品質判定マップによる
人員配置とテストマネジメント

（株）ビーイング
鈴木　誠

発表の流れ

- 大きなプロジェクトは難しい。
- 本論
  - 品質判定マップの作成とその効果
  - 品質判定マップによる人員配置
  - 品質判定マップと工程表
- まとめ

ソフトウェアプロジェクト

規模が小さいと・・・

プロジェクトに関わる人数が少ない。
システム全体を把握し易く、機能間の不整合も気付き易い。

規模が大きくなると・・・

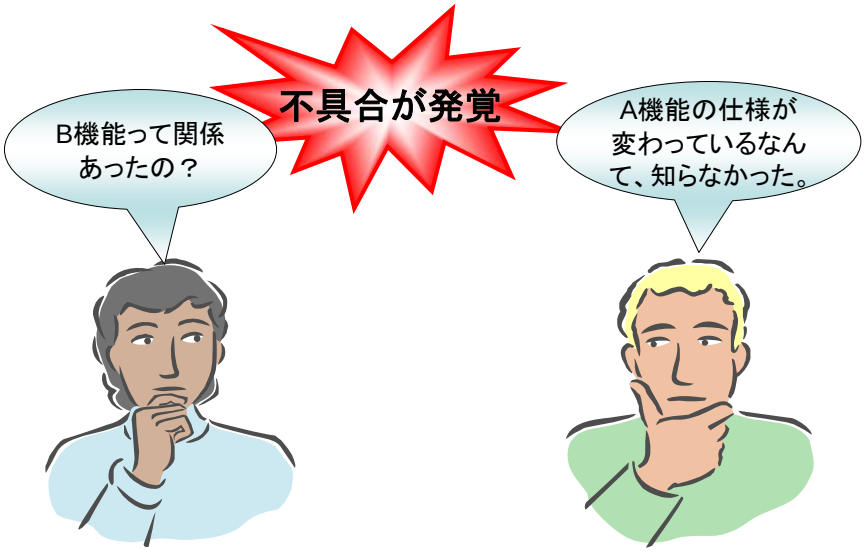
プロジェクトに関わる人数が増える。
自分がシステム全体の何処を担当しているか分かり辛くなる。
コミュニケーションが上手く取れていない所が発生する。

開発者が必要な情報を知らないという事態が発生する。
知らない間に、仕様が変更されていたりする。

同じ人が関連機能を開発していたら？

違う人が関連機能を開発していたら？

B機能のテストを実施すると・・・



ソフトウェアプロジェクト

大きなプロジェクト程、難しい。

規模が小さいと・・・

プロジェクトに関わる人数が少ない。
システム全体を把握し易く、機能間の不整合も気付き易い。

規模が大きくなると・・・

プロジェクトに関わる人数が増える。
自分がシステム全体の何処を担当しているか分かり辛くなる。
コミュニケーションが上手く取れていない所が発生し

不具合の温床となる。

開発者が必要な情報を知らないという事態が発生する。
知らない間に、仕様が変更されていたりする。

ソフトウェアテスト工程

プロジェクトの規模が小さいと

システム全体を把握し易く、機能間の不整合も気付き易い。

プロジェクトの規模が大きいと

プロジェクトに関わるテスト者が増える。
自分がシステム全体の何処をテストしているか分かり辛くなる。

テスト者が必要な情報を知らないという事態が発生する。
知らない間に、仕様が変更されていたりする。

全ての関連機能を把握していたら？

全ての関連機能を把握していなかったら？

他の人が操作したら

ソフトウェアテスト工程

大きなプロジェクト程、難しい。

プロジェクトの規模が小さいと

システム全体を把握し易く、機能間の不整合も気付き易い。

プロジェクトの規模が大きいと

プロジェクトに関わるテスト者が増える。
自分がシステム全体の何処をテストしているか分かり辛くなる。
コミュニケーションが上手く取れていない所が発生する。

幾ら高度なテスト技法を使っても・・・。

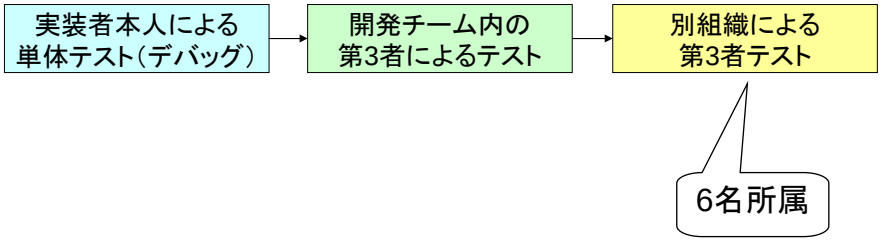
不具合を見付けれない。

テスト者が必要な情報を知らないという事態が発生する。
知らない間に、仕様が変更されていたりする。

品質判定マップ作成に至るまで

私の所属部署

開発チームからは独立した検査組織
　次から次へと異なる商品の検査を行っている。



ことのはじまり

- 2007年5月末
  - 開発チームからテストを依頼される。
- 商品は・・・
  - 当社の中では、ずば抜けて大きなシステム
- 完成予定日は・・・
  - 2007年7月末。
- 仕様は・・・
  - この時点では、全く知らない。
  - テスト対象のソフトウェアの機能説明を受ける。（0.5日）

テストを始めると・・・

- 予想はしていたが、不具合続出。
- 実装者のテスト結果は『OK』と書いているのに、その通りの操作を行っても、不具合になることが多い。
- テスト項目と、実際のシステムが一致しない。
  項目に〇〇ボタンを押すとあっても、〇〇ボタン自体がない。
- 従って、予想以上に、テストの進捗具合が悪い。
- 連日、深夜までテストする日々が続く。

テストマネージャーの思惑

- 市場投入後に、トラブルが多発するような事態は避けたい。
  - 不具合が収束しているか、発散しているか判断したい。
  - テスト漏れを防ぎたい。
- 期日迄に、テストを終了させたい。
  - 現在の状況を把握したい。
  - いつ頃、不具合が収束するのかを予測したい。
- テスト効率、不具合検出率を向上させたい。
  - テスト者が短期間でシステムを把握できる仕組みを作りたい。
  - 不具合が多い機能を優先的にテストさせたい。

信頼度成長曲線と、機能別不具合件数分析を使用した。

信頼度成長曲線

＜利点＞
- システム全体の不具合発生状況を把握できる。
- 潜在不具合数や、不具合収束迄のテスト時間を予測できる。

＜利用上の注意＞
- 不具合の多い機能を優先的にテストしなければ、後半になって発散する。
- 後半になって、テスト者を加入すると、実態よりも収束傾向を示し始める。
- 実装とテストを併行している場合、収束と発散を繰り返し、段々になる。
- 減少傾向を示していない段階での予測は、ほとんど当てにならない。
- 不具合の重要度を考慮せずに、不具合件数とテスト時間でプロットした場合、収束までの時間が長くなる。

不具合がまだまだ検出されるということが分かるだけで、
それ以上、有効な情報は得られなかった。

機能別-不具合件数分析

＜利点＞
- どの機能で不具合が多く発生しているか分かる。

＜利用上の注意＞
- 既に大部分が修正されていて、安定している機能でも、不具合が多いと判断される場合がある。
- テストしていない機能は不具合が少ない機能と判断される。

必ずしも、分析結果がテスト者の感覚と一致しない。

どの機能間に不具合が多いということまで把握したい。

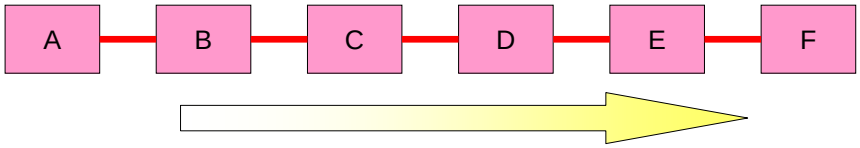
『見える化』したい事

- システムの全体像
- 機能間の繋がり
- どの機能が危険か？/安定しているか？
- どの機能間に問題があるか？

品質判定マップの導入

品質判定マップの作成とその効果

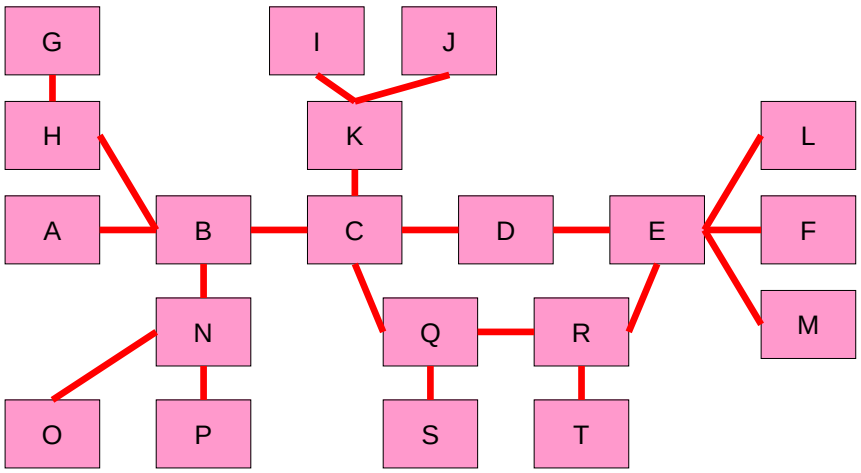
ユーザーの操作の流れに



運用方法から、機能、及び、ツールを全て洗い出す。

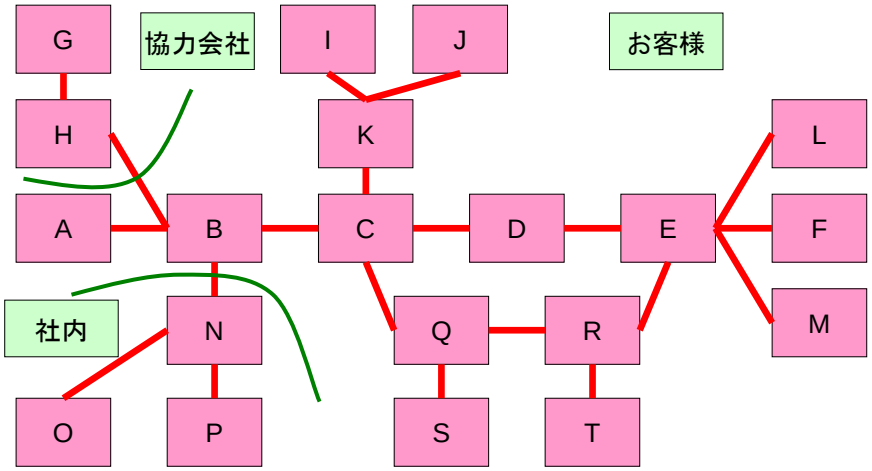
使い手が最も行いそうな操作順に、機能を並べる。（左から右へ）

全ての機能／ツールを繋げていく

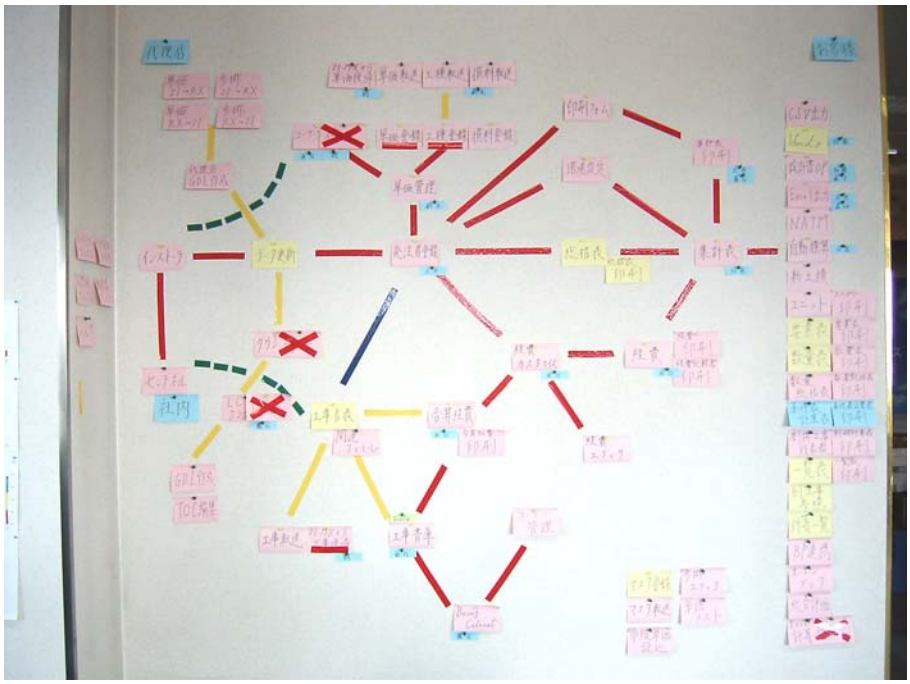


対象のソフトウェアの機能以外でも、関係があれば拾い出す。

使い手を明確に



使い手を明確にすると、どういう現象が起きたら問題になるか想像し易い。
品質を安定させなければならない期日も違うかもしれないし・・・



機能の品質判定

赤	とても危険。
赤	危険。or 未テスト
黄	まだ、注意が必要。
青	そろそろ大丈夫かも。
	テストできません。

色で表すと、判定が分かり易い

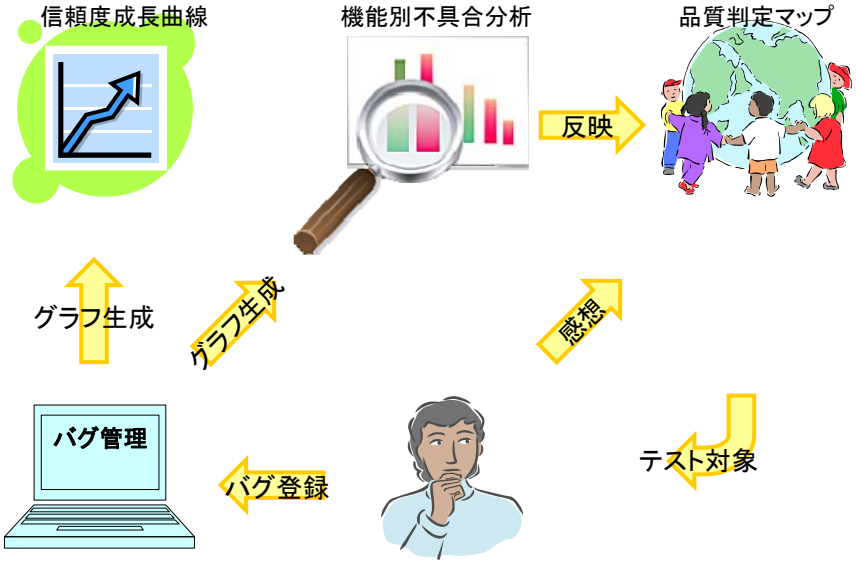
バグの重要度決定基準

レベル	定義	事例
重要度S	お客様の業務/使用上に、危機的影響を与える。	・・・
重要度A	お客様の業務/使用上に、重大な影響を与える。	・・・
重要度B	お客様の業務/使用上に、限定的な影響を与える。	・・・
重要度C	お客様の業務/使用上に、ほとんど影響がない。	・・・

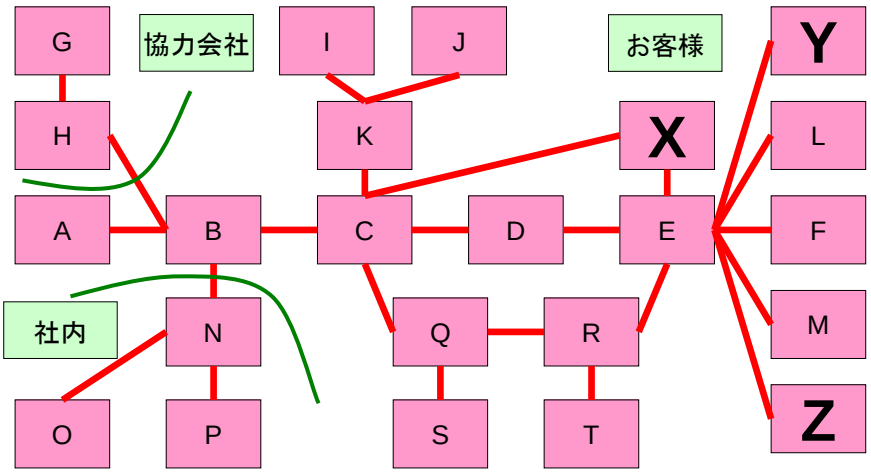
重い

軽い

テストの全体像

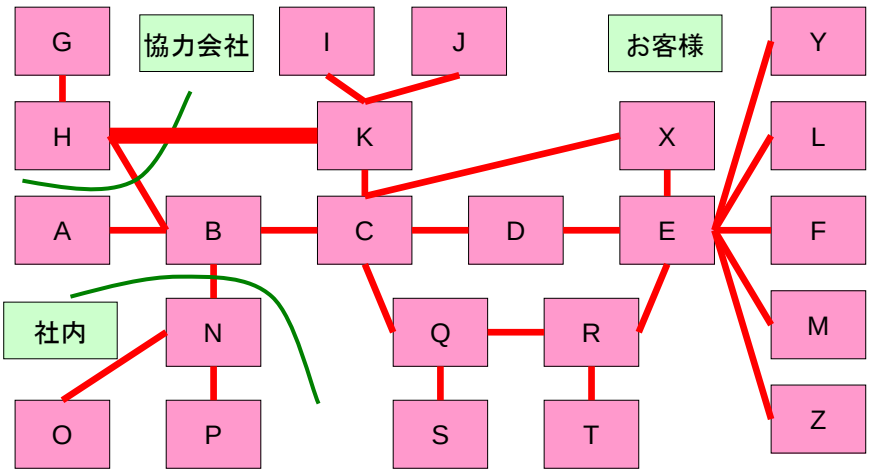


あれはテストしなくていいの？



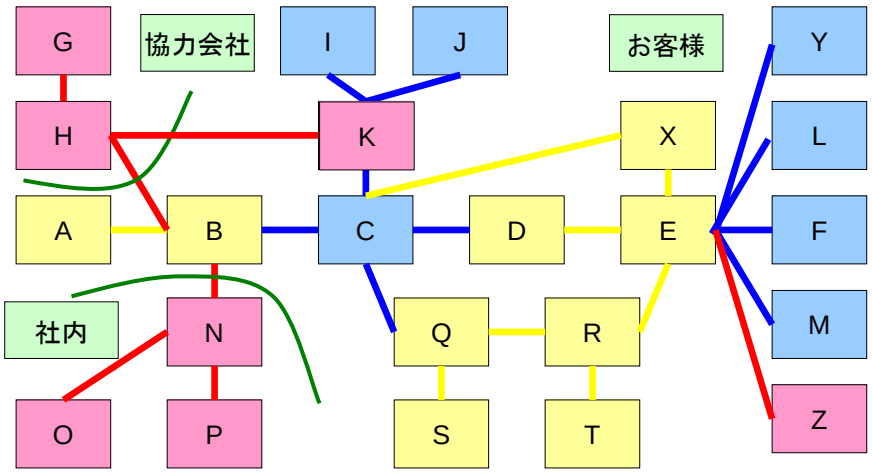
運用／販売方法と照合すると、マップにないものが見つかる。

あの連携は危険だな・・・



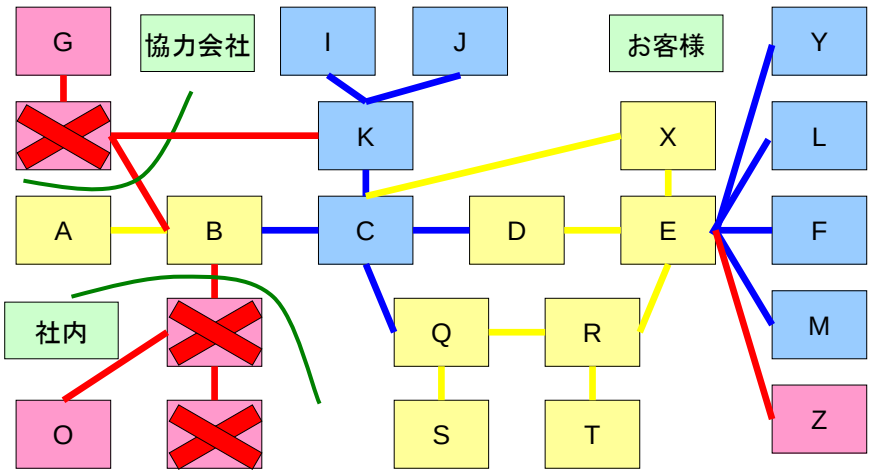
テストを進めると、想定外の危険な連携部分が見つかる。

不具合が見つかると・・・



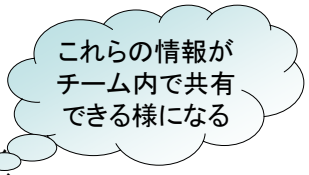
青（安全）と判定されていても、黄（注意）や赤（危険）に戻る。

テスト出来ない・・・



テスト出来ない機能は、×印を付ける。

品質判定マップから見えるもの



- システムの全体像。
- 機能間の繋がり。
- ユーザーが使用する流れ。データの流れ。
- どれが重要な機能なのか？（ど真ん中に）
- テストできない機能（重大な不具合がある。or 未実装）。
- まだ、テストしていない機能。（テスト漏れ）
- 危険な機能。or 安全な機能。
- だれが使用する機能なのか？
- テスト時に整合性を確認する必要がある機能が何か分かる。
- どの機能をテストし直す必要があるか分かる。

品質判定マップによる人員配置

全部を把握出来ないのなら・・・

- 全員がシステム全体を把握しているのが理想。
- でも、規模が大きくなると、全体を把握できなくなる。
- 原因は・・・
  - 情報量が能力の限界を超えている。
  - 把握するために必要な時間が取れない。

どういう情報の欠落の仕方がベストなのだろう？

人員配置の工夫（Ｉ）

関連性の高い機能を同一テスト者に任せる。

アイデアのつくり方

アイデアのつくり方＜ジェームス.W.ヤング＞

テスト項目の
作り方も同じ。

- データ集め
- データの咀嚼
- データの組合せ
- ユーレカ（発見した！）の瞬間
- アイデアのチェック

・アイデアとは既存の要素の新しい組合せ
以外の何ものでもない。
・既存の要素を新しい組合せに導く才能は、
物事の関連性を見つけ出す才能に依存するところが大きい。

テスト項目のつくり方

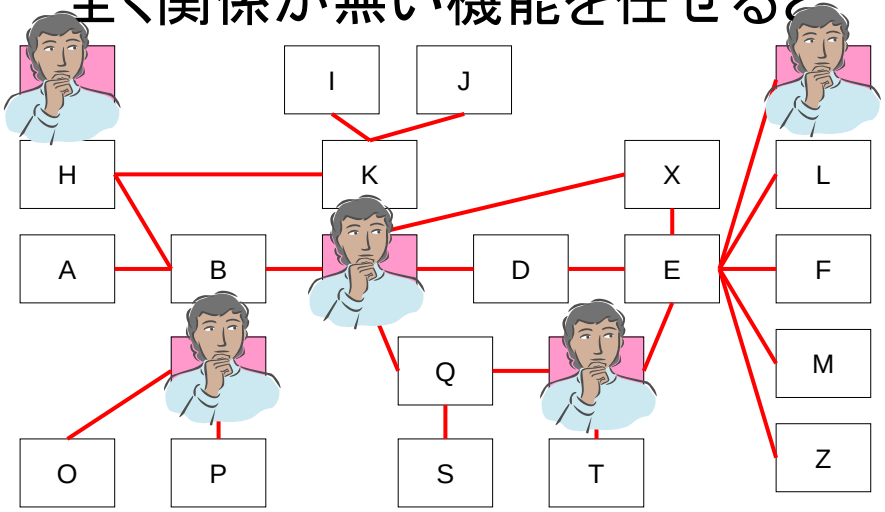
テスト項目のつくり方＜アイデアのつくり方風＞

- 資料を収集する
- 資料の読み込み。仕様の理解
- 不整合がないかいろいろ組合せて検討
- あれ！〇 〇すれば不具合になるのでは？の瞬間
- 実際に、テストしてみる。

関連性の高い
情報を知れば
知るほど・・・

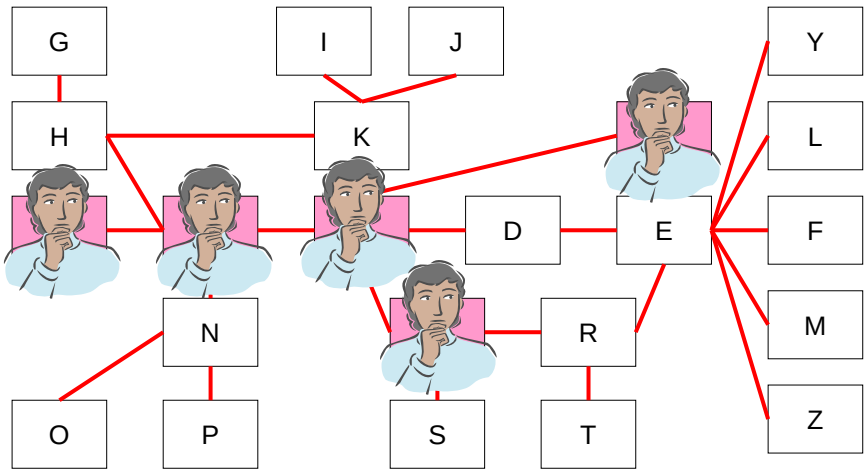
テスト項目の
アイデアは浮
かんでくる。

全く関係が無い機能を任せると



仕様理解に時間が掛かる。機能間の問題は、気づき難い。

関連性のある機能を任せると



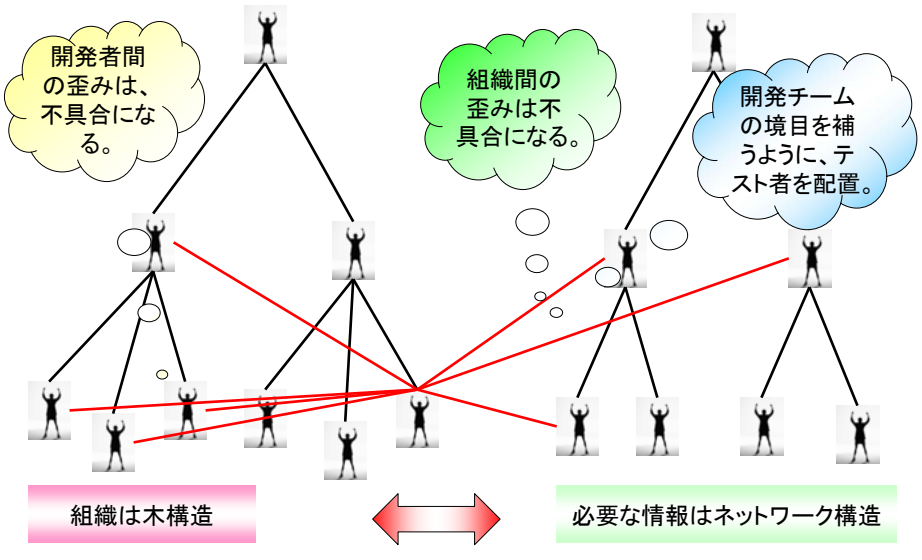
仕様理解が短時間で済む。機能間の問題にも気づき易い。

人員配置の工夫（Ⅱ）

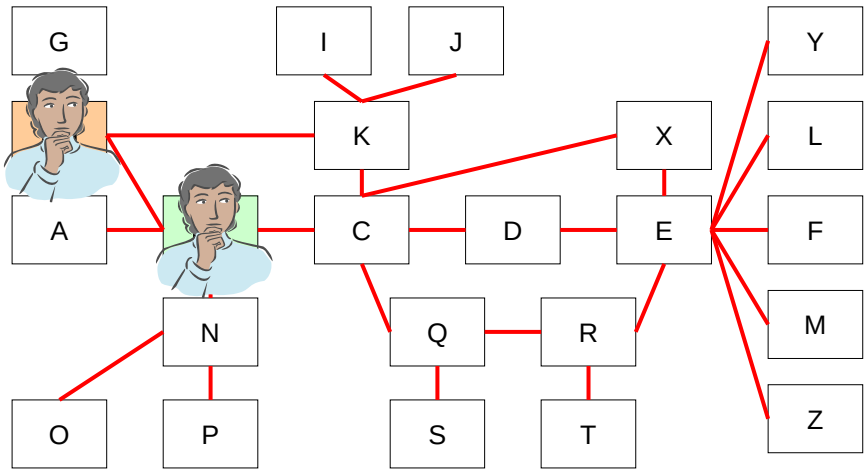
開発者の境目を補うように、配置する。

組織の境目を補うように、配置する。

組織と必要な情報

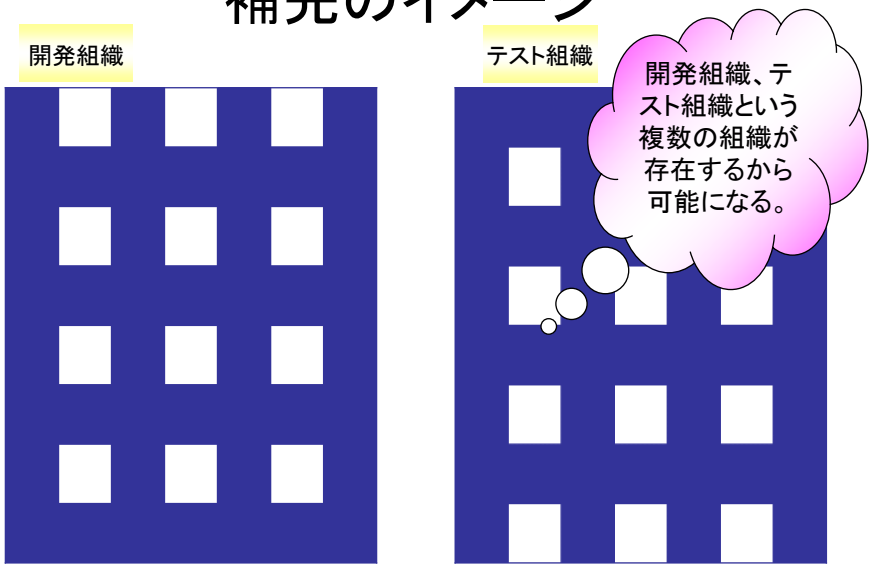


関連機能の開発者が異なるなら



意思疎通の問題があるかも知れないので、同一人物に任せる。

補完のイメージ



テストが進捗するに連れて・・・

- どの機能が不安定であるかが判明する。
- どの機能と、どの機能の連結が不安定であるかも判明する。

- 誰が作成した機能が不安であるかが分かる。
- 誰と誰の間で、コミュニケーションの問題があるかも分かる。

品質判定マップと工程表の相乗効果

工程表の作成

- 品質判定マップから
  - 現在の品質が明確である。
  - テストする機能も明確である。
- 満たさなければならない品質も把握している。

↓

今後、品質を上げるための作業（テスト量）が明確になる。

↓

工程表を作成することは容易である。

危機感を共有

- 現在の品質が明確である。
- 工程表も作成されている。
- 納期迄の期間も分かっている。

↓

- 実現性が高いか、低いか容易に判断できる。
- チーム内で危機感を共有できる。
- 根拠の無い楽観的な考えは芽生えない。

積極的なリスク対策

- 工程が完成している。
- 不具合があった場合、やり直す必要がある機能が分かる。

↓

- どこが遅れたら、全体工程に影響が大きいか把握できる。
- 現在の問題点が把握できる。
- ボトルネックになっている機能が把握できる。

↓

- テストチームからどの修正を優先して欲しいと要求できる。
- 後半になって、問題が発覚したら困る内容を防ぐように、事前に手を打つことができる。

まとめ

まとめ

- メンバー全員がシステムの全体像を捉える事ができる。
- メンバーがシステム全体の何処を担当しているか分かる。
- 各機能ごとに、品質判定ができる。
- テスト工程が引き易い。
- 仕様変更時に、テストし直す必要がある機能が漏れ難い。
- チームで危機感を共有できる。
- 開発者ごとの問題が明確になる。
- 開発者間の意思疎通問題が明確になる。
- 組織間の意思疎通問題が明確になる。
- 問題を踏まえて、テスト者の配置が可能になる。
- テスト者が自発的に必要な情報を集めるようになる。
- テスト者同士が情報交換し易くなる。
- テストマネージャーが的確な判断を行い易い。