

マンガでわかる ソフトウェア開発 データ分析



ソフトウェア開発分析データ集2020
の正しい読み方と賢い使い方



目次



データ分析基礎編

- 01 [データ分析ってなんなの？](#)
- 02 [信頼幅の線、気になる](#)
- 03 [箱ひげ図のひげ、かわゆくない](#)
- 04 [散布図はぜんぜんばらばら](#)
- 05 [どれが本命なの？](#)

- データ分析
- 信頼幅
- 箱ひげ図
- 散布図と箱ひげ図
- 中央値と平均値

分析データ観察編

- 01 [生産性は性癖が出る？](#)
- 02 [バグを愛したソース](#)
- 03 [改修・保守が好き過ぎる](#)
- 04 [規模はアンバランスでアンビバレント](#)
- 05 [開発期間は短くて長くて短い](#)
- 06 [ウォーターフォールってつおい？](#)
- 07 [ここはツールでしょ](#)
- 08 [辛口レビューは正義](#)
- 09 [工期と工数は3乗根](#)
- 10 [果てしなき外部委託の果てに](#)

- 生産性
- 信頼性（不具合密度）
- 開発プロダクトの種別
- ソフトウェア規模
- 開発期間（工期）
- ウォーターフォール型開発
- 開発ツール
- レビュー
- 工期と工数
- 外部委託率

特別編

- 01 [特別編：データは欲しいけど](#)

- データ分析

画像提供：いらすとや

データ分析ってなんなの？

【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。

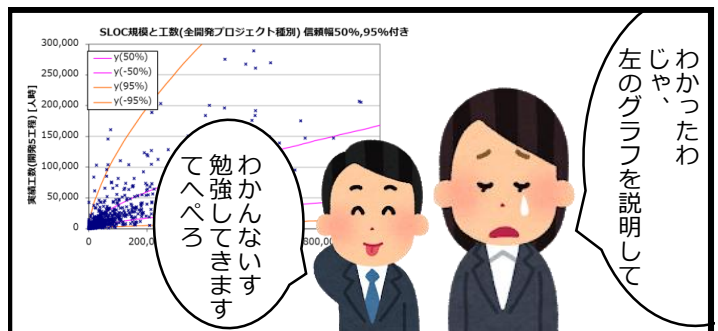
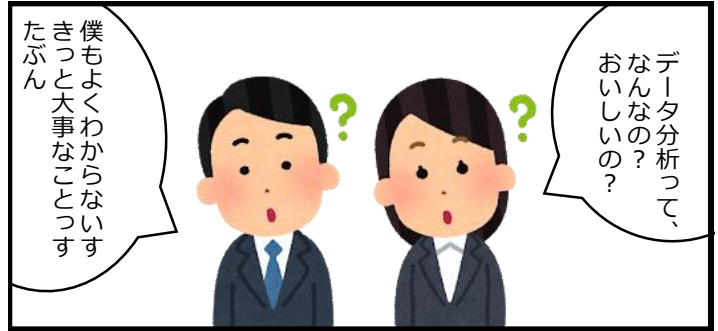
しかし属人的側面が大きいソフトウェア開発にあってはデータ分析はうまく行かず、またデータ分析の結果と同じようには、ソフトウェア開発が進まないことが多いです。

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。

信頼幅の線、気になる

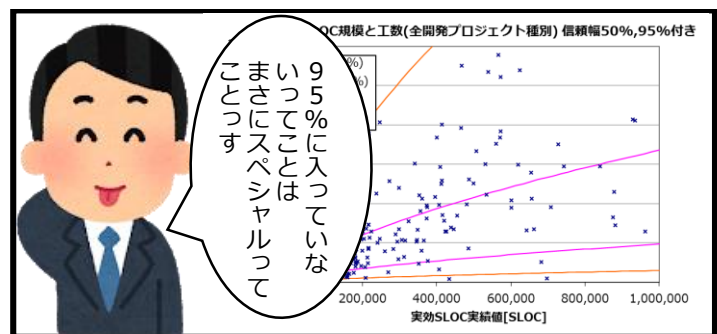
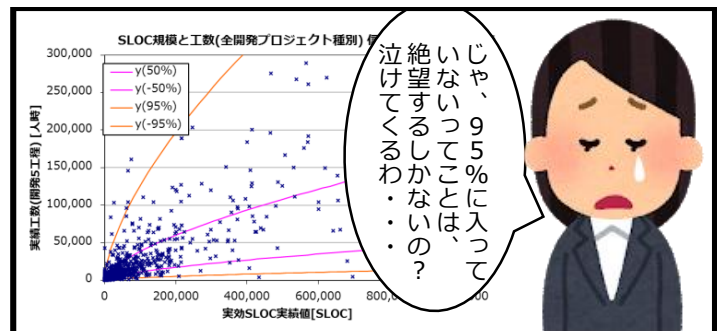
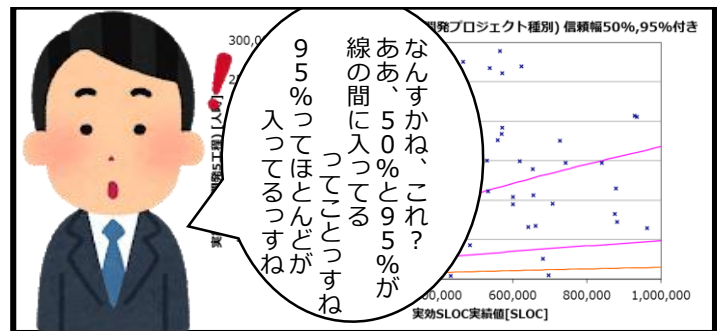
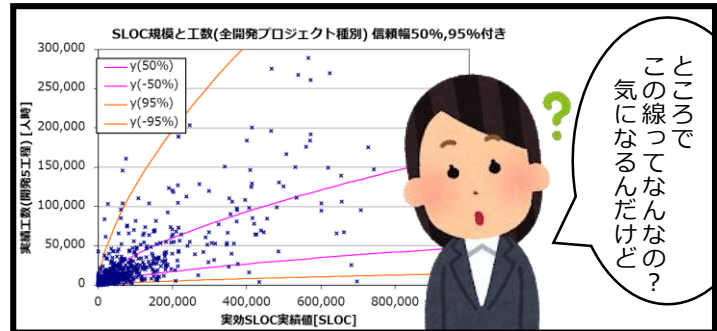
【解説】信頼幅

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。おかしい兆候があれば、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。



箱ひげ図のひげ、かわゆくない

【解説】箱ひげ図

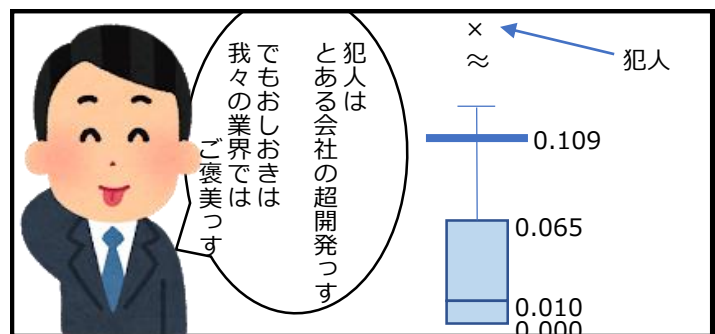
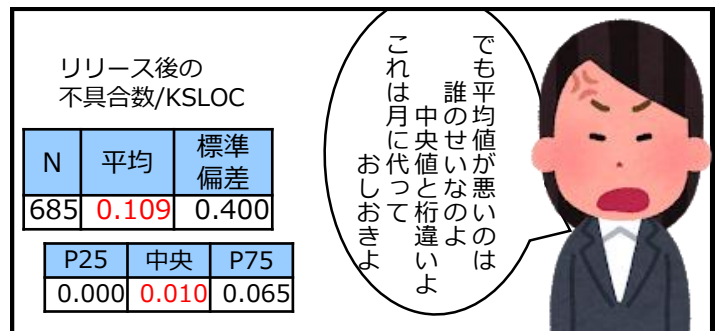
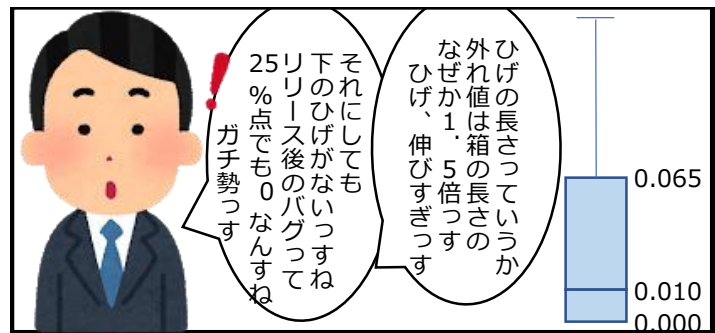
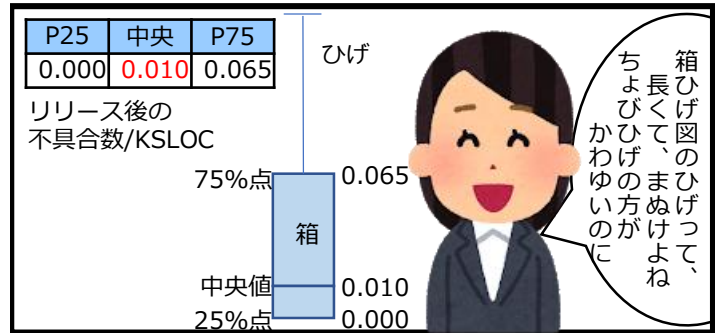
データを順に並べて25%地点(P25)の値から75%地点(P75)の値までを箱と表現し、中央値に線を引きます。箱の長さの1.5倍以上を外れ値として、外れ値でない最大値と最小値から箱までをひげで表現します。この箱ひげ図で分布の概観が一目でわかるようになります。

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。



散布図はぜんぜんばらばら

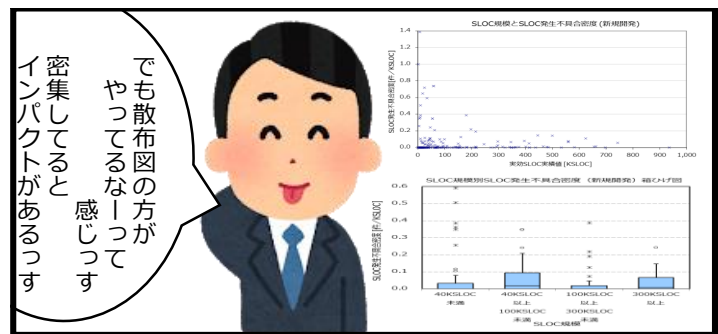
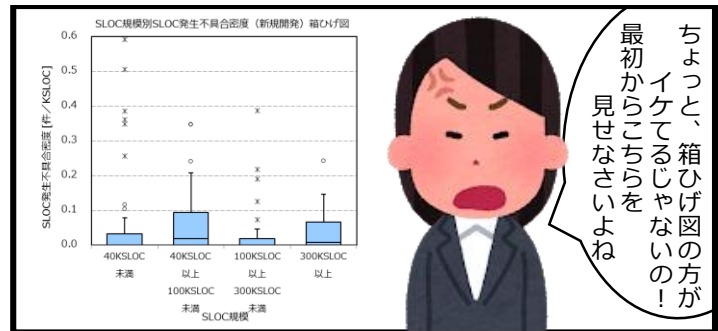
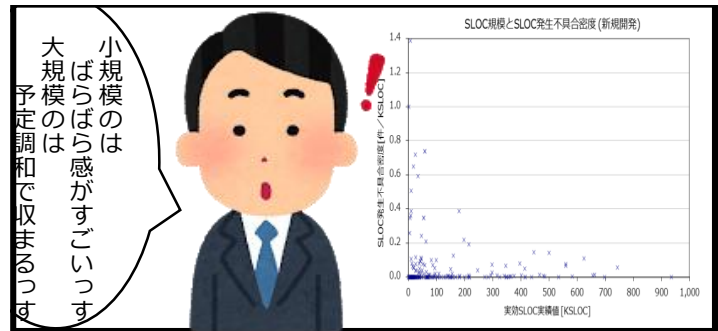
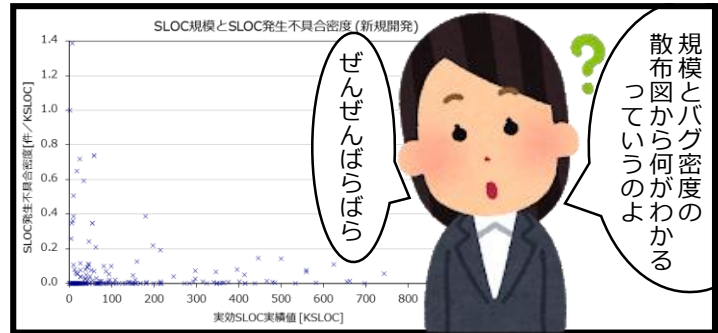
【解説】 散布図と箱ひげ図

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。



【解説】中央値と平均値

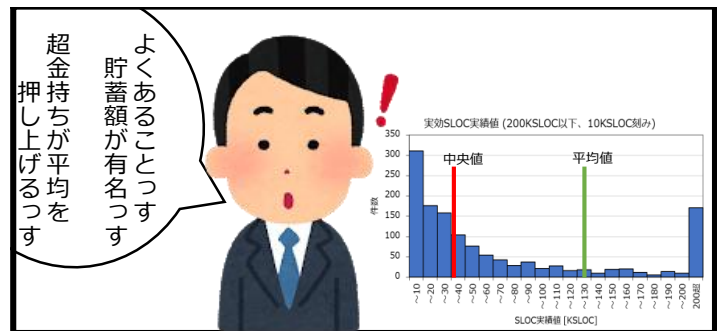
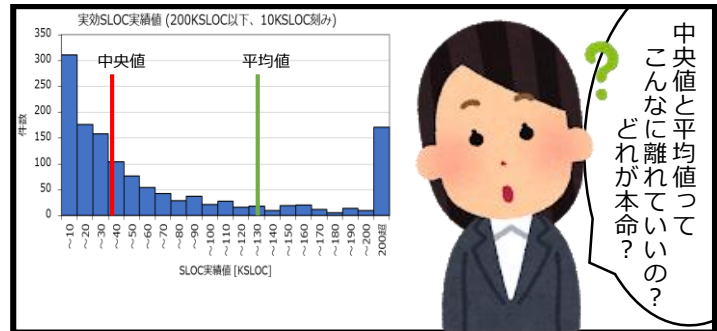
データ全体の特徴を代表する値として、平均値や中央値、さらに最頻値などがあります。

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？



生産性は性癖が出る？

【解説】生産性

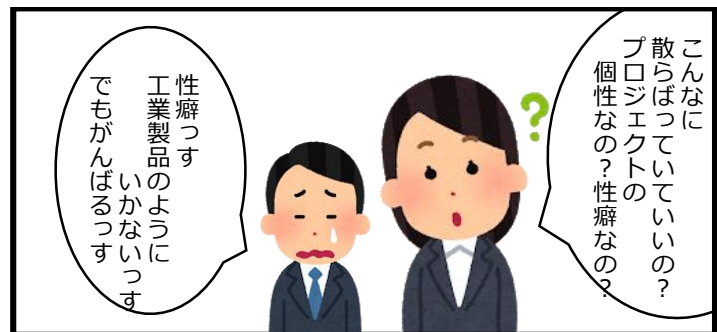
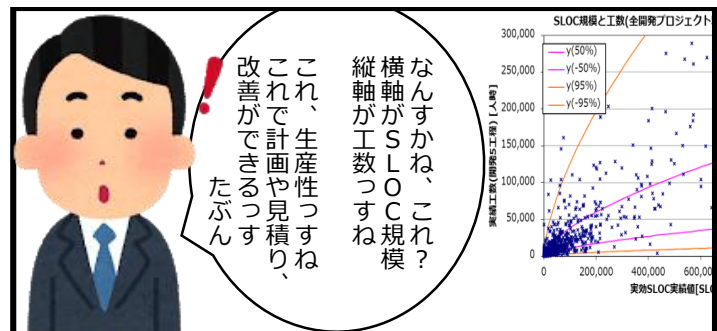
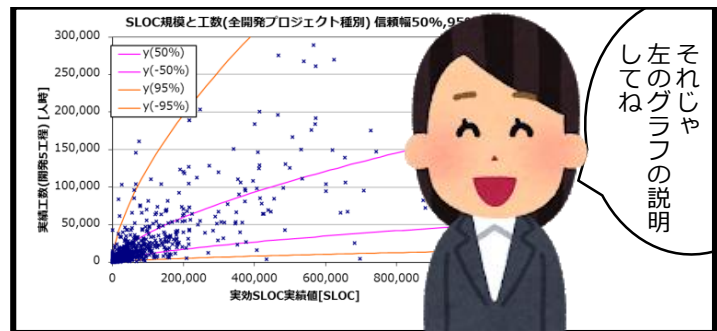
ソフトウェア開発のデータ分析をするときに、一番、気になるのが生産性です。

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。



バグを愛したソース

【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。

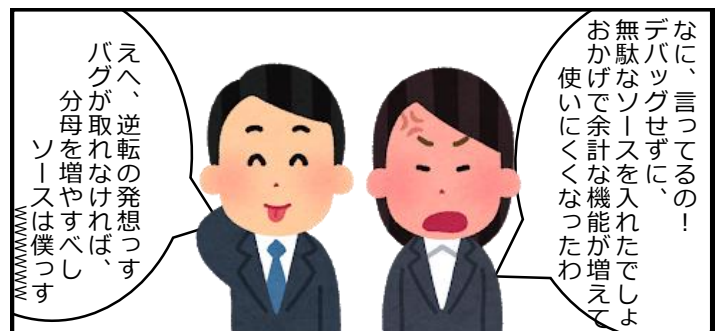
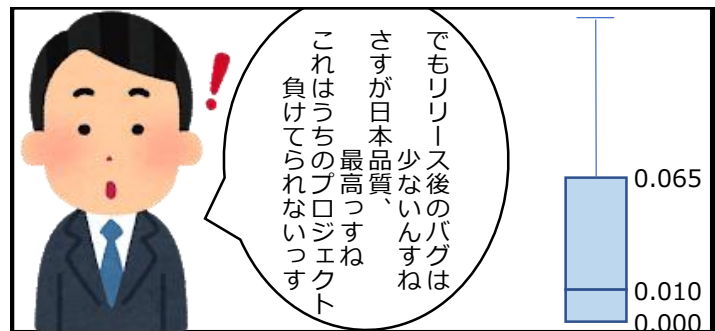
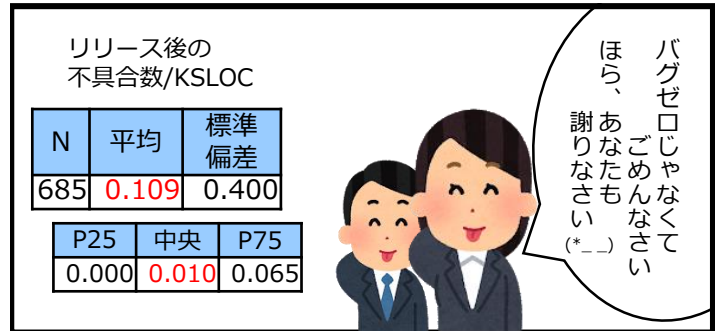
これを厳然たる事実として踏まえて、ソフトウェア開発と、そのソフトウェアの利活用をするようにします。

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たり、使いにくいシステムになってしまったりは、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

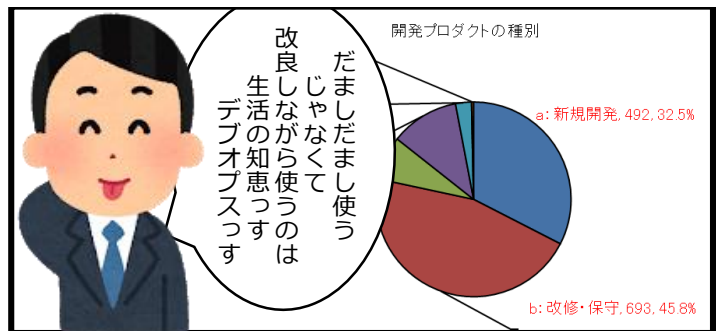
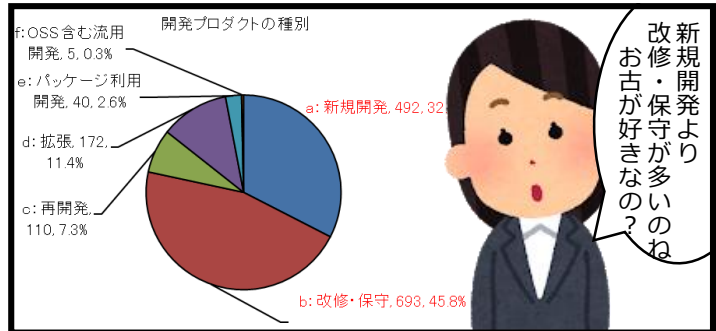
ソフトウェア開発は新規開発のものより、改修や保守、再開発、拡張するものが多くなっています。逆に新規開発はそれほど多くありません。

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。

最近では新規開発でも、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行数（SLOC、Source Lines Of Code）はマンガにあるように、大幅に違ってきます。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。

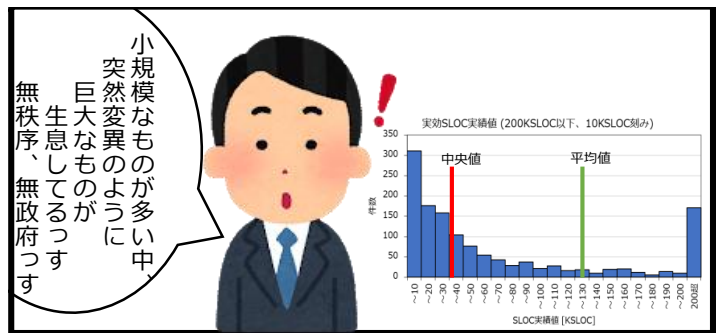
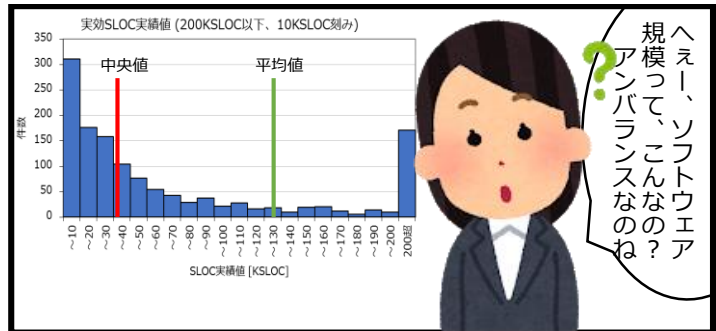
まずはこの散らばり具合を知って、さらに相場観として、中央値と平均値を知っておくことは大事です。

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）

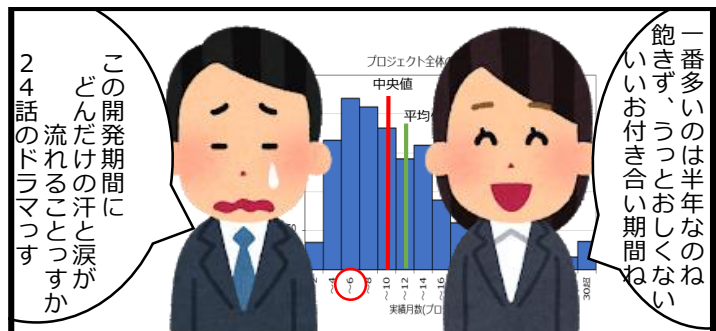
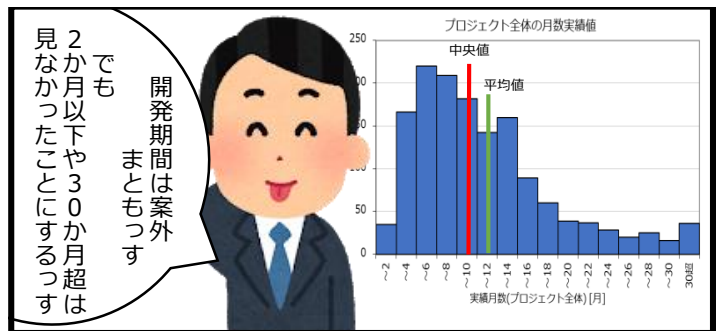
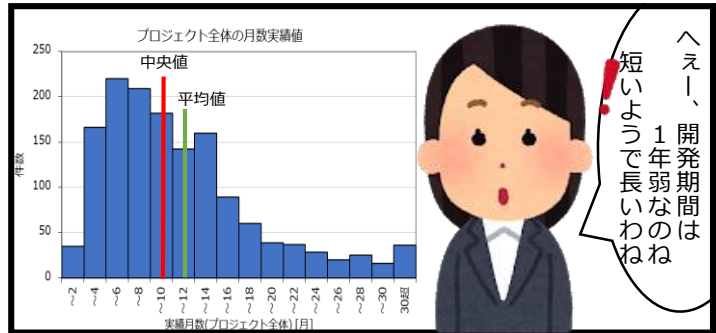
開発期間（工期）の分布は右のマンガのようになっています。中央値で9.1か月、平均値で10.8か月です。2か月で区切ったときの最頻値は4か月から6か月になり、その中間は5か月です。

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

ウォーターフォールってつおい？

【解説】ウォーターフォール型開発

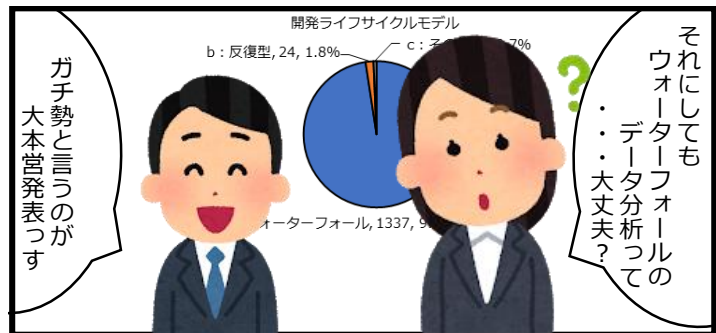
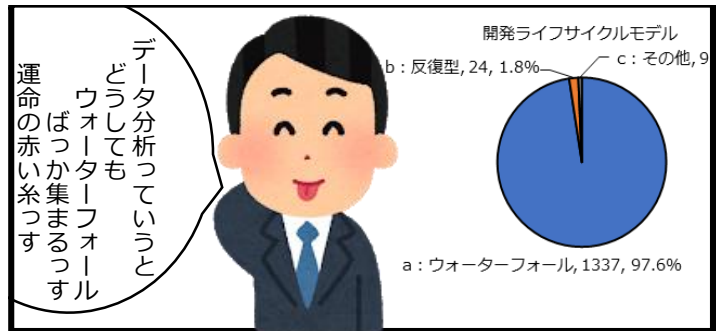
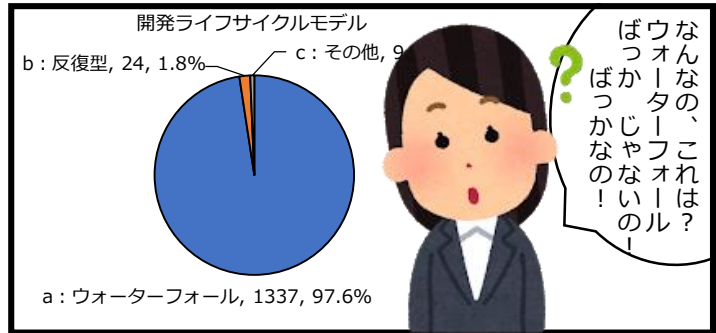
ソフトウェア開発のデータ分析になると、どうしてもウォーターフォール型開発が多く集まります。ウォーターフォール型開発は実に97.8%にもなっています。

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。



- ・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

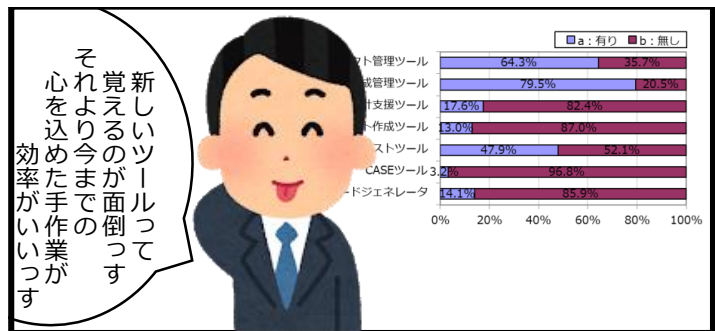
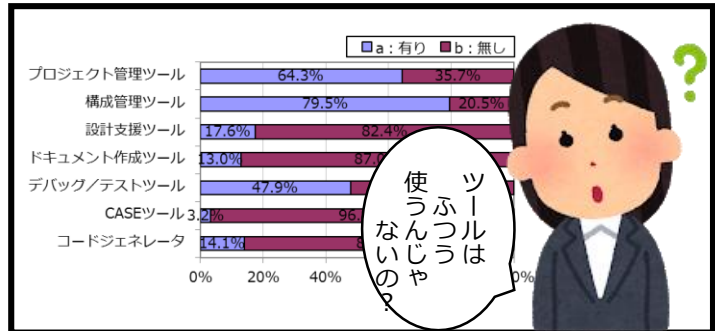
プロジェクトで使われているツールのプロファイルがマンガに掲載されています。グラフのようにプロジェクト管理ツールや構成管理ツールの管理ツール、デバッグ/テストツールの下流の支援ツールは多くのプロジェクトで使われています。

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。

ここはツールでしょ



・くあwse 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

【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。

辛口レビューは正義

レビュー指摘密度（件/KSLOC）

最小	P25	中央	P75	最大	平均
0.0	1.1	2.6	5.4	213.9	5.8



レビューって辛いのか？
甘いのか？
それともやらせ？
システム？

それにしても指摘が多いからね
なにか恨みでもあるのかしら



あの陰キャは何者なの？
うるさいけど
放置でいいの？

レビューアは意識高い系
がやってる
レビューイはエムツ
指摘されると喜ぶ




こんなにボロクソに！
まるで鬼上司みたいー
なぜレビューで褒めて
あげないのかしら

レビュー指摘はバグと
おんなじ
レビューの成功は
間違いを指摘すること
鬼上司も正義

密度は
2桁の桁違い
これは神とミトコンドリア
くらいの差
もう笑うしかない

最小	P25	中央	P75	最大	平均
0.0	1.1	2.6	5.4	213.9	5.8



でも213.9は恐怖ね
5行に1個の指摘なんて
神をも恐れぬレビューア
とプログラムね

・陰キャ 陰気なキャラクタ

工期と工数は3乗根

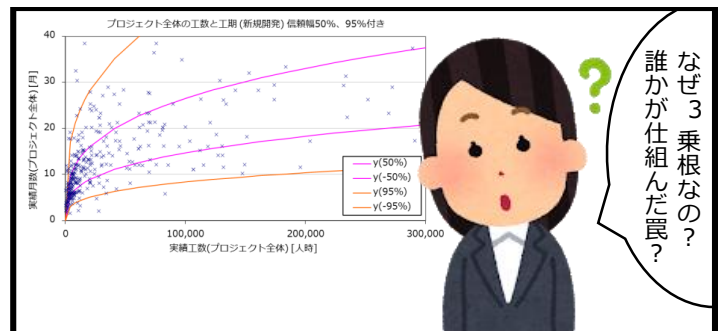
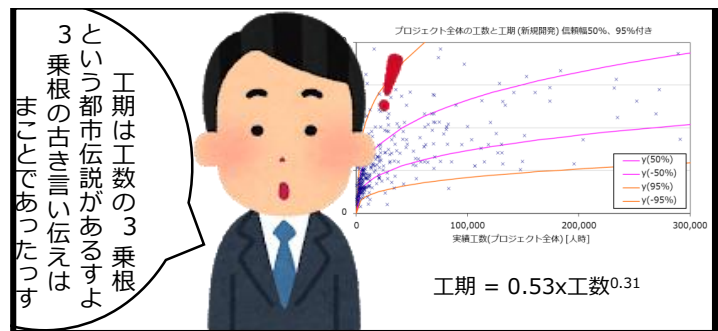
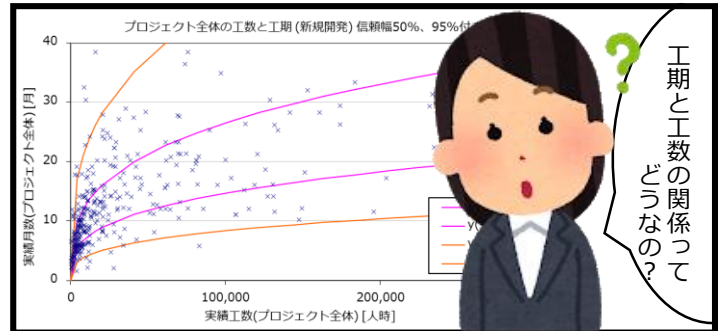
【解説】 工期と工数の関係

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す。



果てしなき外部委託の果てに

【解説】外部委託率

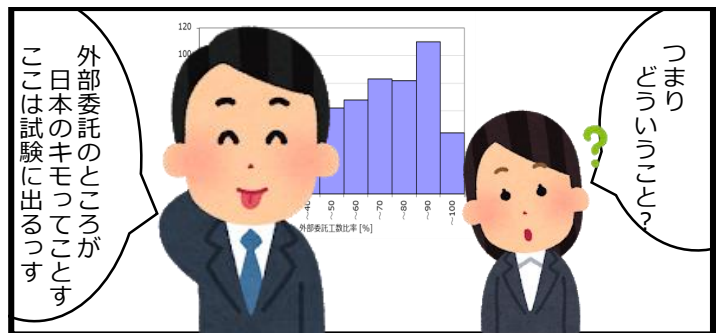
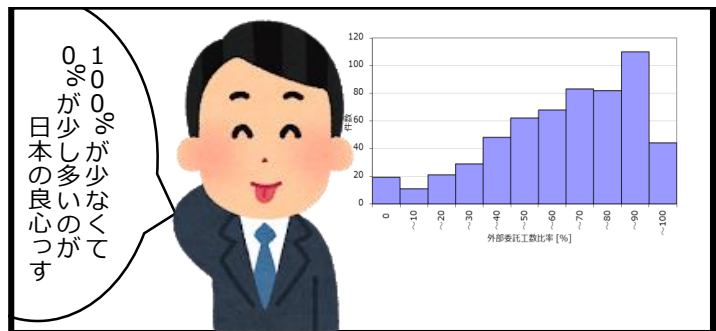
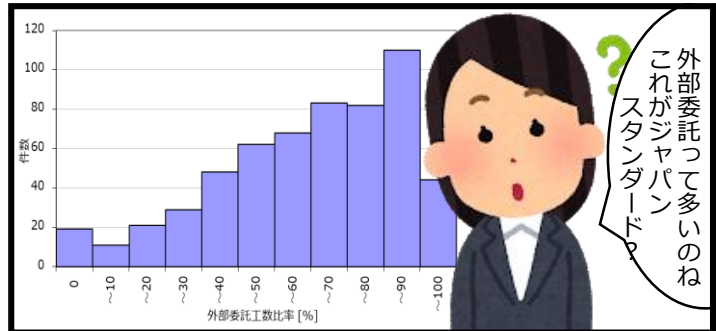
ソフトウェア開発における外部委託率の分布はマンガのように、高い傾向にあります。

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

