派生開発の現場で秩序あるリファクタリングを実施する方法

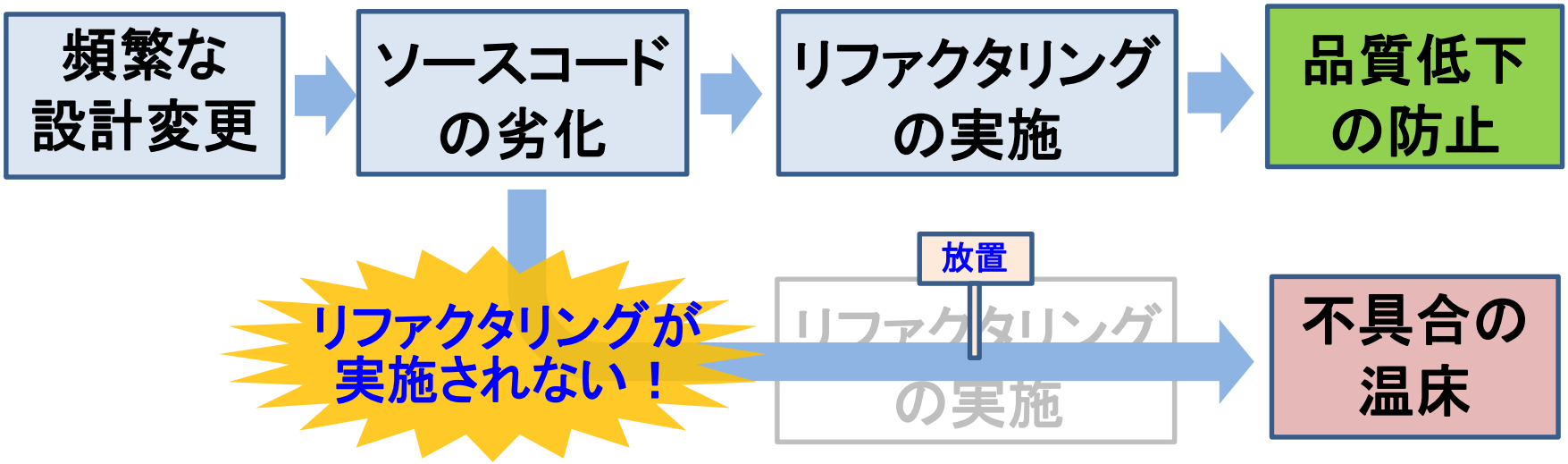
第６分科会Aグループ

リーダー： 菅原 誠一 （株式会社堀場エステック）
加川 俊哉 （株式会社アドバンテスト）
弦巻 智也 （テクニカルジャパン株式会社）
畑山 誠司 （アンリツエンジニアリング株式会社）
森江 里美 （アンリツエンジニアリング株式会社）

目　次

１. 研究の動機
２. 現状分析
３. 先行研究
４. 解決策
５. 解決策の考察
６. まとめと今後の課題

派生開発におけるリファクタリングの現状



【理由】

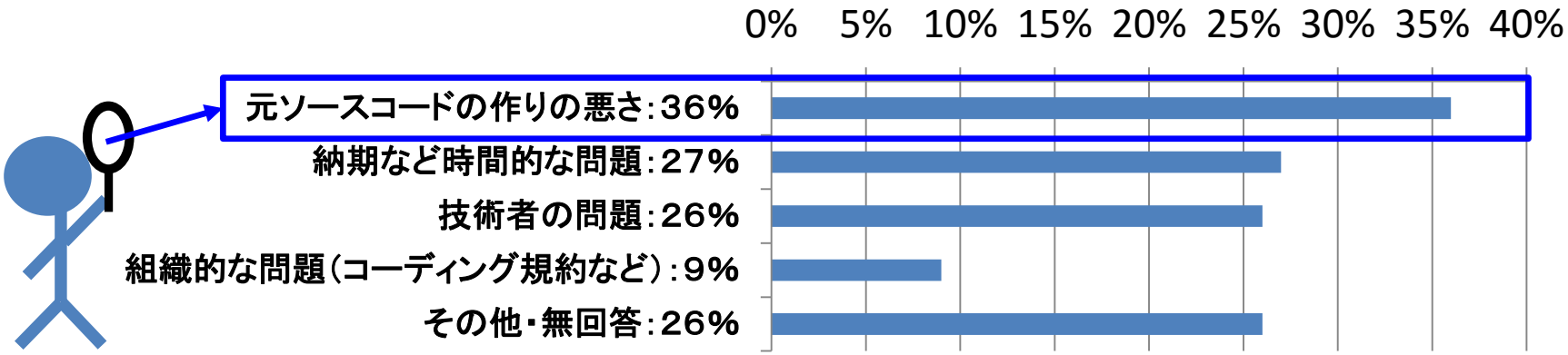
- デグレード・新たなバグを生むリスク
- 現状正しく動作しているのにやる必要があるのか
- 影響範囲の特定が面倒
- 予算・納期への影響がでるリスク

適切なリファクタリングの実施が必要！

■ソースコードを劣化させる要因

現場におけるリファクタリング意識調査(n=92名)を実施

⇒ 78%の人がソースコードの劣化を意識



ソースコード劣化要因として、
元のソースコードの作りの悪さに着目

理由:比較的容易に改善につなげることが可能！！

2.現状分析(2)

■リファクタリングに対する意識調査でわかったこと

- 90%の回答者がソースコードを改善したいと感じている
- 62%の回答者がリファクタリングを実施した経験がある
- リファクタリングを実施した回答者の8割が
  リファクタリングによる不具合発生を経験している

■リファクタリングにより発生した不具合の主な内訳

① 機能を変えずに構造を変えたつもりが、
機能も変わった:6件

② 関数内の機能を不要だと判断し、
削除した結果デグレードを起こした:7件

スキル不足などによる
危険なリファクタリング

リファクタリングついでの
機能削除

実際は回答者の半数以上がリファクタリングを実施
➡ほとんどがリファクタリングによる不具合を経験！

■リファクタリング実態のアンケート結果

不具合発生原因分析のため、先のアンケートと同じ92名に
「リファクタリングの実態」についてアンケートを実施

【アンケート結果から分かったこと】

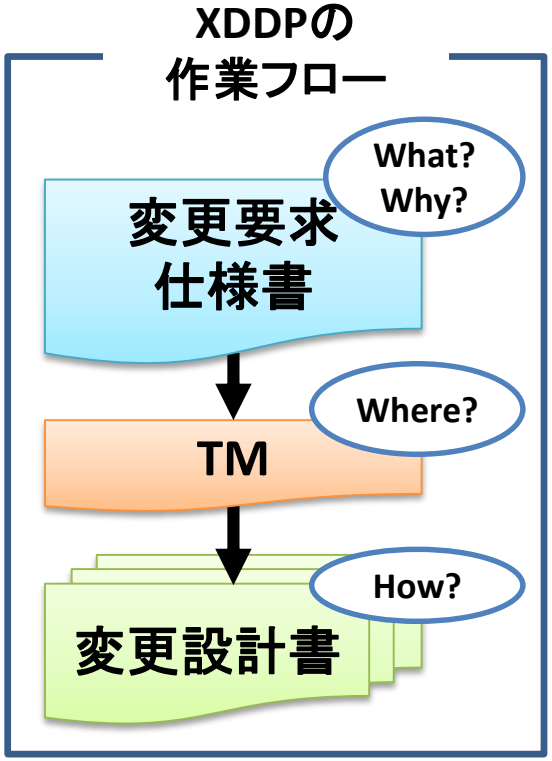
- 現場ではリファクタリング用プロセスが存在しないことが多い
- リファクタリングの実施判断や対象の決定などが担当者任せになっている

**リファクタリング用のプロセスは
存在しないのか？**

■リファクタリング用のプロセスの先行調査結果

① マーチンの『リファクタリング』
　プロセスとしての実施方法の記載はない

② XDDP
　小さなリファクタリングがあるが、限定的



現状、派生開発の現場において
リファクタリング用のプロセスがない

**リファクタリング用のプロセスがあれば
解決できるのでは？**

■リファクタリング専用の変更プロセス R-XDDP (Refactoring-XDDP) の提案

・XDDPの機能の変更のプロセスを応用

XDDPとR-XDDPの違い

XDDP

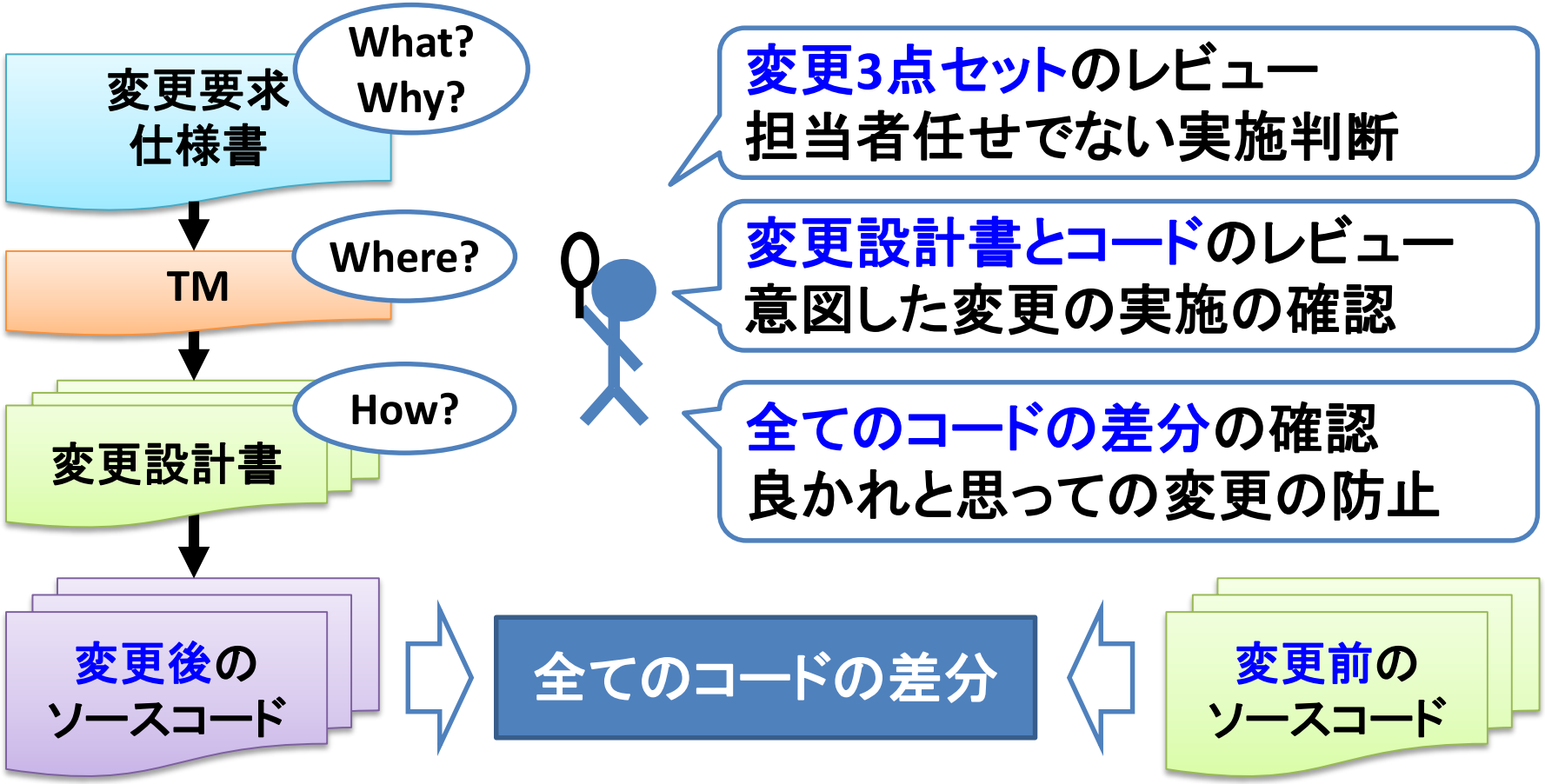
機能の変更プロセス
- 機能の変更
- 機能の削除
- バグ修正

R-XDDP

構造の変更プロセス
- リファクタリング

R-XDDPで、『機能の変更』を混同せずに『リファクタリング』を実現可能

■R-XDDDPでリファクタリングを実施することの効果



**R-XDDPの成果物をレビューすることで
リファクタリングのプロセスを管理できる**

■R-XDDPのリファクタリングの特徴

リファクタリング対象の特徴を、
リファクタリングパターンとして抽出

再利用性や凝集度
などの観点から抽出

リファクタリングのパターンの例
重複している処理を見つけ出し、「共通関数」としてまとめる
今後必要になりそうな再利用性の高い処理は、「共通関数」として抽出。
一時的凝集度の特徴を持つ初期化処理を行っている関数に対して、 関数の分割を行う
論理的凝集度の悪い特徴を持つswitch構造をなくし、 呼び出し側から処理を選択している引数をなくす

**リファクタリングパターンを限定することで、
担当者任せでない対象の選択が可能になる**

4.解決策(4)

■リファクタリングパターンを使ったリファクタリング(1)

変更要求仕様書の内容

要求:リファクタリングパターンを記述

変更要求・変更仕様			
要求	QUA01	手順的凝集度になっている関数を、内包する機能ごとに独立させる	
	理由	呼び出し側の便宜を図りすぎて不必要に処理が詰め込まれているため、今後、再利用される可能性のある関数を独立させておく	
	補足	機能ごとの分割をおこなう関数(QUA02-01の対象)と、その呼び元の関数(QUA02-02の対象)には同じ記号を付記し、繋がりを明記すること(#1,#2・・・で表現する)	
	□□□	QUA01-01	手順的凝集度の関数を内包する機能ごとに独立させる
	□□□	QUA01-02	当該関数を呼び出している上位関数では、独立させた関数の中で必要な関数を呼び出すように変更する

理由:リファクタリングする理由

変更仕様:構造のBefore/After
Before:変更前の構造
After:変更後の構造

要求としてリファクタリングパターン
変更仕様として構造のBefore/Afterを記載

4.解決策(5)

■リファクタリングパターンを使ったリファクタリング(2)

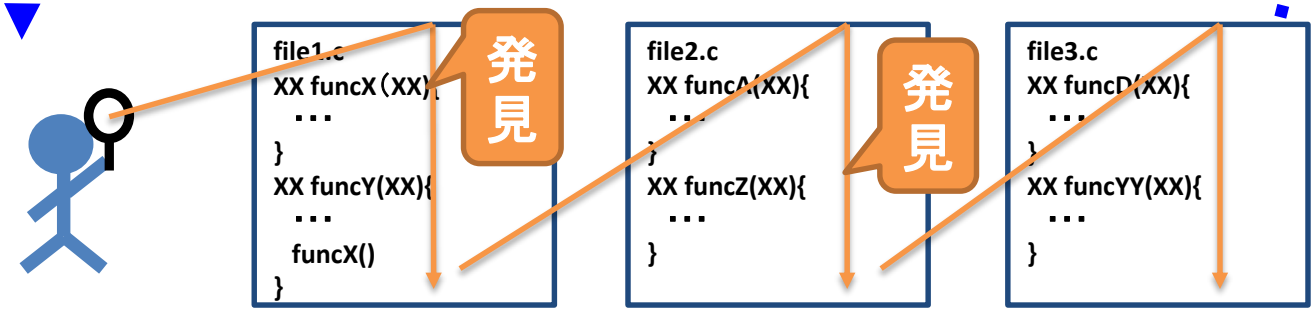
リファクタリング対象の検索時

TM

			変更要求・変更仕様	file1.c	file2.c	file3.c
要求	QUA01		手順的凝集度になっている関数を、内包する機能ごとに独立させる			
	理由		呼び出し側の便宜を図りすぎて不必要に処理が詰め込まれているため、今後、再利用される可能性のある関数を独立させておく			
	補足		機能ごとの分割をおこなう関数(QUA02-01の対象)と、その呼び元の関数(QUA02-02の対象)には同じ記号を付記し、繋がりを明記すること(#1,#2・・・で表現する)			
	□□□	QUA01-01	手順的凝集度の関数を内包する機能ごとに独立させる	(#1)funcX	(#2)funcZ	
	□□□	QUA01-02	当該関数を呼び出している上位関数では、独立させた関数の中で必要な関数を呼び出すように変更する	(#1)funcY (#2)funcB		(#2)funcE

①手順的凝集度の関数を探す

②TMに記録



リファクタリング対象の検索に専念できる

■リファクタリングパターンを使ったリファクタリング(3)

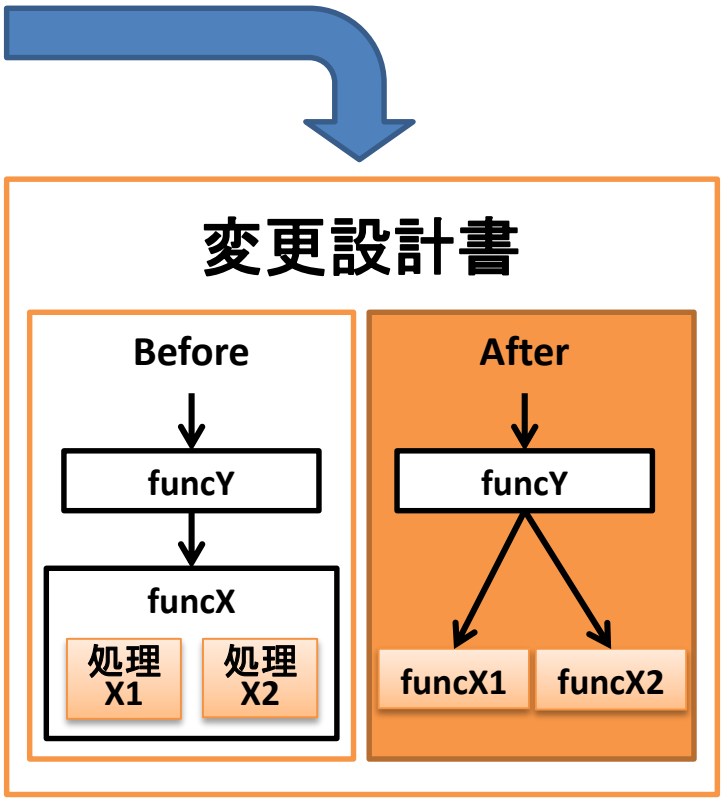
リファクタリング対象のレビュー時

TMに同じパターンの構造の変更が表れるので、レビューが容易

変更要求・変更仕様			file1.c	file2.c	file3.c
要求	…	…			
	□□□QUA01-01	手順的凝集度の関数を内包する機能ごとに独立させる	(#1)funcX	(#2)funcZ	
	□□□QUA01-02	当該関数を呼び出している上位関数では、独立させた関数の中で必要な関数を呼び出すように変更する	(#1)funcY (#2)funcB		(#2)funcE

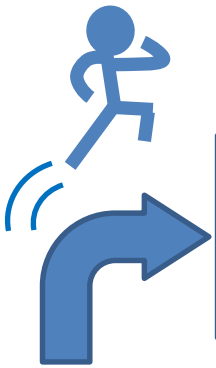
ソースコードの変更時

同じパターンの変更設計なので、ソースコードの変更が容易



同じパターンのレビュー及びソースコードの変更を行うため、作業が容易になる

■派生開発の現場への導入方法



3. R-XDDPで現場の課題を解決
- 不具合を未然に防ぐ、今後の作業の効率化

2. R-XDDPの導入
- 簡単なパターンに絞り、短期間で成果を出す

1. XDDPの小さなリファクタリング
- リファクタリングの基礎技術を習得

**今できることからはじめ、
段階的にリファクタリングを導入する**

担当者任せのリファクタリングで発生した不具合を防ぐことができるか

	不具合	R-XDDPの効果
1	スキル不足などによる危険なリファクタリング	レビューで担当者の理解不足等を把握＆判断することができる
2	リファクタリングついでの機能削除	全てのソースコードの差分の確認で発見＆阻止することができる

R-XDDPには、担当者任せのリファクタリングで発生した不具合を防ぐ効果が期待できる

■まとめ

- 担当者任せのリファクタリングを防止するための変更プロセス「R-XDDP」を提案
- 担当者任せのリファクタリングによる不具合の発生を防止する効果が期待できることを確認

■今後の課題

- 実プロジェクトにR-XDDPを適用し、効果を確認する
- リファクタリングパターンの種類を拡充し、R-XDDPの有効性の向上を図る

ご清聴ありがとうございました