第 6 分科会 A グループ

XDDP の変更設計書から間接リソース変化点を抽出する手法

The method of extracting indirect resources change point from program specification of XDDP

主査　　：清水　吉男（株式会社システムクリエイツ）
副主査　　：飯泉　紀子（株式会社日立ハイテクノロジーズ）
　　足立　久美（株式会社デンソー）
リーダー　：小瀬　聡幸（日本プロセス株式会社）
研究員　　：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）

研究概要

派生開発の現場ではベースシステムが大規模で全体を把握しきれない，設計書が存在しない，などの事情によりデグレードが後を絶たず悩まされている．そこで，不具合事例を分析したところ，特に調査・修正に時間がかかり現場を悩ませている不具合の原因は変更要求で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であり,かつ,リソースに関わる変更漏れであることに気づいた．我々はその不具合の要因を「間接リソース変化点」と定義し，設計段階で抽出するべく「変更設計書(間接リソース変化点付き)」を作成した．これを実際に発生した不具合事例に対して適用しシミュレーションを行った結果，改良した変更設計書を使用することで間接リソース変化点を抽出することができ，変更漏れを防ぐ効果が得られた．

Abstract

It suffers from degraded does not have end to the situation-based system is not completely understand the entire large-scale, design document is such, does not exist in the field of development derived. So, Analysis of the failure case is a change leak another features that affect because you changed the features that are clearly directly in the change request is the cause of the defect, which has plagued the field is time-consuming research in particular, to the modification, and you notice that it is a change leakage related to resources. We defined as "indirect resources change point" the cause of the failure, you have created a resource indirect change point with change design document in order to extract at the design stage. Results of simulation was applied to the problem cases have actually occurred this effect it is possible to extract the indirect resource change point by using the change design document that was modified to prevent changes leakage was obtained.

1.　はじめに

近年のソフトウェア開発は，ベースシステムの部分的な変更や，ベースシステムに新しい機能を追加するといった派生開発が主流となっている．しかし，派生開発の現場ではベースシステムが大規模で全体を把握しきれない，設計書が存在しない，などの事情によりデグレードが後を絶たない．デグレードの中には変更した機能とは一見，直接関係のない機能で不具合現象が発生するものもあるため，原因の特定に多くの時間を要することもあり悩まされている．デグレードの回避策として一般的にはリグレッションテストの実施が有効であるとされているが，短納期・低コストを求められる派生開発の現場では全機能を

網羅するような膨大な量のテストは実施困難である．

そこで我々は，担当している派生開発の現場で発生した不具合事例について調査を行った．その結果，発生件数は少ないものの調査・修正の工数が多い不具合が存在し，プロジェクトに大きな影響を与えているということがわかった．このような不具合の要因は，変更要求で直接明示されている機能を変更したことで時間や領域といったリソースが変化し，ベースシステムの時間，領域の限界値を超えてしまうことであった．そこで，我々はそのリソースの変化を「間接リソース変化点」と定義した．

現状，間接リソース変化点の抽出と，ベースシステムの限界値を超えるかどうかを判断するには，有識者がいなかったり，ドキュメント不足であったりするため困難な状況にあることが多い．

我々は解決策として，派生開発の現場で有効とされる XDDP（eXtreme Derivative Development Process）[1]に着目した．その成果物（変更要求仕様書，TM，変更設計書）の中でも具体的な変更箇所が明確になる変更設計書を記載するプロセスで間接リソース変化点を抽出することが効率的であると考え，「変更設計書(間接リソース変化点付き)」を作成した．これにより設計者は以下の判断が行えるようになる．

・変更箇所により生まれる間接リソース変化点があるかないか

・間接リソース変化点がベースシステムの限界値を超えていないか

これを適用することにより，間接リソース変化点が抽出可能となり，変更漏れを防ぐ効果が得られた．本論では「変更設計書(間接リソース変化点付き)」の適用結果について述べる．

2.　現状分析

2．1.　不具合事例の分析

我々は，担当している派生開発の現場で発生した不具合事例(全 93 件)について調査を行った（付録 A「付表 1：分析を行った不具合事例(一部抜粋)」参照）．これらの不具合事例を「変更要求」「変更内容」「不具合内容と原因」について分析した結果，派生開発にて発生する不具合は，大きく 4 つに分類されると考えた（表 1）．

表 1．派生開発にて発生する不具合の分類

		変更要求	
		直接	間接
原因	機能	【(A) 直接機能不具合】 変更要求で明示された機能の変更漏れ	【(C) 間接機能不具合】 変更要求で明示された機能を変更したことで影響する別機能に対しての変更漏れ
	リソース	【(B) 直接リソース不具合】 変更要求で明示された機能のリソース変更漏れ	【(D) 間接リソース不具合】 変更要求で明示された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

不具合には変更要求で直接明示されている機能の変更漏れ（直接）と，変更要求で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（間接）がある．また，原因としては，機能そのものに対する変更漏れと，リソースに対する変更漏れがある．通常，不具合は機能と非機能（品質）に分類される．しかし我々は，ユーザー視点であいまいな言葉で語られがちな非機能ではなく，開発者視点で捉えやすいシステム上のリソース（資源）として分類した．これにより不具合の原因を処理時間，データサイズ，画面スペ

ースなどのリソース上の問題として扱うことができ，発生原因を見える化，定量化することが可能となる．

4 つに分類した 93 件の不具合事例を以下 3 つの観点でグラフ化した．

①件数（図 1）

②調査・修正の平均工数（図 2）

③調査・修正の総工数（①件数×②平均工数）（図 3）

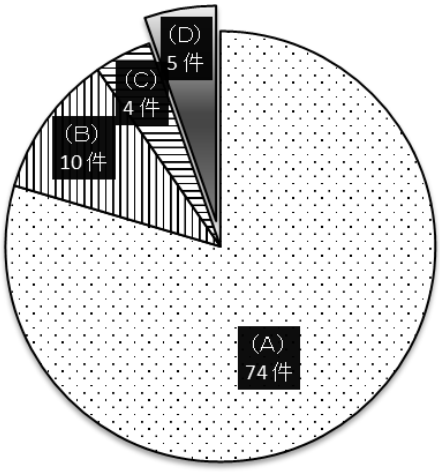


図 1．件数

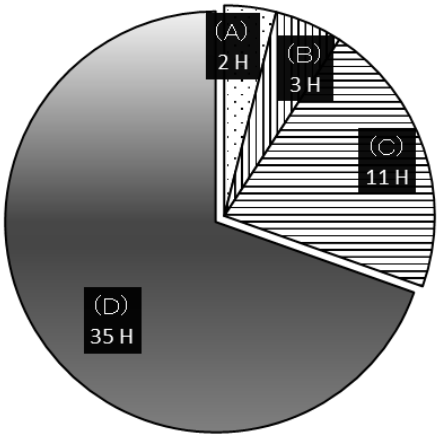
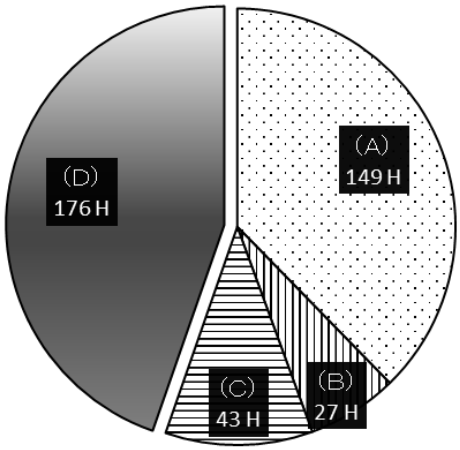


図 2．調査・修正の平均工数



（A）直接機能不具合
（B）直接リソース不具合
（C）間接機能不具合
（D）間接リソース不具合

図 3．調査・修正の総工数

この分析結果から，「(D) 間接リソース不具合」は発生した件数としては少ない．しかし，調査・修正の平均工数は一番多く，総工数としても一番多くなっている．このことから，「(D) 間接リソース不具合」はプロジェクトに与える影響が大きい不具合であることがわかる．

上記の結果から，「(D) 間接リソース不具合」を解決することで，プロジェクトに与える影響を大きく軽減できると考えた．

2．2．　間接リソース変化点

間接リソース不具合の要因は，変更要求で明示された機能を変更したことで影響を受けたリソースの変化がシステムの限界値を超えたからである．我々は，そ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は変更要求に直接明示されな

いため，変更箇所の具体的な変更内容（処理回数やデータサイズを XX から YY に変更するなど）が明確になるまでは見えてこないリソースの変化である．

収集した間接リソース不具合の要因を分析したところ，機能を追加したことにより処理時間，処理回数が変化する「時間系リソース」，メモリ領域，画面領域，スタック領域が変化する「領域系リソース」という二つの共通項があることに気づいた(表 2)．

表 2．間接リソース変化点の分類

	間接リソース変化点	
	時間系リソース	領域系リソース
要因	・処理時間の変化 ・処理回数の変化	・メモリ領域の変化 ・画面領域の変化 ・スタック領域の変化

例えば時間系リソースの変化を特定できなかったことで発生した不具合に，「原因は不適合装置　ＪＲ九州のシステム障害」[2]がある．本障害の原因は，ストレージを HDD から SSD へ変更したことにあった．その結果，ストレージのリセット処理時間が 0.3 秒に変化していた．これがシステム上の限界値 0.2 秒を超えたいたため，システムの基盤交換時に行ったリセット処理で処理が完了せずエラーとなった．通常の動作においては影響がなかったため，2010 年 7 月の交換から 2013 年 7 月の障害発生まで正常に稼働していたが，障害発生時は始発から 3 時間列車を運行できず約 7 万 9000 人に影響を与えてしまった．

このような不具合が起きる要因は間接リソース変化点がベースシステムの限界値を超えてしまうことである．限界値を超えてしまう要因としては，単純に変更要求の内容によって超える場合もあるが，以下のような要因も考えられる．

- 機能追加を繰り返すことでリソースを徐々に消費し，あるタイミングで限界値を超えてしまう
- 複数チームでの開発時に各チームがリソースをそれぞれ消費し，結合した際に限界値を超えてしまう
- 引き継ぎの際，機能に関する情報は引き継がれるが，リソースに関する情報が引き継がれず，設計者がベースシステムのリソースの限界を意識せずに変更を行ってしまう

派生開発の現場では，ベースシステムへの機能追加が多く行われるが，既存機能の削除やベースシステムのリプレイスといったリソースの見直しが行われることは少ないため，間接リソース変化点が限界値を超え易い環境にある．

2．3．　先行研究

以上で挙げた間接リソース変化点に起因する問題を解決するために，開発プロセスである XDDP[1]の技術や先行研究[3][4][5]が参考にならないか調査した．

XDDP では機能変更の開発プロセスにおいて 3 点セット（変更要求仕様書，トレーザビリティーマトリクス（TM），変更設計書）を使用することで担当者の思い込みや勘違いを低減し，影響範囲を網羅的に特定する方法であるが，間接リソース変化点の検知はそれ自体が顧客要求としてあげられない限り変更仕様として明記されず気づきにくい．

また先行研究では，仕様・変更漏れ，他システムへの連携漏れ，デグレードに対して防止する技術を挙げているが,いずれも XDDP の変更要求仕様書の項目をベースに対処しているため，変更箇所をどう変更するのか具体的な内容が明確になるまで抽出が困難な間接リソース変化点に対しては対処できていない．また，XDDP や先行研究で提示されている方法で変更要求仕様書から間接リソース変化点を抽出すること自体は可能ではあるが，派生開発の現場ではベースシステムが大規模で全体を把握しきれない，設計書が存在しない等の

理由により，調査項目が多岐に渡り調査範囲が発散してしまうため多大な工数を要する可能性がある．

3.　解決策

派生開発におけるすべての不具合は変更箇所から発生したものである．XDDP ではすべての変更箇所が変更設計書に書かかれる．よって間接リソース変化点も変更設計書に書かれた変更箇所の内容から抽出することができると考えた．

XDDP の変更設計書は設計意図を検証し易くすることを目的として変更箇所を変更前/変更後（Before/After）の形式で記載する．この記載形式の特徴は間接リソース変化点の抽出に都合がよい．変更箇所が Before/After で記載されていることにより消費されるリソースの増減も Before/After で検証することができる．

そこで我々は変更設計書に間接リソース変化点を抽出するための項目を規定した（付録B「変更設計書（間接リソース変化点付き）」参照）．これにより設計者は間接リソース変化点の有無と，それがベースシステムの限界値を超えていないかを判断しながら設計を進めることができ，間接リソース不具合の発生を防ぐことが可能となる．また，変更設計書の中で間接リソース変化点を抽出することにより，スペックアウト（ソースコードを中心とした現状システムの調査・分析）する項目・範囲を必要最小限に絞り工数を抑えることが可能となる．

3．1.　間接リソース変化点の抽出

図 4 に「変更設計書(間接リソース変化点付き)」を用いた運用手順を示す．

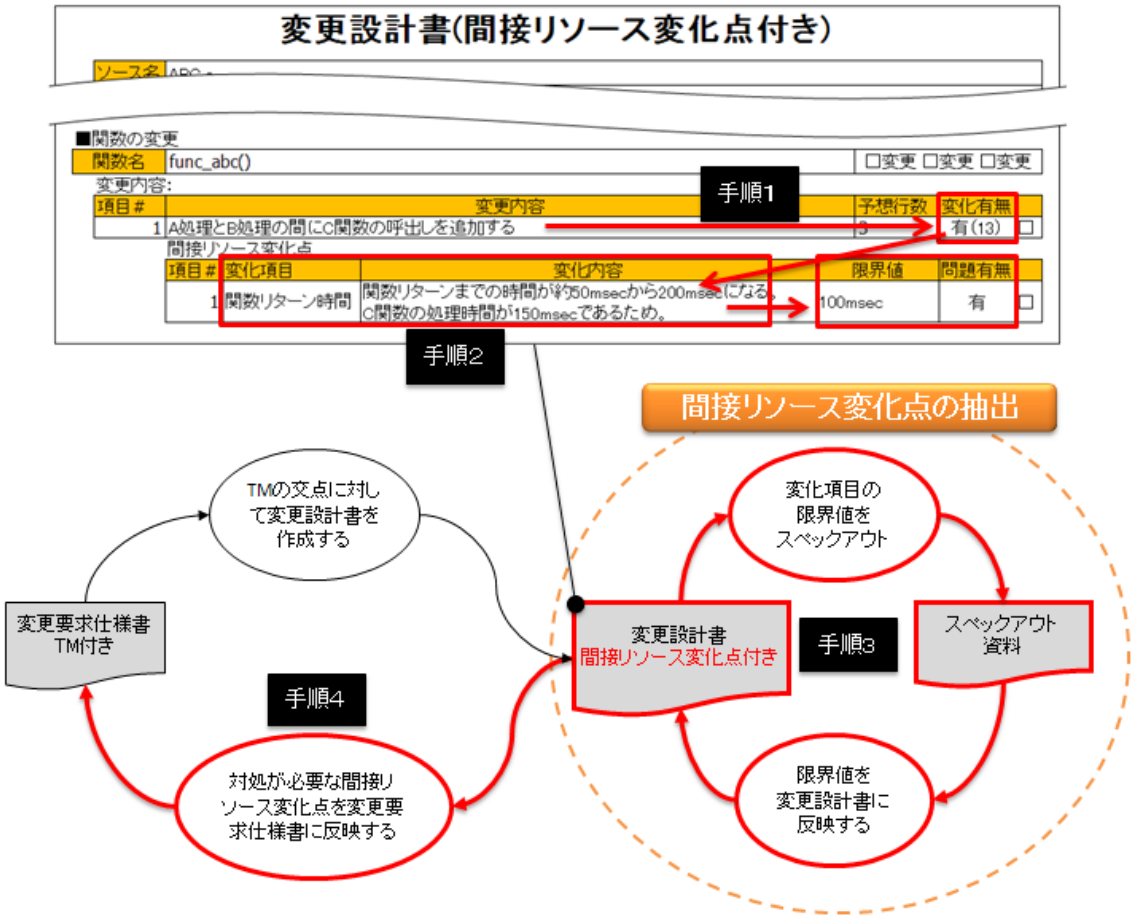


図 4.「変更設計書(間接リソース変化点付き)」を用いた運用プロセス

手順１．

変更設計書に記載された変更箇所の Before/After に着目し，変更箇所から生まれる間接リソース変化点があるかないかを判断し「変化有無」欄に有無を記入する．判断にはベースシステムで扱われているリソースを一覧にしたチェックシートを用意し（付録 C「リソース変化点チェックシート」参照），該当したチェック項目の No も合わせて記入する．変更内容によっては複数のチェック項目に該当する場合もある．リソースには 2．2 章で挙げたように時間系リソースと領域系リソースとがあるが，システムによって傾向が異なるためプロジェクトに合わせたチェック項目を用意する．また，プロジェクトが進むにつれ追加される観点もあるため随時ブラッシュアップが必要である．

手順２．

手順１で間接リソース変化点があると判断された場合，その内容（処理時間，メモリサイズ等）を「変化項目」欄に記入する．さらにその変化する値を「変化内容」欄に Before/After の形で記載する．After の値が正確にわからない場合は想定される値を根拠と伴に記入する．Before/After で記載することで，その変化項目の影響度の検証が可能となる．

手順３．

手順２で挙げた変化項目に関するベースシステムの「限界値」を記入する．限界値が不明な場合にはベースシステムのスペックアウトを行う．これにより明確になった間接リソース変化点の変化項目とその Before/After 値，さらにベースシステムの限界値を比較し問題の有無を判定する．

手順４．

最後に，手順３で問題有となった間接リソース変化点の変化項目を表 3 のいずれかの方法で変更要求仕様書/TM にフィードバックする．

表 3．間接リソース変化点の変更要求仕様書/TM へのフィードバック

	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

4．　解決策の検証

「変更設計書(間接リソース変化点付き)」を用いることで，間接リソース変化点を抽出できるか検証を行った．本章ではこの検証結果について述べる．

4．1．　検証内容

「変更設計書(間接リソース変化点付き)」が有効であるか判断するため，次の 2 点について検証した．

（１） 間接リソース変化点を抽出できるか

（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

ここで，（２）を検証内容に含めたのは，目的である（１）が達成されたとしても，間接リ

ソース変化点を抽出するためにかけた追加工数が不具合修正工数を上回ってしまうと，短納期,低コストを要求される派生開発プロジェクトでは運用しにくいと考えたためである.

検証方法は，メンバから収集した不具合の中で,「2. 1 不具合事例の分析」において「(D)間接リソース不具合」に分類した事例 5 件を用いてシミュレーションし,結果を分析した.シミュレーションは，事例で上げた不具合が発生したきっかけとなった変更要求に対し，「変更設計書(間接リソース変化点付き)」を用いて変更設計書を作成し，表 3 のフィードバックまでを実施するという方法で行った. 対象とした不具合事例の詳細は, 付録 A の「付表 1：分析を行った不具合事例(一部抜粋)」を参照.

4. 2. 検証結果

シミュレーションを実施し，作成した変更設計書の一例を図 5 に示す. 以下のように，単なる定義値の変更という変更内容から，変化有無を調査し，結果を間接リソース変化点欄に記載することで問題有無が一目で確認できるようになったことがわかる.

■関数外の変更

項目#	変更内容	予想行数	変化有無
1	設定データ編集バッファのサイズの定義値を, 64*102*1024 から 256*1024*1024 に変更する	1	有(6)

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	メモリ	メモリ使用量が 64MB*2 から 256MB*2 に増加する	512MB	有

図 5. 事例 1 のシミュレーション結果

（1） 間接リソース変化点を抽出できるか

事例 1 の場合，バッファサイズの定義値を変更することで，ソフトウェアが使用するメモリ量（領域系リソース）が増加し，システムの限界値である 512MB に到達してしまうことが判明した. このままでは動作に支障をきたすことが予想されるため，問題有無を“有”と判定し，解決策として，不要なメモリ使用量の削減やバッファの確保の仕方を変更するという新たな変更仕様を追加し対応するようにした. 他の事例に関しても同様の方法でシミュレーションを行った結果，5 件すべてで間接リソース変化点を抽出することができた.

（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

今回のシミュレーションにおいて,「変更設計書(間接リソース変化点付き)」を使用しなかった改善前の工数と，使用した改善後の工数の比較を**表 4** に示す.

表 4. 改善前後の工数比較（単位：H)

	改善前			改善後				差分
	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

表中の追加設計とは，解決策で示されている手順 1 ～ 4 までを実施した工数，追加実装とは，追加設計で必要となった新たな実装工数を示している.

この結果から，事例 4 が差分 0 となっているが，全体を通して改善前より改善後の工数が減少する傾向にあることがわかった．

4．3．　考察

検証の結果,「変更設計書(間接リソース変化点付き)」を使用することで間接リソース変化点を抽出することができ，期待した間接リソース変化点の影響による不具合を削減するという効果が得られた．このことにより，設計工程で間接リソース変化点の設計漏れを防ぐことができ，開発終盤や出荷後の時間に余裕が無い状態で，不具合修正を強いられることは減少する．これは，時間に余裕が無い状況でのその場しのぎの変更によるソースコードの劣化を防ぐという効果も期待できる．また，設計工程で対応規模を見積ることで，実装工程における日程や人員計画を見直すこともできる．

課題として，間接リソース変化点の有無や影響を判断するための調査工数が膨らむ可能性がある．それは「リソース変化点チェックシート」において何をチェック項目として扱うかがポイントになり，いかにチェック項目をプロジェクトやベースシステムの傾向にマッチさせることができるかが重要である．さらに，本手法を繰り返し適用し調査結果を蓄積することで確実に改善につなげる効果が期待される．

5．　本研究のまとめ

5．1．　研究成果

これまで我々は，変更した機能と直接関係のない機能で発生する不具合に悩まされていた．不具合の原因を調査し，特徴を調べたところ間接リソース不具合がプロジェクトに与える影響が大きいということに気づいた．その不具合の要因を間接リソース変化点と定義し，変更設計書から間接リソース変化点を抽出する方法を考案した．検証の結果，本手法を適用することで以下の効果があることがわかった．

- 間接リソース変化点を抽出することができ，変更漏れを防げる．
- 不具合となってから調査・修正するよりも工数を削減することができる．

5．2．　今後の進め方

私たちの考案した間接リソース変化点の抽出方法は，実際に発生した不具合事例に対するシミュレーションにより効果があることがわかった．今後は，実際のプロジェクトで運用し有効性を確かめたい．また，さらに調査工数をいかに削減できるかに着目し,「変更設計書(間接リソース変化点付き)」を改良していきたい．

6．　参考文献

[1] 清水 吉男，“「派生開発」を成功させるプロセス改善の技術と極意”，2007 年

[2] 日本経済新聞，“原因は不適合装置　ＪＲ九州のシステム障害”，［http://www.nikkei.com/article/DGXNASJC2202V_S3A720C1ACY000/］，最終検索日 2014 年 1 月 28 日

[3] 津田 剛宏，田中 聡，古畑 慶次，“設計手法を活用した変更要求仕様書の作成手法”，2010 年

[4] 木下 良介，中澤 康郎，大杉 仁司，“変更依頼の対応箇所を検討する前に他システムへの影響を検知する方法”， 2011 年

[5] 中井 栄次，間瀬 研二，古畑 慶次，“無知見プロジェクトに対する XDDP の適用”，2009 年

付録 A

付表 1：分析を行った不具合事例(一部抜粋)

No	変更要求	Before	after	不具合	原因	不具合分類	間接リソース変化点
1	設定できるデータサイズを256MB に拡張してほしい	最大データサイズ 64MB	最大データサイズ 256MB	一番負荷がかかる環境で起動すると，アプリが起動するのに 25 分かかる(通常 30 秒程度)	機器のメモリサイズが 512MByte しかなく，アプリが使用できるメモリサイズの上限を超えた．そのため，起動時に HDD へのスワップ処理が頻繁に発生しパフォーマンスが著しく低下した	(D) 間接リソース	領域
2	フラッシュメモリを後継機種に変更してほしい	フラッシュメモリは既存機種	フラッシュメモリを後継機種に変更	フラッシュへの書込みがたまに失敗する	ブロックイレースにかかる時間の最大値が，前機種より遅くなることがあったため，想定している処理時間を超えてしまい，ソフトがエラーと判断し，失敗扱いにしていた	(D) 間接リソース	時間
3	測定器と制御ソフトで組まれたシステムのうち，測定器を現行モデルに変更するため，制御ソフトも合わせて変更してほしい	置き換え前の機種の結果取得コマンドは〇〇〇	現行モデルの結果取得コマンドに△△△変更する	測定停止を押しても，止まるのに時間がかかる	タイマー周期で結果を取得する処理が現行モデルのほうが遅く，周期より時間がかかっていた．そのため，周期タイマーの結果取得イベントがたまっており，その最後尾に停止イベントが追加されるため，たまっている取得イベントの処理をすべて終えないと，停止イベントの処理が実行されなかった	(D) 間接リソース	時間
4	測定器と制御ソフトで組まれたシステムのうち，測定器を現行モデルに変更するため，制御ソフトも合わせて変更してほしい	置き換え前の機種データ転送コマンドの引数を変更する	制御対象機器を，現行モデルに変更	データ転送の量が大きいと失敗することがある．データサイズが小さいと問題ない	通信タイムアウトが 3sec になっていたが，データ転送に最大 4sec かかることがあったため	(D) 間接リソース	時間

5	DDR メモリにアクセスする前に Wait を入れてほしい	Wait なし	DDR メモリにアクセスする際は，200msec の Wait を追加する	手動で設定を次々と変えていると，アプリ-ファーム間通信が切断される	Wait 中に設定中断扱いになる場合が発生した．しかし，その場合を想定した作りになっておらず，想定していない処理を呼び出してしまい，グローバル領域を破壊していた．破壊した領域に通信関係の変数がマッピングされており，その変数の値を書き換えてしまうことで，以後通信不可能になっていた	(D) 間接リソース	時間
6	既存機種(型名：〇〇A)の機能追加版(型名：〇〇B)を追加したい	〇〇A	型名判定に〇〇B を追加	ファームウェアが起動しなくなった	電源投入時は起動用の機能限定ファームウェア（改造予定無）で起動し，アプリ起動時に本番用のファームウェア（型名追加に対応）に切り替えている． 問題となったのは機能限定ファームで，起動時に型名で判断している箇所があり，知らない型名のため処理を中断してしまっていた	(C) 間接機能	---
7	画面にファイルを添付する機能を追加したい	ファイル添付機能なし	ファイル添付機能追加	添付ファイルが一部の担当者（権限）では参照できない	特定の担当者（権限）でログインした場合，画面自体に読取専用の制御がかかっていたため，ファイルの参照ができなかった	(B) 直接リソース	---

付録 B

変更設計書(間接リソース変化点付き)

ソース名		
変更仕様	ID	
	内容	
作成日		
作成者		
修正者		
確認者		
見積もり	行数	行
	時間	分
実績	行数	行
	時間	分

■修正方針

なし

■データ構造の変更

なし

■関数呼び出し構造の変更

なし

■関数外の変更

項目#	変更内容	予想行数	変化有無

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

■関数の変更

関数名	

変更内容

項目#	変更内容	予想行数	変化有無

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

確認項目

項目#	確認内容	チェック

付録 C

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