア規模

ソフトウェアプロダクトのソースプログラムの行数（SLOC、Source Lines Of Code）はマンガにあるように、大幅に違っています。また正規分布のようにきれいな分布ではありません。小規模が多い中、逆に巨大なものもあります。

この範囲も実に数行から何百万行までに渡ります。このように規模には大きな差があるので、規模によって開発方法もプロダクトも違うものになります。

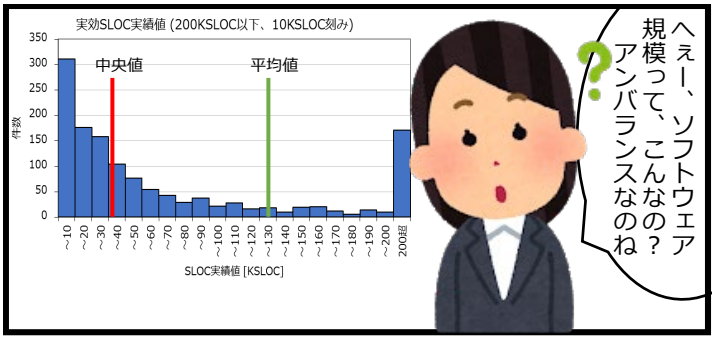
まずはこの散らばり具合を知って、さらに相場観として、中央値と平均値を知っておくことは大事です。

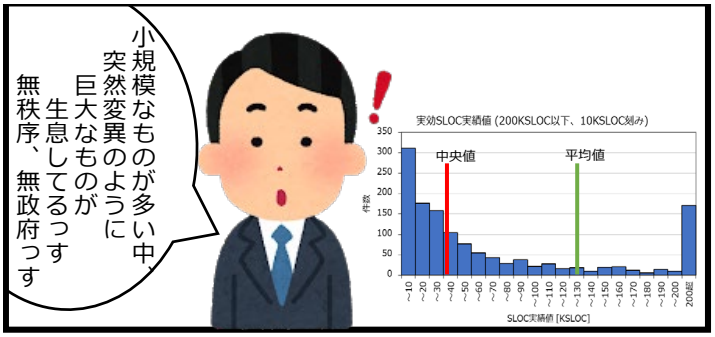
そして、この規模の差を受け入れて、データ分析をする必要があります。

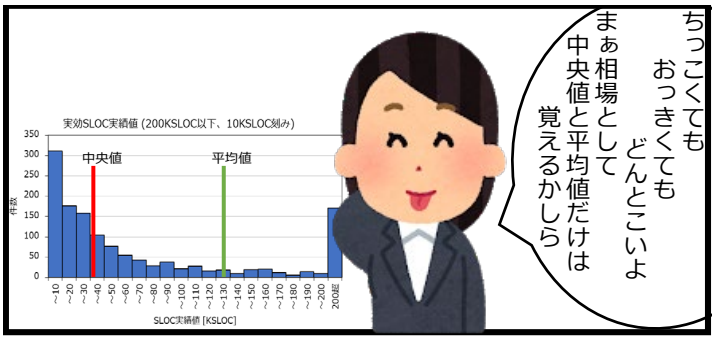
一般的に小規模のものは分散が大きく、逆に大規模のものは分散が小さくなる傾向があります。

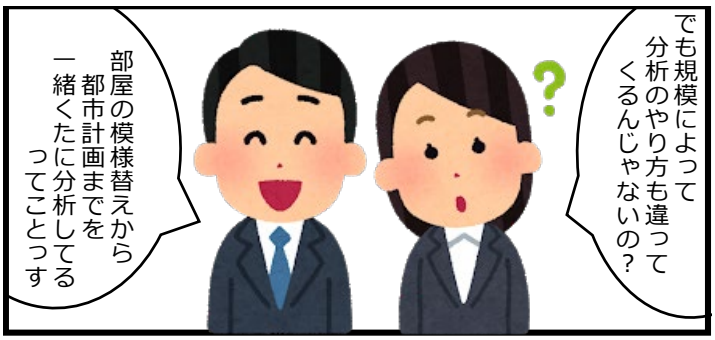
このため、小規模のものはデータでベンチマークをして、一喜一憂しなくてもいいでしょう。

逆に小規模より大きいものはデータ分析をしてベンチマークすることは意味があります。それに大規模なものはリスクも大規模になるのでデータ分析をしてリスクを分析することは大事です。









開発期間は短くて長くて短い

【解説】開発期間（工期）

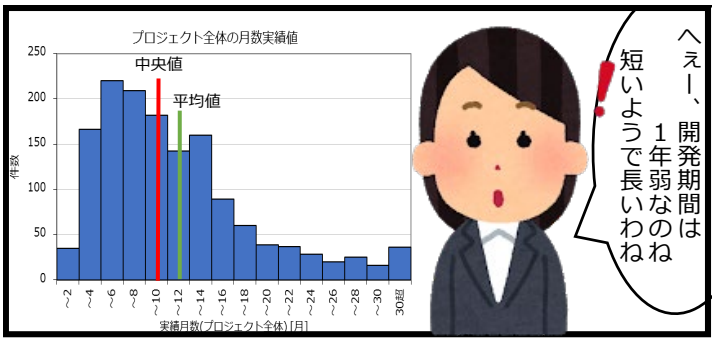
開発期間（工期）の分布は右のマンガのようになっています。中央値で9.1か月、平均値で10.8か月です。２か月で区切ったときの最頻値は４か月から６か月になり、その中間は５か月です。

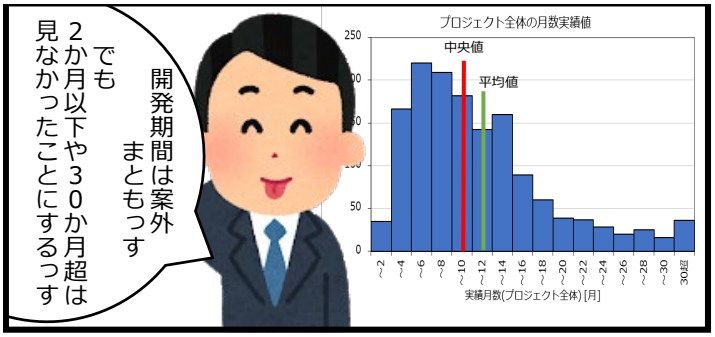
この数字を工期の相場観として覚えておいてください。プロジェクトを計画し見積もるときの参考になるでしょう。

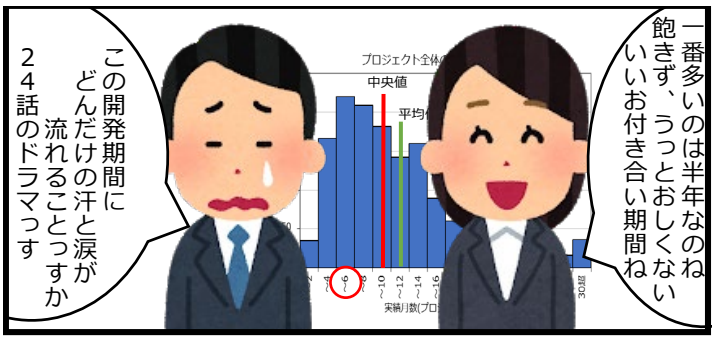
ソフトウェア規模の分布と比較すれば、この工期の分散は小さく落ち着いた分布に見えますが、それでも分散は大きくアンバランスな分布になっています。

このデータは多くはウォーターフォール型開発のものになっています。アジャイル開発であれば、違う結果になるでしょう。たぶん短くなるでしょう。

工期と工数、規模には相関があります。この相関については別のところで紹介する予定です。







・おれたたエンド
「俺たちの戦いはこれからだ！」という最終回の終わり方で、話が永遠に続くこと

【解説】ウォーターフォール型開発

ソフトウェア開発のデータ分析になると、どうしてもウォーターフォール型開発が多く集まります。ウォーターフォール型開発は実に97.8%にもなっています。

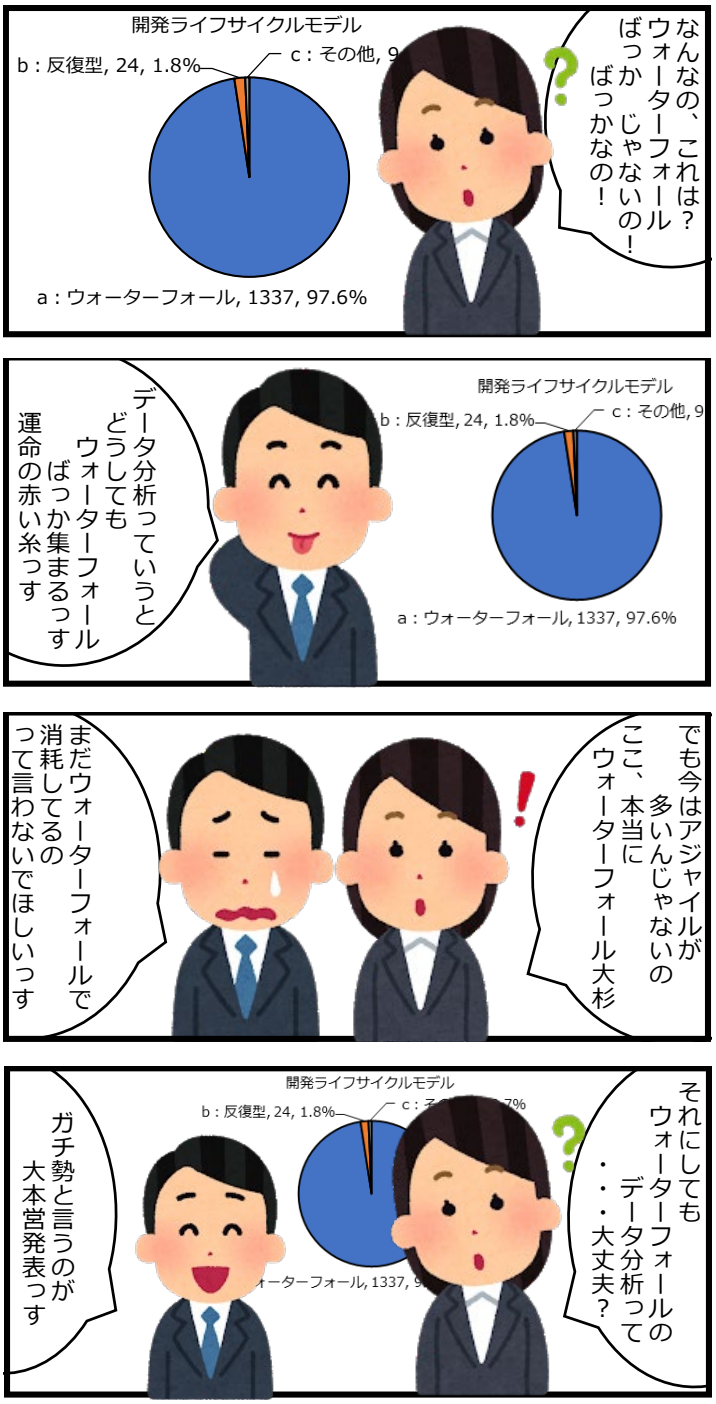
これは日本の開発方法の分布ではなく、データを提供してもらっている組織のプロファイルの統計結果になります。

またウォーターフォール型開発であっても、データ分析を全力で取り組んでいる（ガチ勢）組織もありますが、表面的な分析で終わっていることも多いようです。

逆にまだアジャイルの定量データ分析が慣れていない組織が多いようです。特に生産性に関しては、どのようにアジャイル開発の定量データ分析をするのかで悩んでいる組織も多く、アジャイル開発の定量データ分析は課題の一つになっています。

一方、信頼性に関するデータ分析は開発方法論に依らず、重要な指標となっています。ウォーターフォールでもアジャイルでも他の開発プロセスでも信頼性をおろそかにすることはできず、そのためにデータ分析も重要になってきます。

ウォーターフォールってつおい？



・大杉　多すぎ
・ガチ勢　ガチに（まじめに）勢いがあること

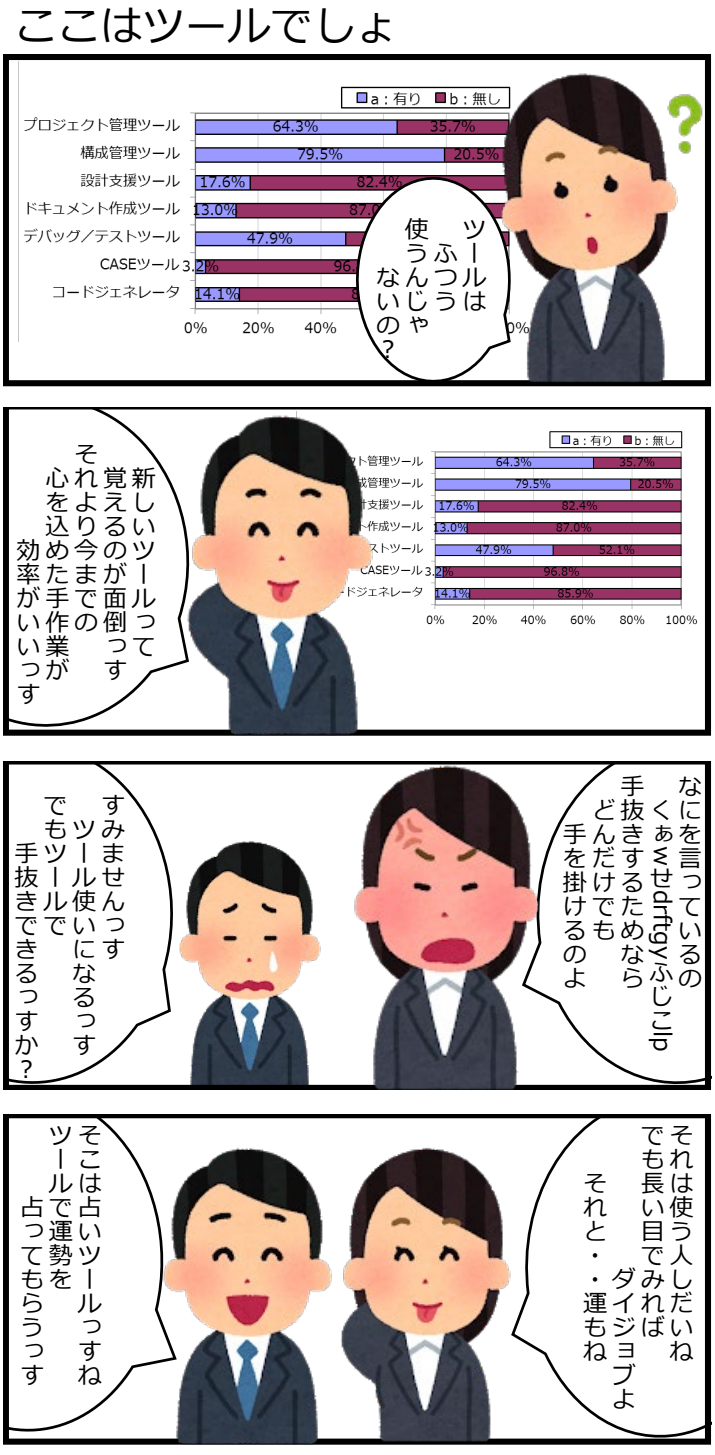
【解説】開発ツール

プロジェクトで使われているツールのプロファイルがマンガに掲載されています。グラフのようにプロジェクト管理ツールや構成管理ツールの管理ツール、デバッグ／テストツールの下流の支援ツールは多くのプロジェクトで使われています。

ツールを使えば、生産性が向上するのか、信頼性が向上するのかは難しい問題です。これは使用経験やプロジェクト、プロダクトに依存しますので、データからは相関関係が明確には示せません。

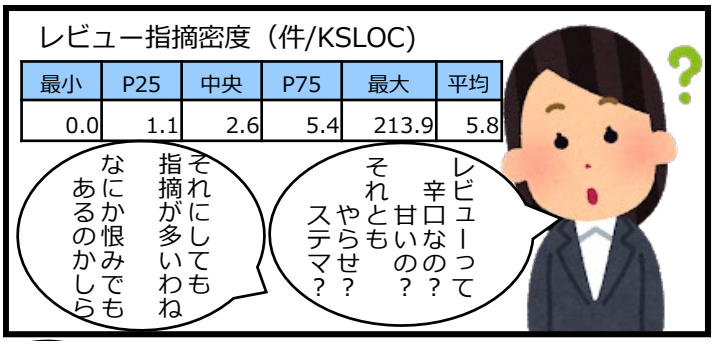
例えば、定量データ分析からはデバッグ／テストツールは信頼性の向上には寄与しますが、生産性は落ちるという分析結果もあります。

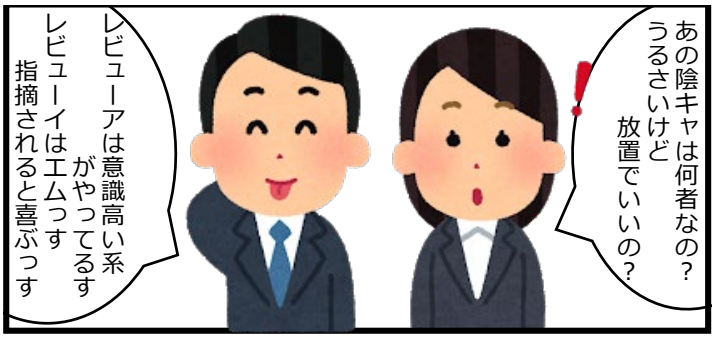
しかし長い目で見れば、ツールを使いこなすぐらいに使えば、ツールによる恩恵が得られるようになると思います。効率を向上させるためには一時的に効率が落ちることも覚悟して使っていくのがいいでしょう。

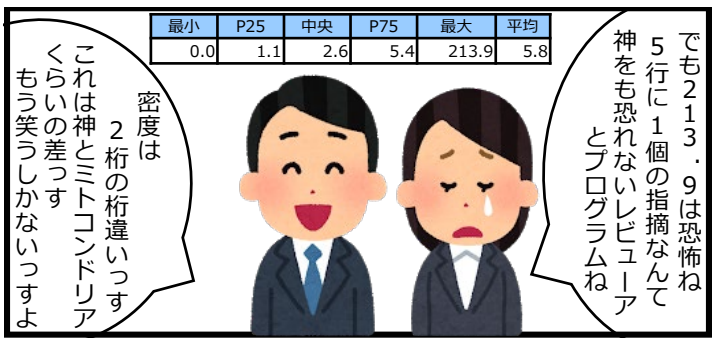


・くぁwせdrftgyふじこlp
何を言っているのかわからないさま。qawse・・・とキーを打つと出る

辛口レビューは正義







【解説】レビュー

マンガのレビュー指摘密度は1000行あたりのプログラム規模のときの、基本設計書でのレビュー指摘件数です。1000行のプログラムを開発するときに基本設計書のレビューのときに、中央値で5.4件の指摘があったことになります。

レビューは早期にプロダクトの欠陥を発見するのに役立つ有用なものです。データ分析としてもレビュー指摘密度を対象にするのは意味があります。

しかしレビューはプロダクト品質と、それからレビューアのスキルや経験、知識などの品質があり、変数が多くなります。テスト検出バグ密度はテスト品質がレビューよりも一定なので、ほぼプロダクト品質（プログラムの品質）に依存しますが、レビューでは上記のようにプロダクトだけでなくレビューアの品質にも依存します。

このようにレビュー指摘密度は分析しにくいデータのひとつになりますが、最初にも書いたように早期に欠陥を見つけることは重要なので、このデータ分析も重要になります。

・陰キャ　陰気なキャラクタ

工期と工数は3乗根

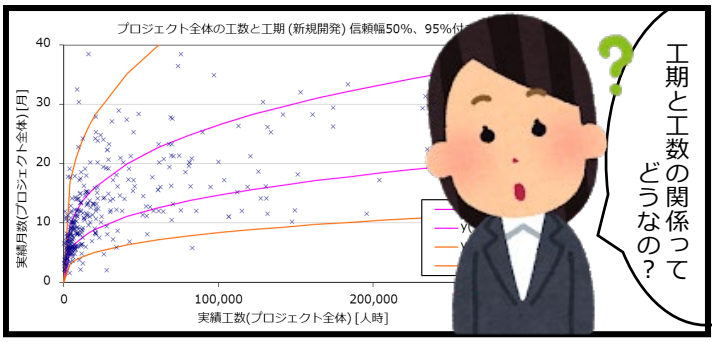
【解説】工期と工数の関係

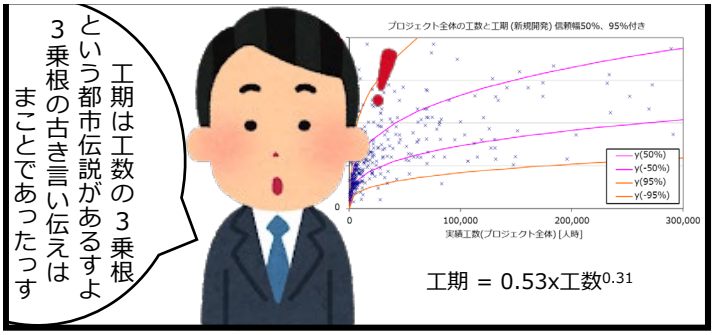
工期と工数の関係は古くCOCOMOで3乗根の関係があると言われていました。そして実際のデータが示すように3乗根に近い分布になっています。

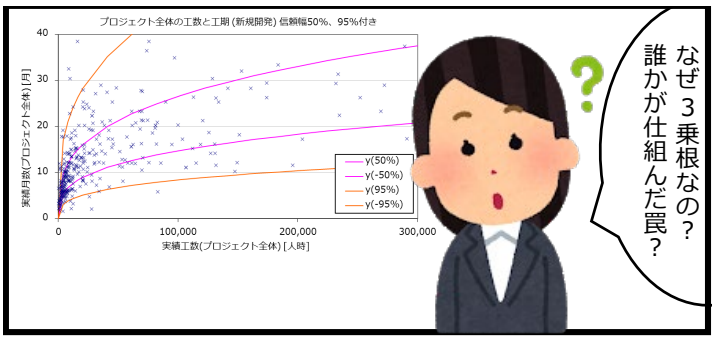
この3乗根の関係が工期と工数の本質的な関係なのか、それともマンガにあるように3乗根という知識を持って工数と工期を計画し実施するから、結果的に3乗根のデータになったのかは、わかりません。

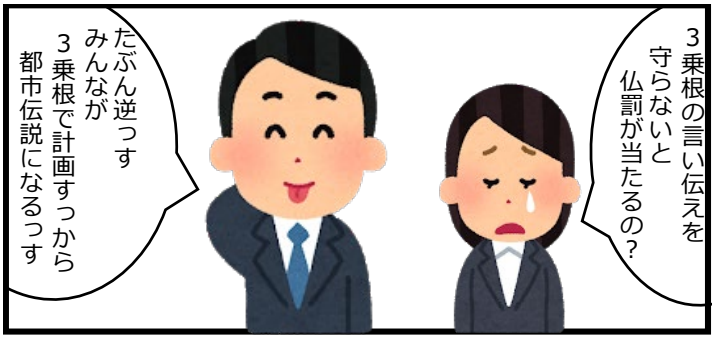
ソフトウェア開発は変動要因が多いので、3乗根の近似式をそのまま使うことは危険です。リスクは見込まなくてはいけません。しかしながら、この3乗根という知識を持って、ソフトウェア開発に当たるのは成功への武器になります。

今回の工期と工数の関係だけでなく、色々な関係があると言われています。実際のデータを収集し、その関係を確認します。そして妥当な関係であったものは、ソフトウェア開発で使うようにしていきましょう。

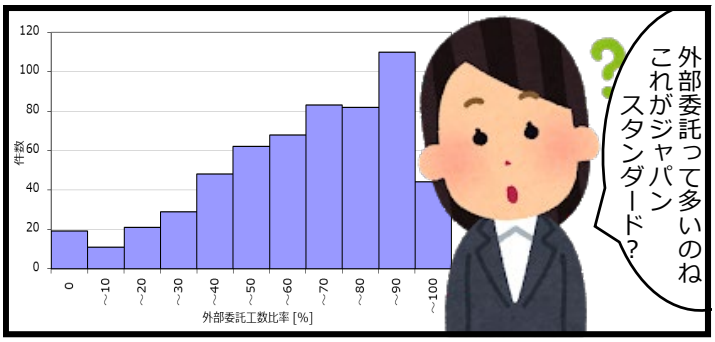


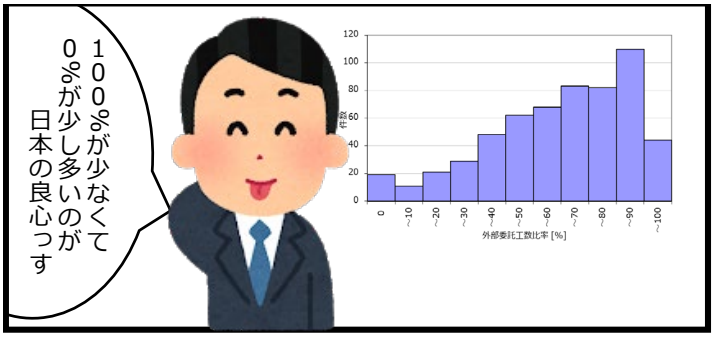


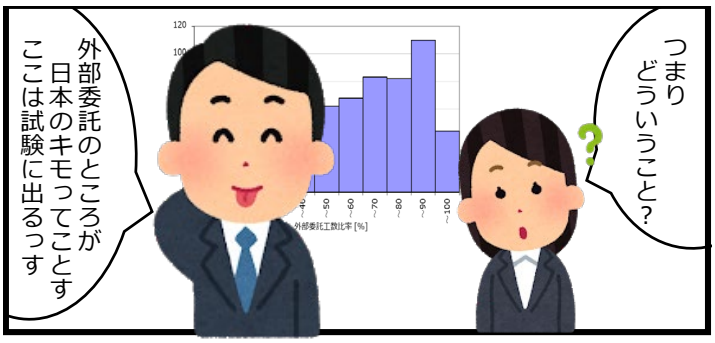




果てしなき外部委託の果てに







【解説】外部委託率

ソフトウェア開発における外部委託率の分布はマンガのように、高い傾向にあります。

最頻値は80%～90%の外部委託率にです。一方、90%～100%の外部委託率は少なくなっていて、完全に外部に委託しているプロジェクトは少なくなっています。

外部委託をしていると開発データが外部委託先を通して入手することになります。このようにデータを間接的に入手することになり、データの即時性や正確性が落ち、データ分析に支障を与える可能性があります。またデータ分析によるリスク分析においても同様の問題（即時性、正確性）が発生し、対策が遅れる可能性があります。

外部委託が必要なプロジェクトではデータ分析が疎かにならないように外部委託先とデータの取り扱いについて定め、共有し、分析するようにします。

委託先とのデータ連携のためにも、コミュニケーションがキーになります。何のためにデータを収集し、どのように分析して、どのように活用するかを決めるためにもコミュニケーションが重要になります。

特別編：データは欲しいけど

マンガでわかる
ソフトウェア開発
データ分析

FAQ 編

ソフトウェア開発分析

繰り返し尋ねられる質問

【質問】ソフト開発でのデータ分析ソフトウェア開発での定量データ分析は役に立ちますか。

ソフトウェア開発に携わる人は色々なレベルの人がいます。一般に言われていることですが、ソフト開発でのレベル差は1桁以上の開きがあるとも言われています。つまり10倍以上の生産性や10倍以上のバグを出すことさえ、あるかも知れません。確かにこのようなソフト開発で定量データによる分析は有効ではないと言う人もいます。

このように思うのは、数人でプロジェクトを運営するとき、これが今までのデータ分析で出てきた数値と同じように扱っていいのかという心配があるからです。そしてその心配はもっともなものです。

ソフト開発に限らずに、仕事のやり方を臨機応変に柔軟にしなければいけないとき、またその対応スキルの差が大きい人が関与しているときのデータ分析では、多数を対象にした「統計」的な分析を使うことになります。

逆にこの統計的な方法では個々人や少人数のプロジェクトを対象にした分析は難しいものになります。統計で有効なのは一定の人数がいる場合になります。

しかし個々人でも少人数のプロジェクトでも、そのプロジェクトの件数が多くなれば、統計で対応可能な一定の数になり、定量データによる分析は有効になります。

つまり勇者レベルから村人Aレベルの数多（あまた）のレベルの人がいても、累算での人数が多くなれば、統計的に分析できるようになります。結論としては、ソフト開発においても人が多くなれば、定量データの分析は有効になります。

データ分析って役に立つの？

・勇者から村人Ａ
冒険ファンタジーの世界では、最強の勇者から最弱の村人Ａがいる
・見習い冒険者
学校を出てソフトウェア開発の世界に入りたての若者レベルのこと
・ひのきの棒
見習い冒険者が最初に装備する武器、ソフトの世界ではメモ帳のこと
・異世界
冒険ファンタジーの世界で魔法と剣がある世界
ソフト開発のデスマ世界

【質問】不具合修正の工程
結合テストや総合テストで見つけた不具合を修正するのは、どの工程でするのですか。試験工程ですか、それとも製造工程や設計工程に戻って作業をするのですか。

テストで見つかった不具合（バグ）を直すのは、基本的にその試験工程で行います。

試験工程では(1) テストの作成・実施、(2) テストで検出した不具合の原因追及、(3) 不具合の修正という作業を行います。不具合の原因追及をして、その修正方法を検討した結果、仕様変更や大幅な修正が必要になる場合は、製造工程や設計工程、さらに要求定義の工程にまで戻って、作業をする場合があります。

ウォーターフォール型開発では通常は後戻りしない開発ですが、このような場合は少ないですが発生することがあります。逆に考えれば、マンガにあるように戦略的撤退もプランＢとして想定すべきかも知れません。

バグはいつ直すの？

・3密
新型コロナウイルスが流行っていた当時、その流行を防ぐために避けるとされた密閉、密集、密接の3個の密のこと
・プランＢ、戦略的撤退
単純に「逃げる」というのではなく、このように言うと、事前に決めていた作戦のように聞こえる

【質問】工数の工程比率
工期や工数の工程比率はどのようになっているでしょうか。また新規開発と改良開発で異なるものでしょうか。

工期や工数の工程比率は各所で統計の分析結果があります。このソフトウェア開発分析データ集では新規開発の工期の工程比率はp.161の表A3-3-5に掲載されていて、概略は設計：製造：試験＝35：25：40＝７：５：８（なごや）です。

新規開発の工数の工程比率はp.167の表3-3-8にあり、概略は設計：製造：試験＝1：1：1です。これはCOCOMO IIで言われていた３：４：３と比較して製造工数の比率が小さくなっています。

また改良開発の工数の工程比率はp.169の表A3-3-9にあり、概略は設計：製造：試験＝3：3：4です。新規開発と比較して試験工数の比率が大きくなっています。これは改良開発では回帰テストなどの試験に多くのコストを掛けるからです。

工程比率は清く正しく美しく

・黄金比
デザインを美しく見せる比、建造物や美術品、そして顔にある
・続編
新編よりも安定感があるため、どの世界でも多用される
例えば、深夜アニメも続編だらけである

【質問】UI設計の工程
画面インタフェースの設計はどこの工程でするのでしょうか。基本設計の工程でしょうか、それとも要件定義の工程でしょうか。

ここで基本設計とは、SLCPのシステム方式設計とソフトウェア方式設計に相当するものとします。

要件定義と基本設計の差異は成果が要件定義なのか、設計仕様なのかになります。この考えでは画面設計という言葉なので、基本設計の工程になるとするのが自然です。

しかしマンガにもあるように画面に対してはユーザからの仕様変更を含む要求が多く、それも連続的に来ることが多いです。このため、ユーザとの要件定義なのか、それを元にした設計仕様なのかの区別が明確ではありません。

つまり画面設計におけるユーザの関与の度合いにより、それが要件定義なのか基本設計なのかに分かれます。

この意味ではウォーターフォール型開発よりもアジャイル開発の方が画面設計は向いていると言えるでしょう。

どのような開発方法を取るにせよ、画面設計はユーザとの対話が重要なのは間違いはありません。

画面は誰のもの？

・いちゃもん
文句を言う側は正当だと思っているが無茶な要求を含むもの
・死亡フラグ
戦いの前にこのセリフを言うと、その人物はこの戦いで死ぬ

【質問】単体テストのデータ
単体テストのデータは収集して分析するのでしょうか。した方がいいのでしょうか。

単体テスト（コンポーネントテスト、ユニットテスト）は他のテストとは異なり、プログラマ自身がインシデントレポートを書くことなく、テストを実施することが多くなっています。

これは単体テストがプログラミングと不可分な作業として行い、バグが見つかれば、そのままデバッグする作業までが一連の流れとして実施されることから、上記のようなテスト方法になります。

そしてテスト結果もテストを実施したテストケース数で管理するのではなく、コードカバレッジ率などの別の観点で管理するようになっています。

このようにホワイトボックステストである単体テストではテストケース数よりも意味のあるコードカバレッジなどでテストが管理できます。一方、ブラックボックステストではどうしてもテストケース数やバグ曲線などで管理するしかありません。

このため、単体テストではテストケース数はあまりデータ収集されず、それよりもカバレッジなどの指標が重要視され、カバレッジ率だけで管理されるようになっています。

そして単体テストの検出バグ数も上司に報告することなく、ばれずにすませることもできます。

単体テストはひとりでこっそりと

・オールグリーン
プログラムコードを単体テストで実施すると緑色になるテストツールが多く、オールグリーンになるとカバレッジが100%になったことになる
・緑一色（りゅーいーそー）
麻雀で役満貫（役満）となる役
・ハコテン
麻雀で持ち点が0点以下になること

【質問】組込みソフト開発でのレビュー
組込みソフトウェア開発ではエンタープライズ系などのソフトウェア開発と比較して、レビュー指摘率などのデータ収集が少ないのはなぜですか。

エンタープライズ系などの大規模なソフトウェア開発では各種の開発基準を設け、また開発フレームワークも使い、比較的定型的に開発されることが多いですが、それに対し組込みソフトウェア開発では機器に直結して開発するため、定型的な開発よりも独自の開発形態になることが多くなります。この結果、組込み系の開発では職人文化やその中の徒弟制度の独特の文化があります。

このため、設計レビューなども定型的に実施されることが少なく、職人文化の元に暗黙的に実施される場合が比較的多いようです。

これが組込みソフトウェア開発でレビュー指摘率などの上流工程での定量的なデータ分析が定着していない理由のひとつになっているかと思います。

もちろん組込み系でもテスト検出バグ率などの下流工程での定量的なデータ分析は実施されています。

組込みはレビューがお嫌い？

・小一時間（こいちじかん）
小言を言うときの標準的な時間
・我々の業界ではご褒美
褒美に見えない行いも特定の人からはご褒美に見えることもある
・全集中
身体を活性するための呼吸法

【質問】バグ件数はバグの原因数で測定した方がいいのでしょうか、それともテストの結果で測定できるバグの現象数でいいのでしょうか。

バグの原因を見つけて、その原因数で測定する方が、バグを入れ込んでしまう時点に注目できるので、望ましいです。

しかしバグの原因を見つけるのは面倒で、多くの時間を必要とします。

一方、バグの現象数はテストケースを実行した結果、直ちにわかりますので、測定コストはテスト実施時に低コストでできます。

しかし1個のバグの原因から複数のバグの現象が見つかることもあり、一般的に現象数では原因数よりも多くなります。またこれからバグの原因の１個をデバッグすると複数のバグの現象がなくなることがあります。

これからバグのメトリクスをバグの現象数だけにするのか、現象数に加え、原因数も対象にするかのトレードオフをする必要があります。

例えば、重要なプロダクトのときは原因まで追究し、そうでないときは現象数だけを測定する方法があります。

バグによる怪奇現象の正体は？

・怪奇現象
昼間は動いているのに真夜中になると突然エラーを吐き出すポルターガイスト現象、数えると一つ足らない番町皿屋敷現象、ログを入れると怪奇現象がなくなるハイゼンバグ現象などが見つかっている
・運命の赤い糸
2者の相関関係が強いものを結ぶ赤い糸のこと、ただし実は裏事情がある疑似相関のときも赤い糸で結ばれるので注意が必要

【質問】プログラミング言語により、生産性が大きく異なると思うのですが、どうでしょうか。

確かにプログラミング言語により、生産性（工数あたりのSLOC/FP数）や信頼性(プログラム規模あたりのバグ数)に差があります。これはプログラミング言語だけの違いによるものではなく、その言語の開発環境にも依存しています。

ソフトウェア開発データ白書2018のp.272ではCOBOLの4.88SLOC/人時からVBの7.24SLOC/人時の差があることがわかります。

また言語による差よりも新規開発か改良開発かなどの他の要因による生産性や信頼性の差も大きいので、言語の違いばかりに目を奪われないようにしてください。

お値打ちなプログラミング言語は？

・それでも男ですか、軟弱者！
セイラ・マスが言った励ましのお言葉。
・働いたら負け
自宅警備員（つまりニート）が決めゼリフにしているお言葉。

【質問】生産性を分析データ集のグラフと比べて、生産性に問題がないかどうかの判断をしてみたいのですが、どうすればいいのでしょうか。

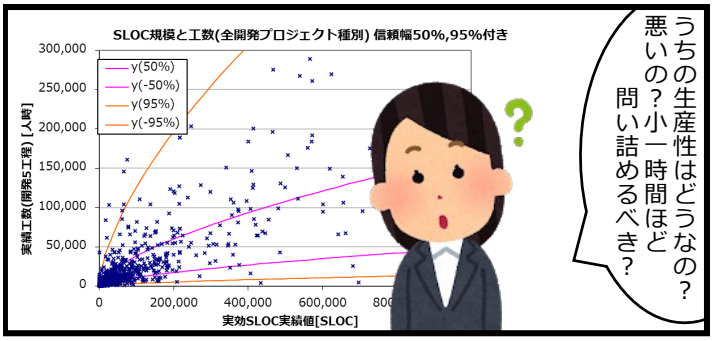
右のマンガにあるような生産性のグラフから、対象のプロジェクトと比べることで生産性の評価ができます。

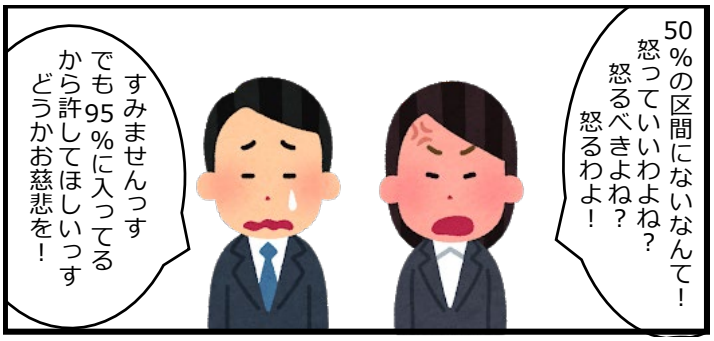
このグラフは生産性の予測値の信頼区間（予測区間）になります。つまり、新たに取得した生産性の値がどの信頼区間の中にあるかがわかります。

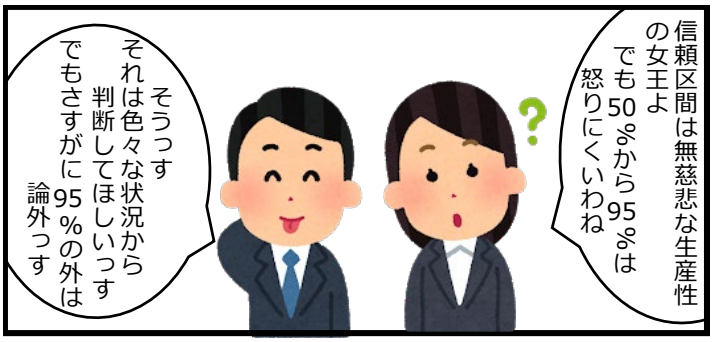
グラフには50%と95%の信頼区間が描画されていますので、新しく測定した生産性の値がどの区間にあるかがわかります。

50%の信頼区間にあれば、問題はないでしょう。95%の信頼区間にないときは、その原因を探りましょう。担当者にヒアリングするのも一つの方法です。50%から95%の間にあるときにどうするかはプロジェクトで決めることになります。重要なプロダクトであれば、原因を見た方がいいかも知れません。

生産性はこれでいいの？







・信頼区間は無慈悲な生産性の女王
ハインラインの「月は無慈悲な夜の女王」から
・言い訳を聞こうか
各地で発せられる脅しの言葉。この言葉の次は「言い訳はするな」となる。

テストケースの粒度は揃うの？

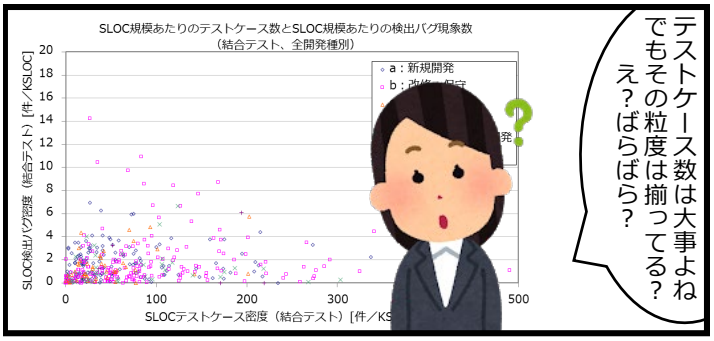
【質問】テストケースの粒度はどの程度にすればいいのでしょうか。その記述はどれくらい具体的に書けばいいのでしょうか。

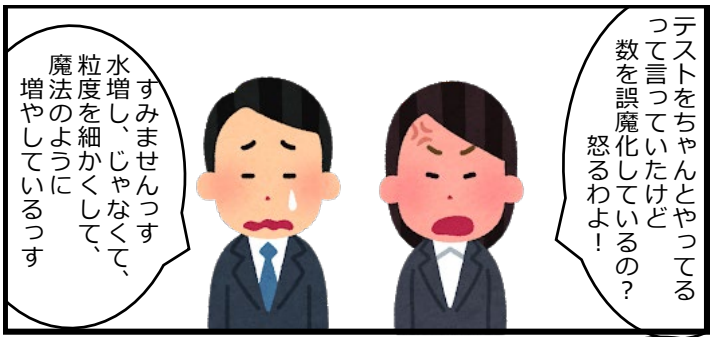
テストケース数やその密度は信頼性を計測するときの重要なメトリクスです。

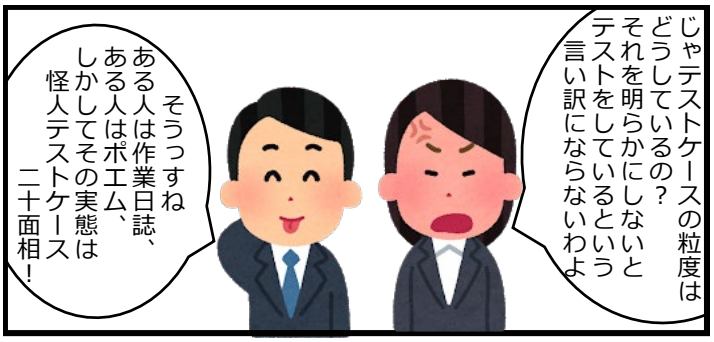
設計はすべてを記述するために抽象的になりますが、テストケースは逆に具象的に記載して、設計の具体的なユースケースになることが多いでしょう。これはテストの正確性を確保するためにも必要なことになります。

しかしテストケースをいくら具象的に記載すると言っても、その具象度の違いによる粒度の差は出てきます。

またテストの段階によっても粒度は違うでしょう。システムテストと単体テストを比較すれば、システムテストは抽象的で、単体テストは具象的になり、粒度は異なります。そこで各段階のテストでテストケースの粒度を揃える必要があります。







・魔法
反同値分割法の魔法を使えば、テストケースは増える。
・ポエム
「こう動けばいいな」のような希望が散りばめられているテストケース。
・テストケース二十面相
一つのテストケースを二十のテストケースにできる怪人。

【質問】ソフトウェアの価値はどのように評価すればいいのでしょうか。どのように定量的に計測すればいいのでしょうか。たとえばソフトの価値をお金に換算するのはどうすればいいのでしょうか。

ソフトウェア製品の価値は評価しにくく、また定量的な計測はさらに難しいものです。
ソフトウェアの価値のひとつに製品品質があり、たとえばソフトウェア品質標準SQuaREには、機能適切性やユーザインタフェイス快美性などの多数の項目がありますが、そのどれもが定量的に計測するには難しいものばかりです。
開発時は人月のコストから見積もることが多く、逆にソフトウェアの価値から見積もることは少ないでしょう。

ソフトウェアの価値は、そのソフトウェアを含むシステムを使うユーザの満足度に大きく依存するものです。そこでユーザの満足度はアンケートやヒアリングで計測するという方法もあります。
一方、バグが少ないソフトウェアは、それだけでなく、操作感などの使いやすさや実行効率などもバグの少なさに相関して良いものが多いと考え、バグ密度をソフトウェアの価値の代替に使うこともあります。もちろんバグが少ないソフトウェアでも使いにくくユーザの満足度は低く価値が小さいものはあります。逆にバグが多くて使いやすく満足度の高いソフトウェアもあるでしょう。それでもバグが少ないソフトウェアは一般的には価値があると考えることもできるでしょう。

上記の２つの考えを紹介しましたが、やはりソフトウェアの価値は評価も定量的計測も、お金に換算するのも難しいものですが、悩みながら、より良いものを見つけるようにしましょう。

ソフトの価値って？

・人月の神話
ブルックス先生執筆の人と月は交換できないという神話。
・マッチングアプリ
相性ぴったりで満足度が向上する相手とマッチングできるとされているアプリ。

(参考)

マンガでわかる アジャイル メトリクス

FAQ 編

アジャイルとそのメトリクス

繰り返し尋ねられる質問

アジャイルメトリクスについてよく寄せられる質問を記載していますが、その回答に対して、アジャイル開発に関わる方々からのご意見をお願いします

はじめに「アジャイルメトリクスのFAQ」

【質問】メトリクスとはどういうものでしょうか。測定データとは違うのでしょうか。

メトリクスとは辞書で調べると、メートル法や韻律法という意味です。

ここでのメトリクスとは、測定基準や測定尺度、測定評価という意味です。

つまり測定した値そのもののデータではなく、その測定基準や尺度のことです。

しかし測定現場では測定データをメトリクスと呼んでいることもあるので文脈で判断する必要があります。

測定をするときにはメトリクスに従って実施します。しかし多くの場合、メトリクスが明示されず暗黙的に決められていることがあります。このときは暗黙的なメトリクスを意識して測定する必要があります。

メトリクスの例としては、健康診断のときは身長や体重、視力、血圧などを測定項目として選び、個々の測定値に対して基準を設け、また複合した測定データの値に基準や尺度を設け、これらの測定値から評価を行います。健康診断ではメトリクスで決めた閾値となる基準値を越えた場合、再診断となります。

ソフトウェア開発に関するメトリクスでは、コード行数(SLOC, Source Lines Of Code)やバグ件数、工数などがあります。

上記のうち、コード行数やバグ件数は注目を集めやすいメトリクスですが、アジャイル開発では、開発中のコード行数やバグ件数の評価が難しいため、ウォーターフォール開発と比較して、より注意する必要があります。

メトリクスってなんなの？

・ヲタク
嗜好性の強い趣味に没入するもの。アニヲタ、鉄ヲタ、ゲーヲタなどが生息している。
・しめる
首を絞めることで、殺すと同意。

【質問】アジャイルソフトウェア開発でメトリクスや定量データ管理はどうでしょうか。コストに見合うくらいに役に立つでしょうか。

アジャイルだけでなく、人手で開発するソフトウェア開発では定量データ管理やそのメトリクスの有効性に疑問を持っている人は多くいるでしょう。そしてこの疑問はもっともなものです。

アジャイル開発では対象が小規模であることが多く、また複数回の繰り返し開発（スクラムではスプリント）になるため、1回の開発は小さくなります。このため、統計的な分析が有効でない場合が多いです。

しかしアジャイル開発でも、特に大規模なものを対象にするときには、リスク対応や評価のためにメトリクスは必要になり、メトリクスは有用な羅針盤になるでしょう。リモート開発のときも同様に必要になるでしょう。

またアジャイル開発では生産性に関する捉え方が難しく、そのメトリクスも困難なものになります。一方、リリース後のバグ密度などの信頼性に関するメトリクスは開発方法に依存せずにソフトウェア製品とその開発プロセスのいい羅針盤になるでしょう。

しかし開発中のバグはその定義や件数の評価が難しいため、ウォーターフォール開発と比較して、より注意する必要があります。

このようにアジャイルメトリクスには色々と困難な面と有効な面があります。ここではこのアジャイルメトリクスの困難さを乗り越え、有効に使う方法を一緒に考えていきたいと思います。

アジャイルでメトリクスどうなの？

・逃げ恥
ドラマ「逃げるは恥だが役に立つ」から、ここではウォーターフォールからの戦略的撤退組や転向者の別名。
・お布施
有料で物品購入をすること。ここでは布教に物理的に尽力すること。
・聖地巡礼
作品に関係のある地や店を訪れて感動を新たにすること。ここではアジャイル信仰者が訪れる聖地を指す。

【質問】アジャイルソフトウェア開発で生産性はどのように測定するのでしょうか。そもそもアジャイルでの生産性とはどういうものでしょうか。

アジャイル開発では仕様変更を通常のこととして受け入れるため、同じコード部分を複数回変更することがあり、結果としてのソフトウェアのコード行数(SLOC、Source Lines Of Code)や機能量（ファンクションポイント:FP）のみで生産性を測定するのはあまり意味がありません。

そこでストーリーポイント(SP)のベロシティを測定して、これで同一チームや類似のプロジェクトの相対的な生産性とすることが多くなっています。つまり以前のものや類似のものと比較していいか悪いかだけの相対評価になります。

このため、ソフトウェア製品として他社や他チームとの絶対的評価（ベンチマーキング）や比較には使えません。マンガのようにボーナス査定にも使えません。

この生産性のベンチマークよりも、SPのベロシティを未完了SPとともに、開発が今までと比べて遅れていないか、何か遅延の原因があるのではないかなどのリスク対応に使うことが有効になります。

しかし一般のソフトウェアの品質保証では、過去の実績値と比較しての十分性を元に評価することが多く、上記の相対値では、この手法は使えないことに注意してください。

アジャイルの生産性ってなんなの？

・コレジャナイロボ
クリスマスプレゼントで欲しかったのは、これじゃない！と言われるロボット
・SP
ストーリーポイント。ユーザの要求（ユーザバックログ、ユーザストーリー）を見積るための架空の単位。チームで大きさが異なる。

【質問】アジャイルソフトウェア開発で品質やその中の信頼性はどのように測定するのでしょうか。

製品品質の一部である信頼性は開発方法に依存せずに同じで、製品のリリース後のバグ数で測定します。

一方、開発中の信頼性は開発中に出たバグ数や実施したテストケース数で測定します。これも基本的に開発方法に依存せずに同様なものになります。

信頼性のメトリクスでの分母に相当する工数は開発方法に依存しませんが、プログラム規模（コード行数やファクションポイント）は仕様変更が当然のように入るアジャイル開発では測定が難しいものになります。

但しアジャイル開発では複数回（スクラムではスプリント）にわけて開発するので、今回の開発では対応せずに次回で対応することもあり、バグの測定対象もそれに合わせる必要があります。つまり対象が完了(Done)するまでのバグについては計測対象外にするなどの注意すが必要です。

また複数回の開発のため対象が小さくなるので、メトリクスによる毎回の定量データ分析はそれほど有効ではありません。

なお、ここでは触れませんが、信頼性以外の使用性や保守性などの品質測定も重要で、特にアジャイル開発ではこれらの品質特性も注目されます。

アジャイルの品質はどう測るの？

・いちゃもん
文句の一種で受け手がうるさいと感じるもの。
・小一時間
説教するときの基本時間。1時間以内とは限らない。

【質問】アジャイルソフトウェア開発でメトリクスによるリスク対応はどうするのですか。それは役に立ちますか。

メトリクスの活用としては、開発終了後にベンチマーキングとして使う場合と、開発中に計測するものがある。

後者の開発中に計測するときは一般的に、メトリクスによる測定データの分析から、例えばバグ密度が通常の値とは大きく異なる異常値が発見された場合は、リスクとして認識し、それをトリガーとしてリスク対応を行います。

アジャイル開発では対象が小規模であることが多く、測定データの分散が大きくなりがちです。このため異常値の判断が難しく、第一種の過誤（犯人でないものを犯人とする誤り、偽陽性）が起きやすくなります。

しかしそれでもこの開発中のメトリクス測定はリスク対応に有効です。第一種の過誤も原因を突きとめていけば、この測定結果も有用な情報となります。

問題はコストですが、メトリクスの測定データの収集を計測ツールで自動化し、分析もなるべく分析ツールで自動化できるようにすれば、コストは低減できます。

マンガでも触れていますが、開発現場にリスク検知のための測定データの収集と分析にコストをかけさせず、逆にリスクを早めに知らせるように、現場ファーストを第一に考えてください。使わないデータはもちろん、将来、使うかも知れないと考えたデータも思い切って測定しないようにします。

そうすれば、アジャイルの開発現場でもメトリクスの有効性が認知されるでしょう。

そこ危なくないの？

・マーフィー
マーフィーの法則で有名なマーフィー。リスクの可能性があるものは必ずリスクになる、それも最悪のタイミングで。
・リスク、ダメ、ゼッタイ
イジメも駄目、絶対。
・はらたま、きよたま
払えたまえ、清めたまえの省略形。サクラの口癖。

【質問】アジャイル開発で測定する対象はどこなのでしょうか。すべての開発対象でしょうか、それとも今回の開発対象だけでいいのでしょうか。

アジャイル開発ではすべての開発対象が明確になっていないことが多く、このため品質や生産性の測定対象は限定する必要があります。

アジャイル開発をするときには、過去の開発対象の改修と、新規の開発が混合します。そして開発完了(Done)もあります。このため、測定の対象となるものは完了部分を除いて改修部分と新規開発分の両方になり、測定は複雑になります。

ウォーターフォール開発であれば、比較的この両者は区別されて開発し、その測定も対象がどちらかに重点を置くことができるので、測定はシンプルにできます。

アジャイル開発では複雑な測定とメトリクスになりがちですが、このような測定もアジャイル開発のように、必要なことだけを必要なときにのみ測定・分析するようにして、無駄な作業を減らし、低コストで素早く実施するようにしてください。

今日はどこを測るの？

・スパゲッティ
スパゲッティプログラムから。スパゲッティプログラムとはスパゲッティのように制御があちらこちらに飛びまくり、ストレート麺でなく、見事な縮れ麺になっているプログラムのこと。
・手抜き
無駄を省くことの別名。省力化とも言う。

【質問】アジャイルソフトウェア開発とウォーターフォール開発でメトリクスはどの程度一緒にできるのでしょうか。それとも別のメトリクスになるのでしょうか。

リリース後のバグやテストケース数などは開発方法に依存しないので同じメトリクスを使用できます。

また開発途中のバグ数やレビュー時間、工数などは調整すれば、同じメトリクスを使うことができます。

例えばアジャイルで次の開発（スクラムであれば次のスプリント）に回すバグなどを除外したり、ペアプロのときの工数を別に測るなどの調整をします。

一方、生産性に関するメトリクスは最終結果の規模としてのプログラム行数やファクションポイントだけでは、仕様変更が当然のものとして入るアジャイル開発ではそのまま適用できないでしょう。

しかしメトリクスをウォーターフォールと共通化できれば、その比較もでき、それぞれの利点と欠点もわかり、またウォーターフォール開発とアジャイル開発のハイブリッドな開発がしやすくなるでしょう。

しかしハイブリッドな開発はメトリクスだけでなく、多くの面で難しい開発になるので注意してください。慣れていないアジャイル開発ではお勧めできない開発になります。

またウォーターフォール開発では、過去の実績値と比較することが多く行われていますが、ハイブリッド開発ではこの手法は難しくなります。

ウォーターフォールと仲良く？

・メトハラ
メトリクスハラスメント。生産性や信頼性の値で嫌がらせすること。
・校舎裏
放課後、不良に呼び出されて〆られる場所。
・伝説の大きな木の下
フラグが立っていれば、卒業式の日に、愛の告白をされる場所。

【質問】アジャイルソフトウェア開発ではテストは多いと思いますがこれはメトリクスに影響は出ないのでしょうか。

アジャイルでは複数回の開発（スクラムではスプリント）で毎回、テストを実施します。その中にはデグレードを発見するためのリグレッションテストも含まれています。この結果、実施するテストは多くなります。逆に考えれば、アジャイルではテストは常時実施することになります。

このためアジャイルでは効率的なテストの実施が求められ、ツールなどを使って低コストな自動テストが求められます。

リグレッションテストは比較的自動化しやすいテストであり、このテストを実施するアジャイルではツールによるテストの自動化が進んでいます。

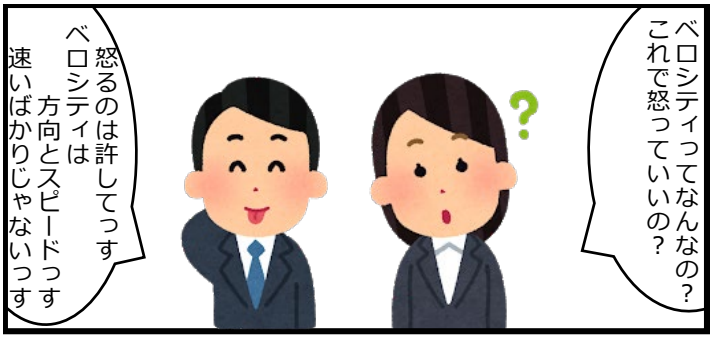
メトリクスとしてテストケース数やテスト工数などを選択するときは、これを考慮して基準値や閾値を決める必要があります。

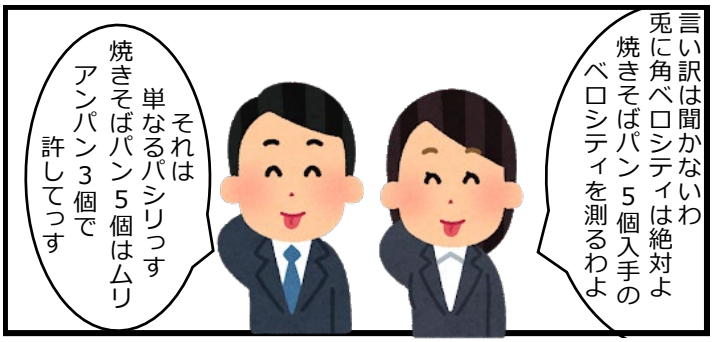
テストケース数の基準としては、例えば「ソフトウェア開発分析データ集2022」のデータでは、ウォーターフォール開発では、コード1000行あたり結合テスト50個、総合テスト15個程度なので、アジャイル開発ではそれ以上になるでしょう。

テストはごはん？

・おかずもごはん
楽曲「ごはんはおかず」にインスパイアされた返し歌。
・意地悪爺さん
ボブ・バトルのマンガの主人公。
・お値打ち
お買い得という意味を持つ名古屋人独特の言い回し。価値があるとは一言も言っていないことに注意。

ベロシティで怒っていい？





・おしおき、折檻
月に代ってするのがおしおきで、火星に代ってするのが折檻、水をかぶってするのは反省
・焼きそばパン
昼休みに学食のパン売り場へパシリさせるときに買わせるパン

【質問】アジャイルソフトウェア開発でよくベロシティという言葉を聞きます。これはどんなもので、これでメトリクスの指標として使っていいのでしょうか。

アジャイルのベロシティとは1回の開発単位でユーザ要求を提供できた量になります。スクラム開発では1回のスプリントで提供できたストーリーポイントの合計になります。

このようにベロシティでは提供できた要求の量やストーリーポイントを対象にしています。これらの量は絶対的な尺度ではなく同一/類似アジャイルチームでの相対的な尺度になります。つまり過去のものと比較して良いか悪いかの尺度になります。

これからベロシティはベンチマークに使えるような絶対的なメトリクスではなく、同一チーム内や類似チーム、類似プロジェクトの相対的なメトリクスになります。

ベロシティだけではベンチマークや見積もりなどの絶対的な指標ではなく、ベロシティの変化分や過去との比較をして、相対的に使うことができます。

例えば、過去と比較してベロシティが悪化したときのリスク検知やプロジェクト進捗状況、プロセス改善などに使うことができます。

ベロシティでリスク検知などに使えるという意味において、アジャイル開発でベロシティをメトリクスとして採用し、そのデータ収集をし、定量データ分析することは重要です。

【質問】アジャイルソフトウェア開発で見積りはどのようにするのでしょうか、見積りにはどのようなメトリクスがあるのでしょうか。

アジャイル開発で見積りをするときは、過去の類似のプロジェクトから類推をして見積もる類推法があります。この類推法はアジャイル開発に限った方法ではなく、開発プロセスに依存しない見積り法です。ただアジャイル開発では仕様変更なども類推することになりますので、より面倒になります。

ユーザの要求量を見積もる方法や、プロダクトそのものの価値を見積もる方法などがありますが、特定の分野やプロジェクトに限定されていて、汎用の見積りには難しいものです。

アジャイル開発における見積りは難しいもので、それを客観的に定量化するメトリクスも難しいものになります。例えば、ユーザの要求量（スクラムではストーリーポイントで代替）を測定するメトリクスも、どうしても主観的な面が入ってきます。

一方、このように難しい見積りの開発では、契約も請け負いでは対応できず、SES(システムエンジニアリングサービス)による準委任契約で実施されることが多くなっています。

SESでのメトリクスも工数などの少数の客観的な定量データを対象にするものしかありません。このため工数を元に契約することが多くなります。

仕事はどんだけ？

・涙がでちゃう
でもコードの中では平気なの、だってエンジニアだもの
・悲劇か喜劇
これらは表裏一体で、中の人は悲劇、外の人にとっては喜劇
・人類に残された最後のフロンティア
エンタープライズ号のカーク船長の言葉

【質問】アジャイル開発やDevOpsでは継続的に機能拡張をしていき、永遠のベータ版とも呼ばれていますが、こうしたものに対して、どのようにメトリクスを運用して品質保証をすればいいのでしょうか。

確かにアジャイル開発やDevOpsでは機能追加や変更をしていく継続的開発になり、継続的にリリースすることになります。

このため1回の設計・製造・テスト（開発）とリリースを行う従来の開発における品質活動とは異なる面も必要になります。

継続的に、段階的に開発とリリースを繰り返すときには、(1) 今回の開発とリリースで品質を保証するところと、(2)今回はそのメトリクスの計測対象にしない（リスク分析などの品質保証の対象にしない）ところを明確に区別する開発方法もあります。

これに加えて、(3)今回の開発により以前にメトリクス対象にして品質保証をしていた箇所を今回もその対象に入れるときは、対象に入れた影響がないかどうか（デグレードを起こしていないかどうか）を調べる必要があります。

上記(1)と(3)のメトリクス計測と品質保証活動は、部分的に見れば、従来の品質保証活動と同等な方法でも実施可能です。ただしこのような継続的に開発とリリースを繰り返すときには、品質保証活動の占めるコストが大きくなりますので、なるべくツールなどで自動化をして、そのコスト低減を図る必要があります。

いつまで続くの？

・いっぺん、死んでみる？
地獄少女の励ましの言葉、自死を勧めているのではない

おわりに「10年後のアジャイルメトリクス」