XDDPの変更設計書から 間接リソース変化点を抽出する方法

2013年ソフトウェア品質管理研究会（29SQiP）
第6分科会 Aグループ

［研究員］
リーダー 小瀬 聡幸（日本プロセス株式会社）
衣斐 省伍（株式会社東京ビジネスソリューション）
井貝 智行（アンリツエンジニアリング株式会社）
杉山 幸雄（株式会社リンクレア）

目次

1. 研究の経緯
2. 現状分析
3. 解決策
4. 解決策の検証
5. まとめ

1．研究の経緯

派生開発の現場で
納期に追われる日々

ベースシステムが
大規模で全体を
把握しきれていない

研究員の解決したいことは？

インパクト大な記憶に残る不具合を無くして、
納期に追われる日々を終わらせたい！

2. 現状分析(1)

■ 不具合の分類

不具合を下表の4つに分類する。

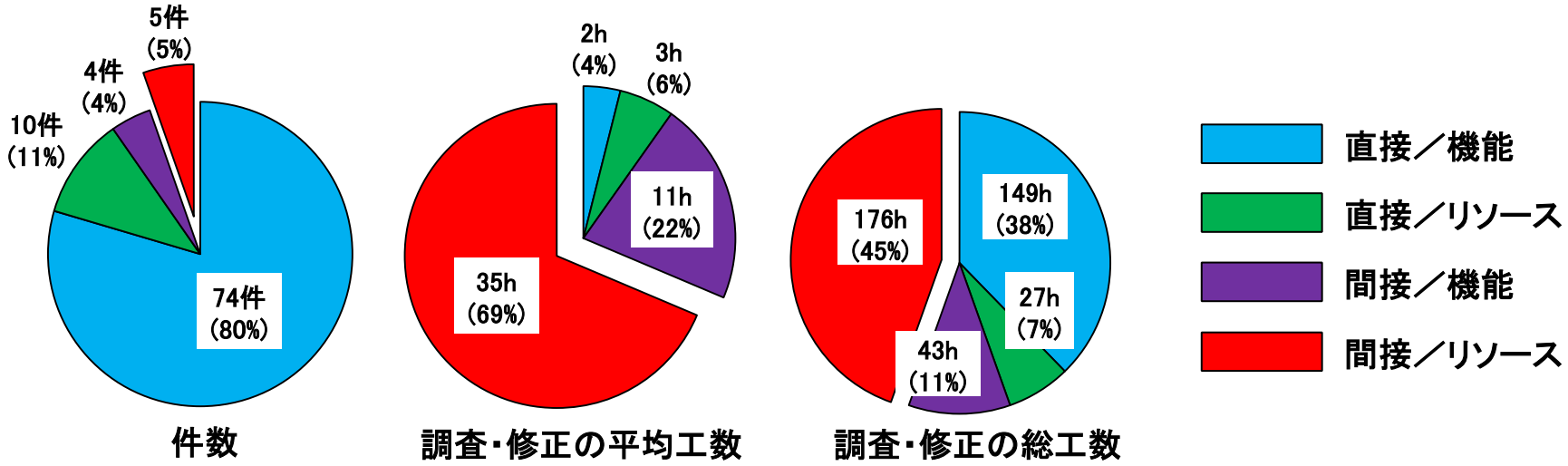
分類		変更要求	
		直接	間接
原因	機能	直接明示されている機能の変更漏れ	直接明示されている機能を変更したことで影響する別機能の変更漏れ
	リソース※	直接明示されている機能のリソース変更漏れ	直接明示されている機能を変更したことでリソースに影響を与えそれに伴い影響する別機能に対しての変更漏れ

※非機能要求はユーザー視点であいまいなので、
開発者視点で**捉えやすく**、発生原因が**見える化・定量化**できる「リソース」を分類の観点として採用。

2. 現状分析(2)

■ 不具合の分析結果

研究員の派生開発の現場で発生した93件の不具合を分析。



- 直接／機能の不具合は、発生件数が約80%と多いが、手戻り工数は、全体の約38%に過ぎない。
- **間接／リソースの不具合は、発生件数は5件と少ないが、手戻り工数は、全体の約45%を占める。**

間接リソースの不具合に着目する！

2. 現状分析(3)

■ 間接リソース不具合とは

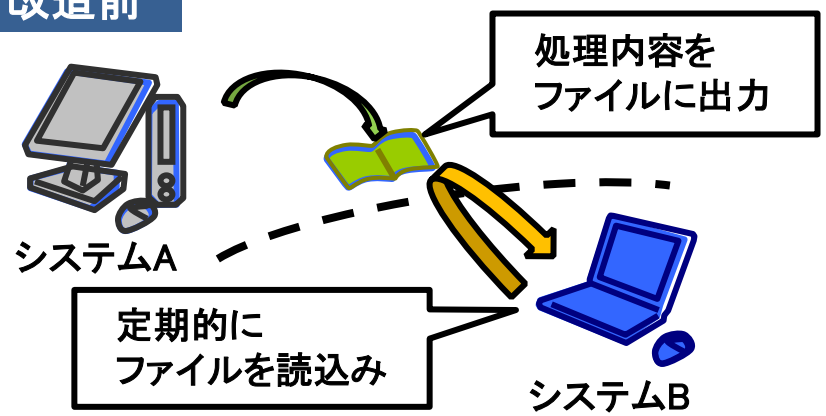
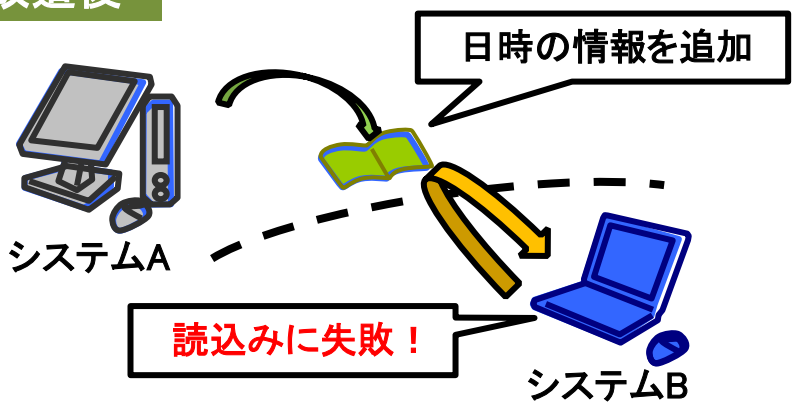
変更によって生まれた新たな**リソースの変化(※)**
が、ベースシステムの**限界値**を超えた時、
不具合となって現れる。

(※)は変更要求から直接紐付かない変化であり、
変更内容がコードレベル(ハンドルの取得など)で
明確にならなければ見えてこない変化である。
このリソースの変化を**「間接リソース変化点」**と定義した。

「間接リソース変化点」とその「限界値」を、
設計フェーズの中で抽出しなければならない！

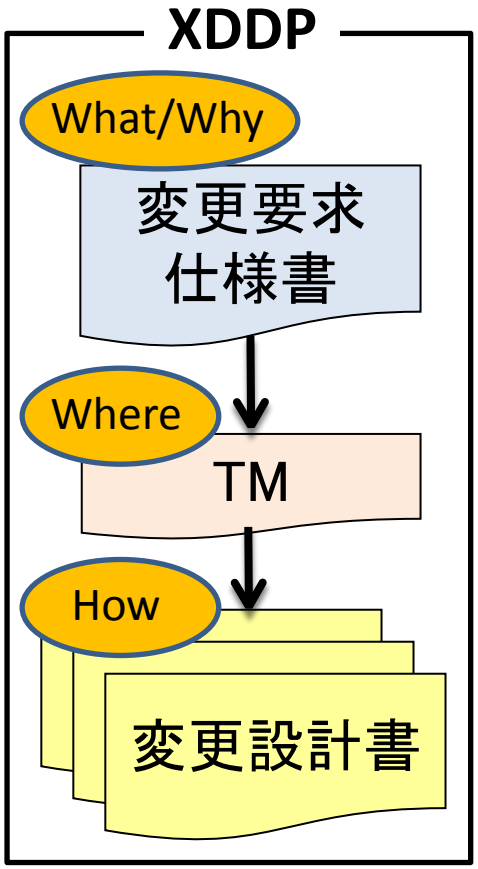
2. 現状分析(4)

■ 間接リソース不具合の例

変更要求	ファイルへ出力している情報に、日時の情報を追加してほしい。	
改造前		
改造後		
原因	システムA	ファイルへの出力開始～終了までの時間が、 9秒→12秒 に変化していた。 → 「間接リソース変化点」が発生！
	システムB	ファイルを 10秒周期 で読み込んでいた。 → システムBの「限界値」

3. 解決策(1)

■ 間接リソース変化点の抽出方法検討



変更要求仕様書・TMで対処できるか？

×変更内容の表現がソースコードレベルであるとは限らない。
→**抽出漏れ**の可能性がある
→**調査範囲が発散**してしまう可能性がある

変更設計書で対処できるか？

○変更内容の表現がソースコードレベルである。
→**確実に抽出**が可能である
→**調査範囲が明確**に絞られている
○リソースの変化をBefore/Afterで検証可能。

「変更設計書(間接リソース変化点付き)」を作成！

3. 解決策(2)

■ 抽出手順

変更設計書(間接リソース変化点付き)

■関数の変更

関数名	func_abc()	□変更 □変更 □変更

変更内容:

項目#	変更内容	予想行数	変化有無	
1	A処理とB処理の間にC関数の呼出しを追加する	3	有(13)	□

間接リソース変化点:

項目#	変化項目	変化内容	限界値	問題有無	
1	関数リターン時間	関数リターンまでの時間が約50msecから200msecになる。 C関数の処理時間が150msecであるため。	100msec	有	□

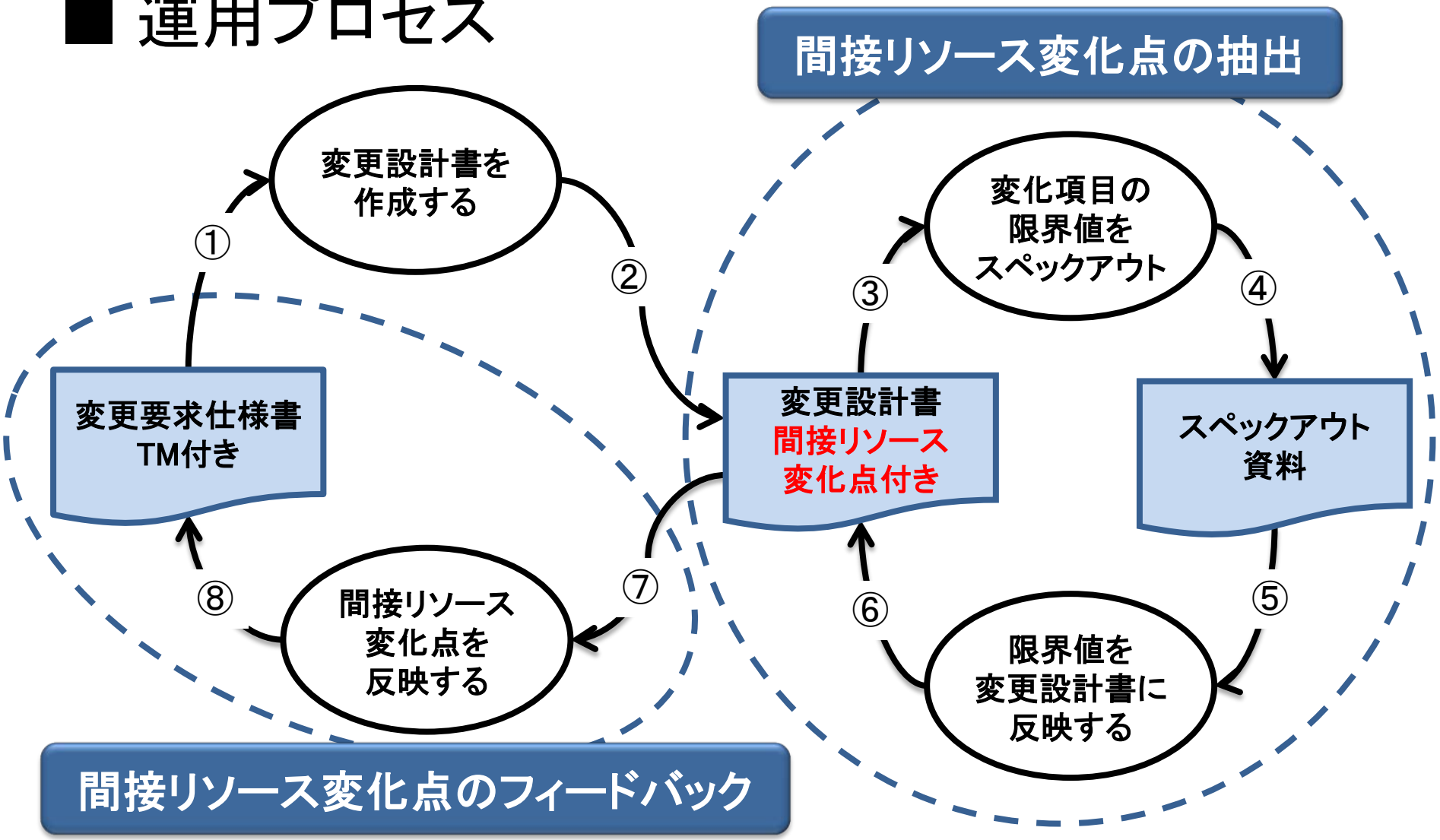
手順1:間接リソース変化点があるかないかを判断。

手順2:変化内容をBefore/Afterで記載。

手順3:ベースシステムの「限界値」を記入。

3. 解決策(3)

■ 運用プロセス



4. 解決策の検証(1)

■ 検証方法と結果

不具合事例を用いてシミュレーションを実施。以下2点を検証。

①間接リソース変化点を抽出できるか。

抽出に成功！

②解決策を適用したことで工数にどのような変化があったか。

	改善前			改善後				差分(h)
	設計	実装	不具合修正	設計	実装	追加設計	追加実装	
事例1	0.5	0.5	80.0	0.5	0.5	10.0	60.0	-10.0
事例2	0.5	0.0	24.0	0.5	0.0	8.0	8.0	-8.0
事例3	0.5	0.5	24.0	0.5	0.5	8.0	12.0	-4.0
事例4	0.5	0.5	8.0	0.5	0.5	4.0	4.0	0.0
事例5	1.0	0.5	40.0	1.0	12.0	12.0	10.0	-18.0

改善前よりも工数が削減(約20%減)できた！

4. 解決策の検証(2)

■ 検証の考察

検証を行って実証された効果

①**必要最低限の調査工数**で間接リソース不具合を防げる！

②不具合で対処することによる**コードの劣化**を防ぐことができる！

注意点

間接リソース変化点の有無の判断基準をプロジェクトやベースシステムにマッチさせることがポイント。

5. まとめ

研究の成果

インパクト大な記憶に残る不具合をなくすため、
「変更設計書(間接リソース変化点付き)」を提案した。

その結果

「間接リソース変化点」の抽出漏れを解消し、
最低限の工数で間接リソース不具合を防ぐ成果が得られた。

今後の課題

実際のプロジェクトにおいて設計段階から適用し、
解決策の有効性を検証する。

ご清聴ありがとうございました。

補足（1）

■ 変更要求仕様書/TMのフィードバック方法

	フィードバック方法	補足
1	新たな変更箇所としてTMに追加する	限界値のスペックアウトにより新たな変更箇所が明確になっている場合.
2	新たな変更仕様として変更要求仕様書に追加する	新たな変更箇所が多岐に渡る可能性がある場合
3	設計方針の再検討を行う	限界値のスペックアウトにより現状の設計方針では対処が困難なことが発覚した場合
4	変更要求を実現可能な内容に再調整を行う	限界値のスペックアウトによりシステムとして要求を満たすことが不可能であることが発覚した場合

補足(2)

■ リソース変化点チェックシート

No	分類	チェック項目
1	関数I／F	引数のパラメータサイズに変化はないか
2		返値の型にサイズ変化はないか
3	変数・配列	型のサイズに変化はないか
4		配列のサイズに変化はないか
5		外部変数，static変数を新規追加しているか
6	メモリ	確保するメモリサイズに変化はないか
7		同時に確保されるメモリサイズに変化はないか
8		メモリへの書込みサイズに変化はないか
9		ハンドルの生成数に変化はないか
10		同時に生成されるハンドルの数に変化はないか
11		スタックの消費に変化はないか
12		メモリマップが変更となる変化はないか
13	処理	関数の処理時間(returnするまでの時間)に変化はないか
14		再帰関数，循環関数が追加されていないか
15		処理回数に変化はないか
16		while文，for文のループ回数に変化はないか
17		応答・通知を返却する時間に変化はないか
18		同期，非同期処理に変更はないか
19	データ	扱うデータサイズに変更はないか