コンテキストの理解による技法、事例の分析
― 苦手の対策にむけて ―

森崎 修司
静岡大学 情報学研究科／名古屋大学 情報科学研究科

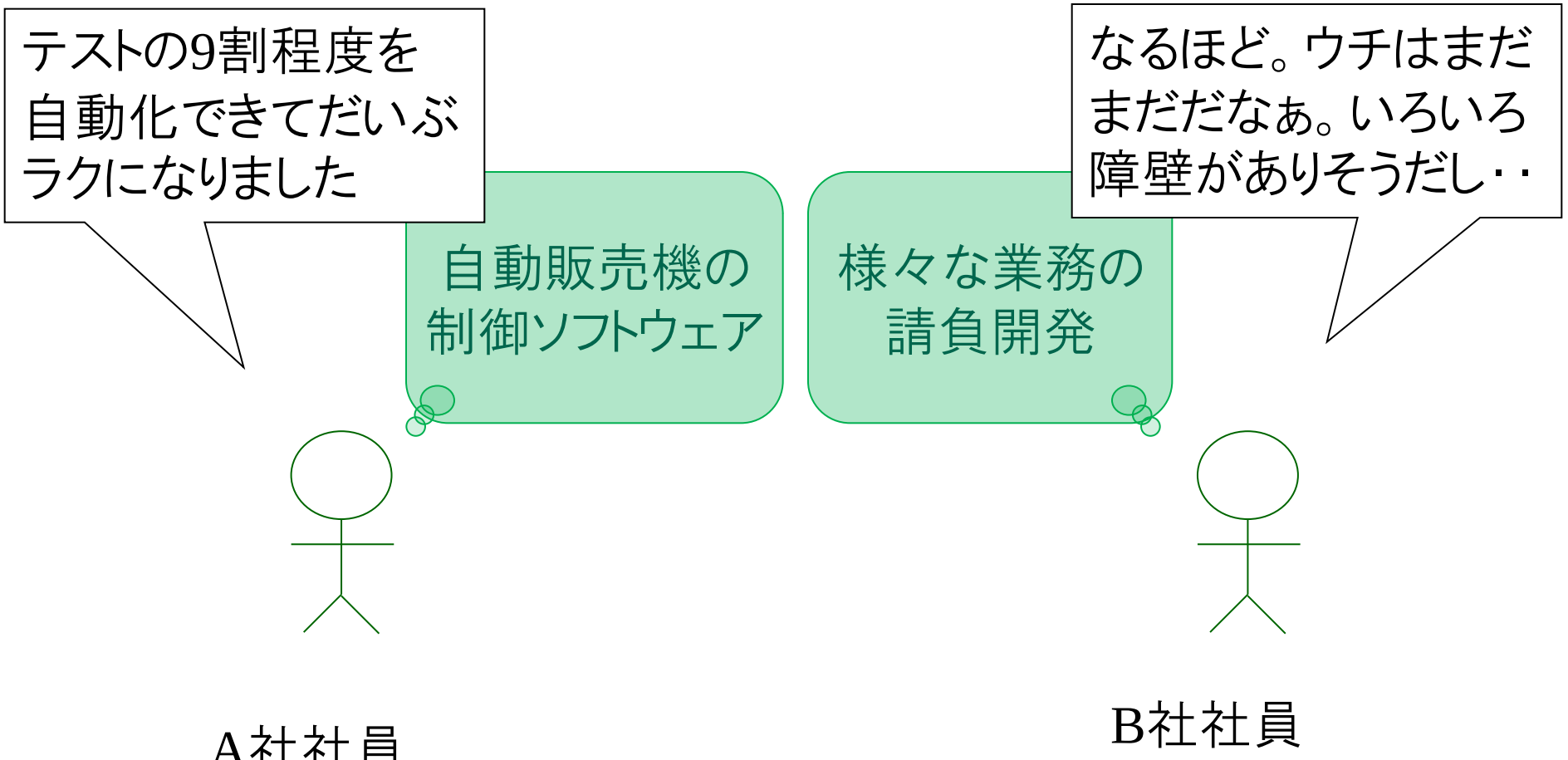
http://blogs.itmedia.co.jp/morisaki
http://twitter.com/smorisaki

名古屋大学 NAGOYA UNIVERSITY

自己紹介 – 森崎 修司

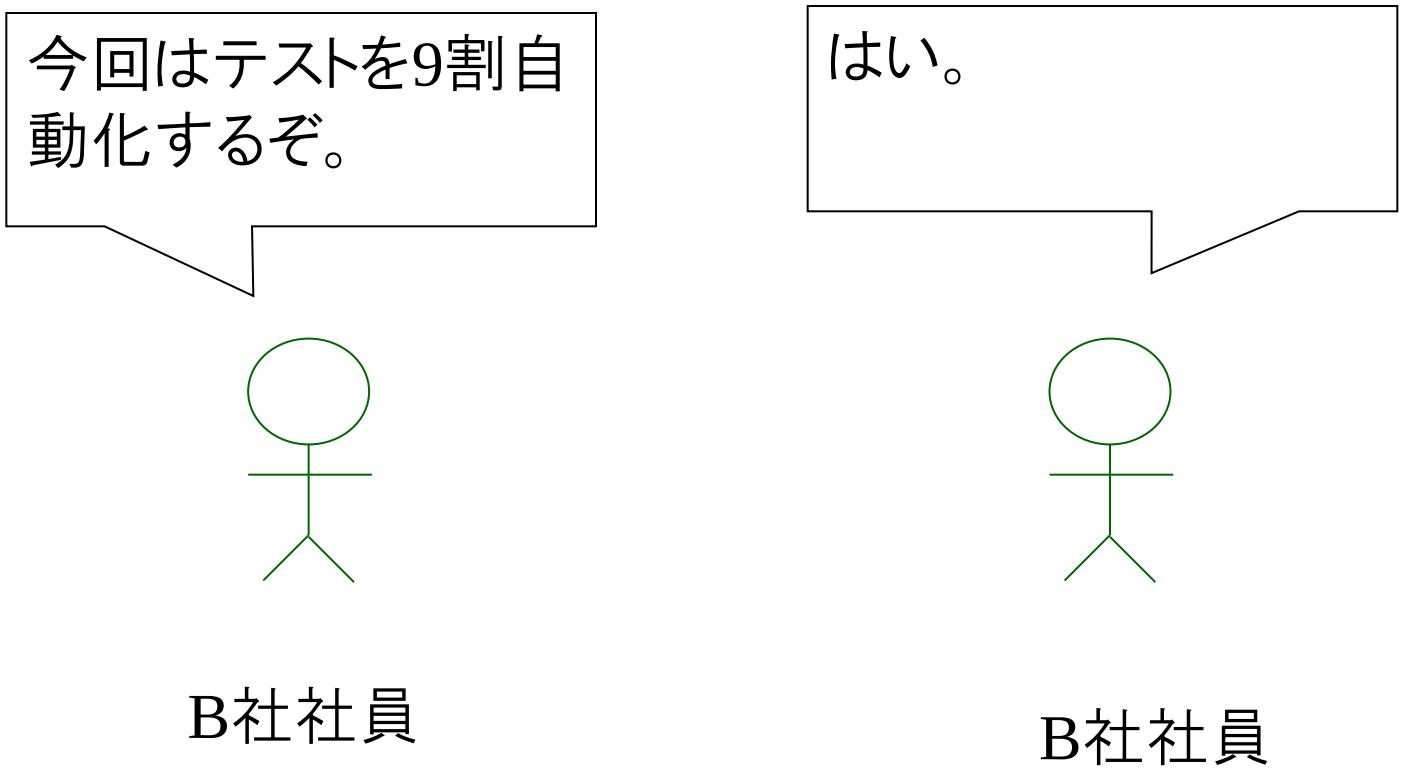
- サービス開発、SIに携わっていた。
  - サービス開発
    - 不特定多数のユーザを前提に定型サービスの開発
    - 想定ユーザ（ペルソナ）の設定をはじめ、実装、運用まで
  - システムインテグレーション
    - エンドユーザから受注し、基本設計よりも後を再発注
  - となりの部門では組込み用ソフトウェアを開発しており、そこでの話も漏れ聞いていた。
- 実証的ソフトウェア工学の研究に従事している。
  - 国際シンポジウムのプログラム委員
  - 多くのソフトウェア開発企業と対話・相談している。
    - 7年間で400社との相談、30社程度とNDA下での連携

勉強会の懇親会で

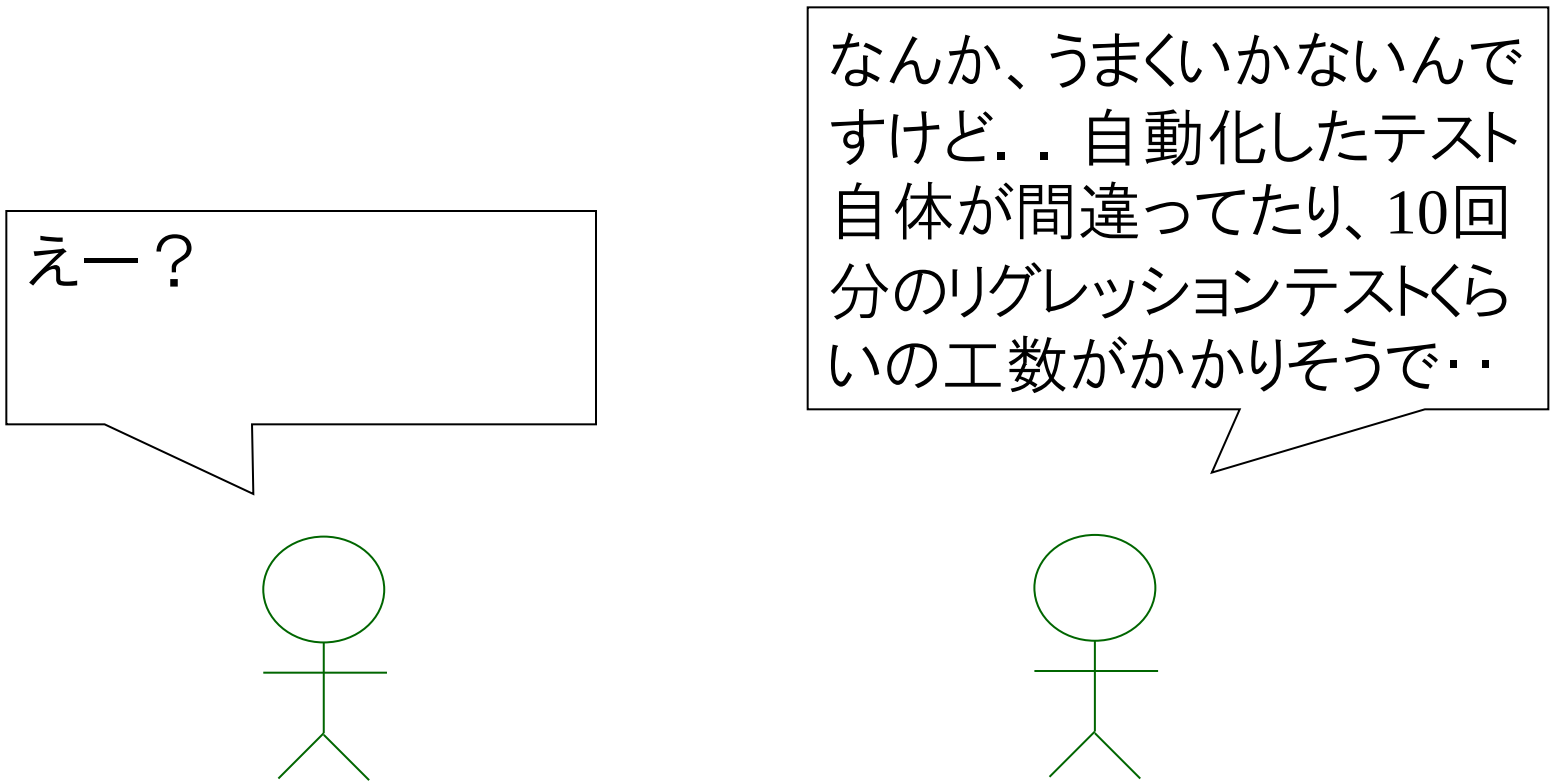


会社に戻り・・・

- 次のプロジェクトで・・・

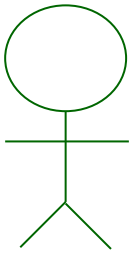


開発途中で･･･

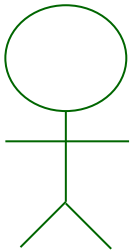


開発が終わり. . .

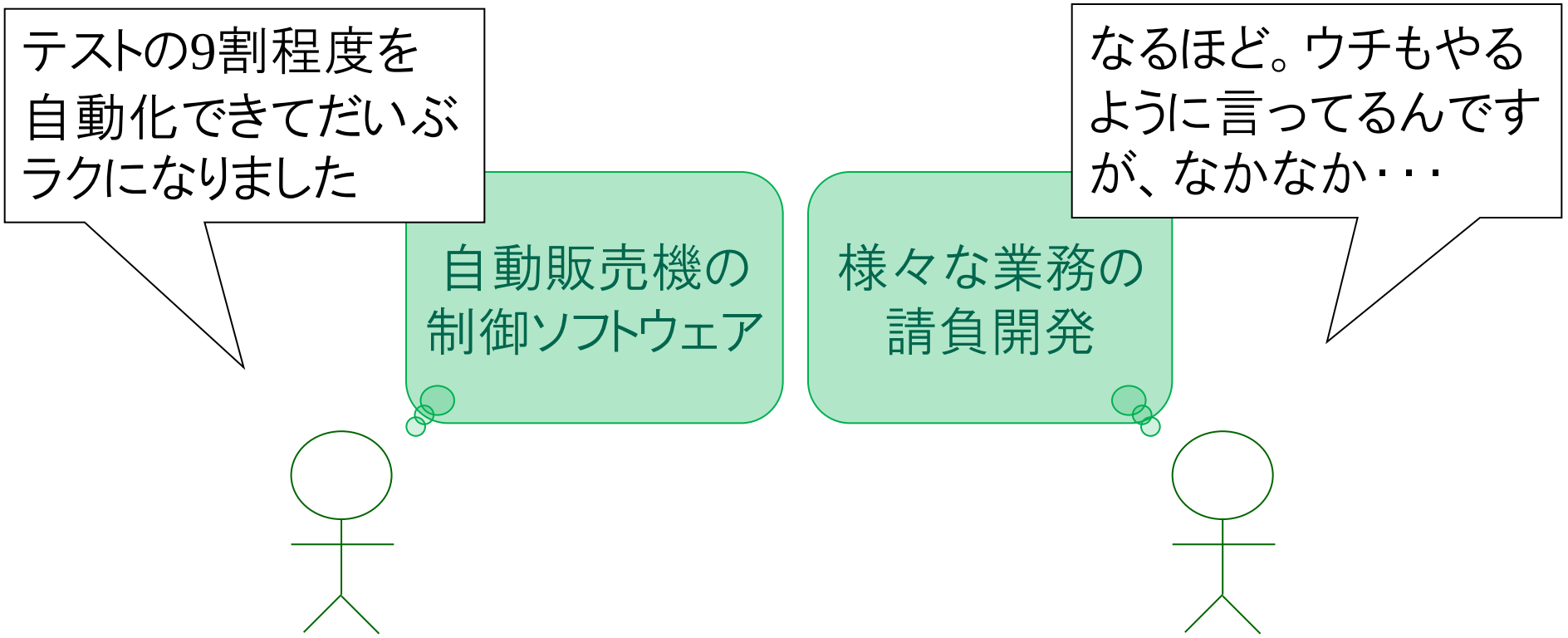
自動化テストってダメなんじゃない？
あいつ、わかってないなぁ



コストも納期もオーバーしてしまいました. .



勉強会の懇親会で(再掲)



コンテキストと効果

- 他で効果のあった施策を自身でも試す場合には、施策の効果が得られるコンテキストを明らかにする必要がある。

- 単純に「組込みソフトウェアだから」「ソフトウェア規模が大きいから」とコンテキストを定義してもうまくいくとは限らない。

勉強会の懇親会で(再掲)

テストの9割程度を自動化できてだいぶラクになりました

なるほど。ウチもやるように言ってるんですが、なかなか・・・

自動販売機の制御ソフトウェア

様々な業務の請負開発

少しずつ異なる多数の機種がある。機種間で共通するテストがあり、いったん自動化すると他機種でも流用ができる。

A社システム部門長

業務間で流用できるテストがほとんどない。テストが共通化できるような更改案件もそれほど多くない。

B社事業部長

コンテキスト

苦手の対策に向けて

- 苦手意識はどこからくるのか？
  - 開発メンバの多くの賛同を得なければならない。
  - 単純に性に合わない。
  - 習得する知識やスキルが多く、時間がかかりそう。
  - 自身のコンテキストでは効果が出ないと直感的に感じている。
- 例）テスト自動化への苦手意識
  - 他のテストエンジニアも自動化しないと効果が出ないのか？
  - 自動化するテストケースの準備を事前にするのが億劫なのか？
  - ツールやフレームワークを使いこなすのに時間がかかりそうなのか？
  - テスト自動化によって省力化できなさそうな理由があるか？

苦手の対策に向けて

- コンテキストを意識すれば･･･
  - 単純な苦手意識であれば、克服できるよう手を施す。
  - よりつきつめて効果が本当に得られるのかを考えることができる。

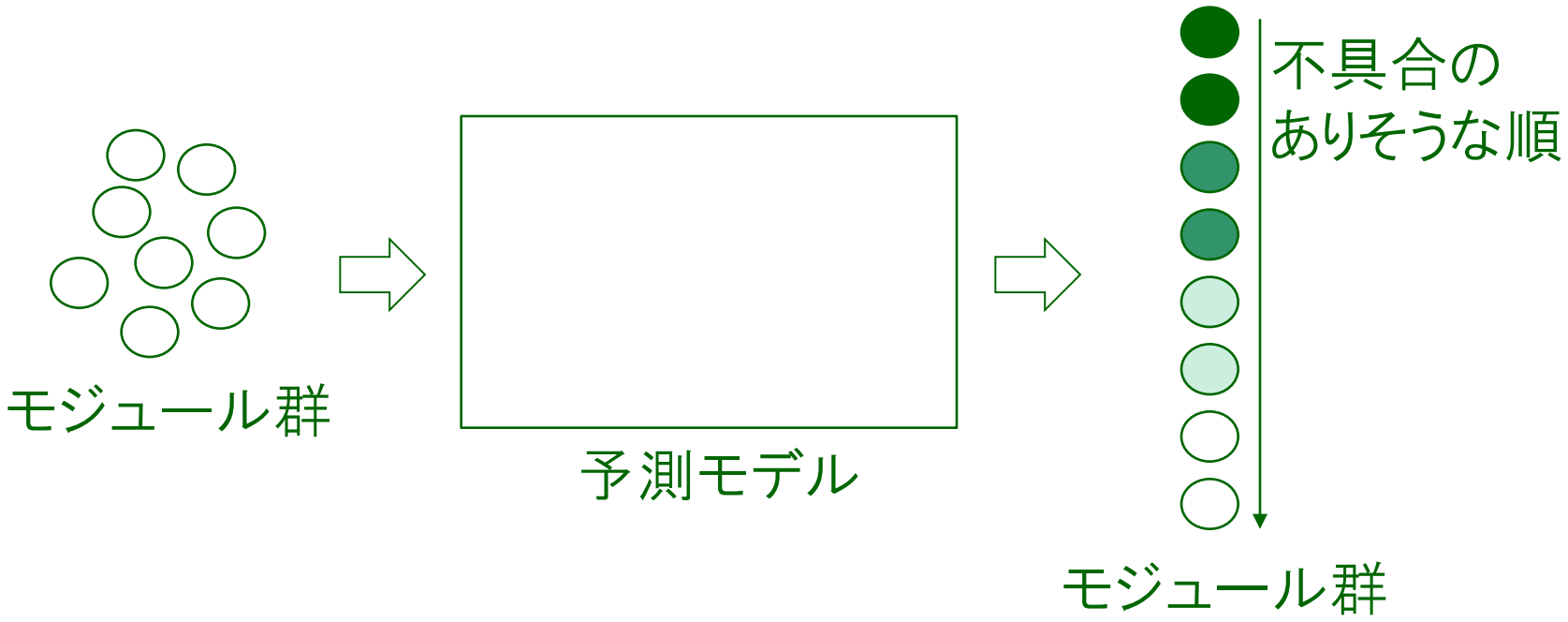
- 漠然と「あー、何か勉強せねば･･」という気分を持ち続けることが減る。
  - 苦手意識を克服できるような機会を待つ。
  - 効果が出ないと判断している場合には、効果がでる状況になるまでは着手しない。

アジェンダ

- コンテキストの違いによる効果の違い
- メトリクスを用いた手法とコンテキスト
- リファクタリングとコンテキスト
- コンテキストを明らかにするための着眼点

Fault-proneモジュール予測

- 不具合のありそうなソースコードモジュールを予測する手法（prone: ～の傾向がある）
  - モジュール: クラス、ソースコードファイル、・・・



Fault-proneモジュール予測

ールを予測する手法

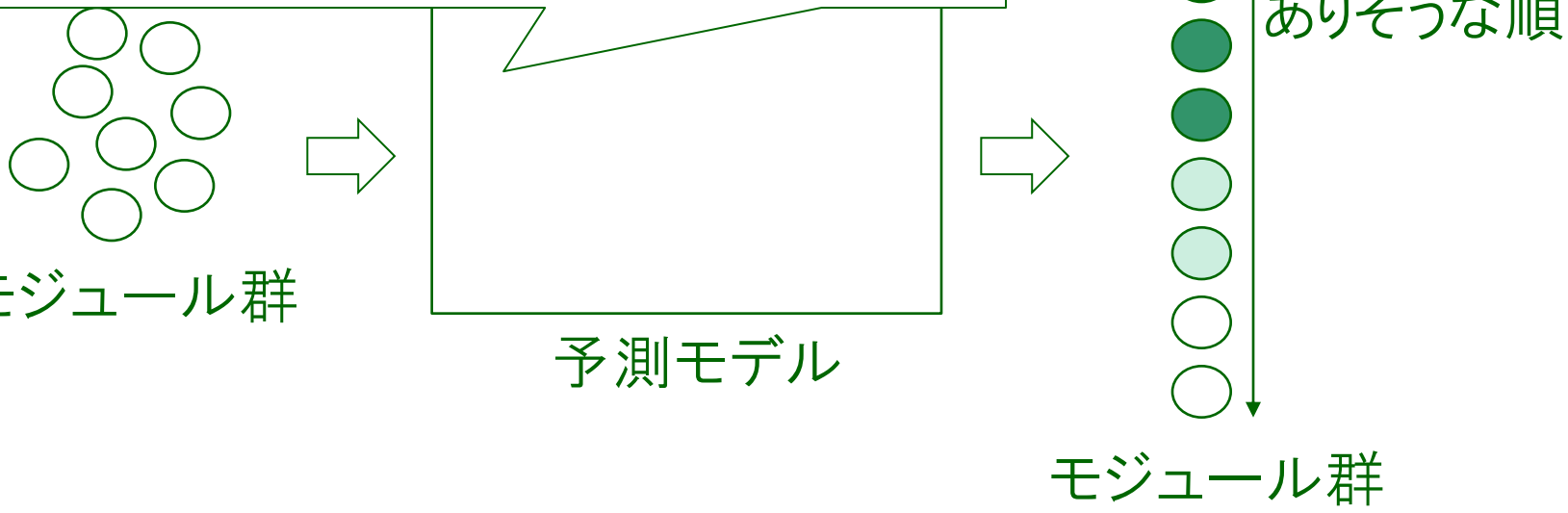
ル、・・・

プロダクトメトリクス

過去の不具合とソースコードメトリクスの関係をモデル化、経験則から予測

プロセスメトリクス

編集作業をはじめとした開発作業を計測し、経験則から予測



変更履歴と不具合分析: FixCache[*1]

- FixCache: 同じモジュールで不具合が再発しやすいという経験則を使って予測する。
  - 7つのオープンソースプロジェクトの変更履歴約200,000件で実証
  - 「メモリキャッシュ」のアルゴリズムを用いて不具合の可能性の高いモジュールを選出
  - 全体のソースコードファイルの1割で73～95%の精度で不具合を予測できた。
- Googleでのソースコード更新履歴に基づく不具合予測に応用されている。*2

*1: Kim S., Zimmermann T., James E. W., Zeller A., “Predicting Faults from Cached History” In Proceedings of the 29th international conference on Software Engineering , p. 489-498.

*2: Lewis C., Ou R. “Bug Prediction at Google” http://google-engtools.blogspot.jp/2011/12/bug-prediction-at-google.html

FixCacheの活用

- ご自身の開発にFixCacheを使うと効果が出そうですか？
  - 不具合のありそうなモジュールからテストすると余裕をもった修正ができます。
  - プログラマに「ここは要注意のモジュールです」と伝えることができます。

- 理由は？

リファクタリング

- コードリファクタリング（code refactoring: 継続的改善の一つ）
  - プログラムのふるまいを変えず、ソースコードの保守性、拡張性を高める。
  - 変更を繰り返したり、場当たり的なコード記述をすることでソースコードの見通しが悪くなり、リファクタリングが必要となる。

リファクタリング対象となるコードの例

```
aggregateTotal(a, b, c){
  total = a + b + c;
  shipping = 0;
  if (total < 3000) {
    shipping = 800;
  }
  total = total * 1.05;
  return total;
}
```

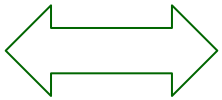
```
aggregateTotal_008(a, b, c){
  total = a + b + c;
  shipping = 0;
  if (total < 3000) {
    shipping = 800;
  }
  total = total * 1.08;
  return total;
}
```

リファクタリング

- コードリファクタリング（code refactoring: 継続的改善の一つ）
  - プログラムのふるまいを変えず、ソースコードの保守性、拡張性を高める。
  - 変更を繰り返したり、場当たり的なコード記述をすることでソースコードの見通しが悪くなり、リファクタリングが必要となる。

リファクタリング対象となるコードの例

```
aggregateTotal(a, b, c){
  total = a + b + c;
  shipping = 0;
  if (total < 3000) {
    shipping = 800;
  }
  total = total * 1.05;
  return total;
}
```



重複

```
aggregateTotal_008(a, b, c){
  total = a + b + c;
  shipping = 0;
  if (total < 3000) {
    shipping = 800;
  }
  total = total * 1.08;
  return total;
}
```

リファクタリングの実施

- ご自身の開発でリファクタリングを実施すると効果がありそうですか？
  - 将来の変更が簡単になったりバグが混入しにくくなります。
  - 将来的にデグレードが減る可能性があります。

- 理由は？

コンテキストを明らかにするための着眼点

- エンピリカルソフトウェア工学の論文*で紹介されている。
  - プロダクト
  - プロセス
  - プラクティス・ツール・技術
  - 開発に関わる関係者
  - 組織
  - 組織タイプ（マトリクス型組織、階層型組織など）
  - 市場

* K. Petersen and C. Wohlin, Context in Industrial Software Engineering Research, In Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09), pp. 401-404(2009)
ソフトウェア品質のホンネ 第114回「コンテキスト把握のための6つの分類」でも紹介しています。
http://www.juse-sqip.jp/wp3/honne/backnumber_114/

まとめ

- コンテキストの違いによるテスト自動化の効果の違いを紹介した。
- コンテキストを意識することで、単なる苦手意識か前提が揃っていないかが明らかになることを示した。
- コンテキストを意識した上で手法を実施すると効果があるのかどうか検討いただいた。
  - Fault-proneモジュール予測
  - リファクタリング
- 参考文献:
  奈良先端科学技術大学院大学:ソフトウェア開発におけるエンピリカルアプローチ、アスキー(2008)