第6分科会 B グループ

後任担当者視点を取り入れた設計背景の形式知化による派生開発の品質向上策

The improved quality strategy in derivative development using design rationale which is made into explicit knowledge by taking in a successor's viewpoint

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
アドバイザー	足立　久美	株式会社デンソー
研究員（リーダ）	関野　浩之	株式会社　山武
	大坪　智治	株式会社インテック
	大内　智之	株式会社リンクレア

研究概要

派生開発におけるソフトウェアの品質は，熟練担当者が同じ製品に長く関わり続けることで維持されてきた側面が強い．しかしながら，熟練担当者の退職や他部署への移動等に対して，引き継ぎ期間が十分に取れないことや熟練担当者の持っている技術･スキルの伝達の難しさ等の理由により，後任担当者への引き継ぎが十分に実施されることは少ない．そして，後任担当者への引き継ぎに起因する変更ミス／モレや影響箇所の特定誤りによるトラブルも多いと聞く．

そこで我々は引き継ぎに起因するトラブルを低減するために，変更方法や影響箇所を特定するための仕様や設計の理由に関する情報（以降「マイスター情報」と呼ぶ）を，XDDP（eXtreme Derivative Development Process）の変更要求仕様書とトレーサビリティ･マトリクスを利用して引き出し，ドキュメントに残すしくみを考案した．これにより，熟練担当者から業務を引き継ぐ後任担当者は，ドキュメントから仕様や設計の理由に関する情報を取得して，変更ミス／モレや影響箇所の特定誤りによるトラブルを防止できるようになる．

Abstract

The quality of the software in derivative development tends to have been maintained by experienced technicians continuing being engaged in same product development for a long time. However, to experienced technician's retirement, change at one's other posts etc., the taking over to a successor has finished halfway in many cases for the reasons of the taking over period cannot fully be taken, the difficulty of taking over of the techniques and skills which experienced technicians have, etc. And it hears that there are many troubles by mistakes on changes, forgetting changes and specific mistakes in an influence part resulting from the taking over to a successor.

Then, in order to reduce the troubles resulting from the taking over by ensuring taking over of the techniques and skills which experienced technicians have, Information on restriction and reason for specification and design to specify change part and influence part ("Master information") is drawn out from experienced technicians by using specifications of change request and traceability matrix of XDDP. And we designed the technique of leaving them efficiently to documents. As a result, a successor who succeeded the business from experienced technicians was made to be able to prevent the troubles by mistakes on changes, forgetting changes and specific mistakes in an influence part because the successor referred from the document to information on the restriction and the reason for the specification and the design.

1. 研究動機

派生開発は新規開発とは異なり，他人が作成した設計書，ソースコードを読み込んで，開発を進めるため，その読解に時間がかかることや，既に動いているシステムに影響を与えないように変更しなければならないなど独特な難しさがある．このような環境の中で品質を確保するためには，変更仕様に対する変更箇所や影響箇所を適切に特定する必要がある．我々の組織ではひとりの熟練担当者を同じ製品に割り当て続けることで，対象システムの品質を維持してきた．そのため派生開発の現場では熟練担当者から後任担当者への引き継ぎは非常に重要である．しかし新製品開発やクレーム対応など中で引き継ぎ期間を十分に確保することの難しさ，熟練担当者の持っている技術・スキルを後任担当者に伝達することの難しさ等の障壁により，後任担当者への引き継ぎは十分に実施できていない．そのため後任担当者は変更ミス／モレ，影響箇所の特定を誤り，本番リリース後にトラブルを発生させている．

そこで我々は熟練担当者が持つ変更方法や影響箇所を特定する情報を，後任担当者が発生させた不具合事例に着目して分析した．その結果，熟練担当者は「仕様や設計の理由に関する情報」を頭の中で補い，それらを根拠に変更箇所や影響箇所を特定していることがわかった．これらの情報は我々の開発現場で利用しているドキュメントには記述されていないことが多い．その理由は熟練担当者間でこれらの情報が暗黙の了解となっており，ドキュメントに記載する情報として認識されていないことにある．我々は熟練担当者が暗黙の了解として扱い，ドキュメントに記載されていない仕様や設計の理由に関する情報を「マイスター情報」と定義した．

次に我々はUSDM(Universal Specification Describing Manner)のフォーマットに理由欄があることやXDDPでは変更の理由を記述することで適切な変更仕様を引き出す効果があることに着目し，解決策として変更要求仕様書とトレーサビリティ・マトリクスを利用し，「マイスター情報」を引き出すことを考えた．そしてこれらのフォーマットに後任担当者の視点（なぜこのようになっているのか疑問に思う視点）を取り入れた改良を追加し，「マイスター情報」を記述するしくみを考案した．

2. 現状分析

2.1. 不具合を発生させた，後任担当者に不足している知識

本章では熟練担当者から業務を引き継いだ後任担当者が発生させた不具合事例に着目し，後任担当者に不足していた情報から熟練担当者が持つ知識を明らかにする．変更方法と既存箇所に関して不足していた情報パターンの分析結果を表1と表2に示す．

表1：不足していた情報パターンの分析結果（変更方法の誤りの場合）

不具合要因	不足した情報のパターン	具体的な情報
仕様書に記述されていない仕様を認識できず，変更方法を間違えた．	A-1）画面仕様が明らかになっていない．	・画面仕様 ・ユースケース，ユースケースシナリオ
	A-2）リソースに与える影響が明らかになっていない．	・リソースの割り当て，及びその理由
	A-3）動作環境（OS，DB,サードパーディ部品など）の選定条件，理由が明らかになっていない．	・選定条件，及びその理由
設計書に記述されていないデータに関する情報を認識できず，変更方法を間違えた．	A-4）変更しようとする処理に対して入力されるデータの属性（データの型，データ範囲など）が明らかになっていない．	・既存データ利用時の制約，及びその理由 データ名称，データの型，データの初期値，データの範囲，データの値の意味
	A-5）変更しようとする処理に対して入力が必要なデータがすべて列挙されていない．	ソフトウェア設計を理解する資料 ・DFD ・フローチャート
設計書に記述されていない制御に関する情報を認識できず，変更方法を間違えた．	A-6）変更箇所の状態遷移が明らかになっていない．	変更箇所の処理構造を理解する資料 ・遷移図（ステートマシン図） ・フローチャート

表2:不足していた情報パターンの分析結果(影響箇所の誤りの場合)

不具合要因	不足した情報パターン	具体的な情報
設計書に記述されていないデータに関する情報を認識できず,変更方法を間違えた.	B-1)変更しようとする処理から出力されるデータの属性(データの型,データ範囲など)が明らかになっていない.	・既存データ利用時の制約,及びその理由 データ名称,データの型,データの初期値,データの範囲,データの値の意味
	B-2)変更しようとする処理から出力されるデータを利用する他の処理が明らかになっていない.	・データにアクセスする処理(読み出し,書き出し)

※各不具合の詳細は付録の「付表1:後任担当者が発生させた不具合事例」を参照

2.2. 後任担当者に不足していた情報とドキュメントに記述すべき情報のマッピング

後任担当者に不足していた情報は本来仕様書や設計書に記述されるべき内容であると考えられる.従って後任担当者に不足していた情報が仕様書や設計書のどの部分に記述すべき情報であるのかを特定することができれば,熟練担当者の持つ知識は仕様書や設計書にすべて記述できることになる.そこで後任担当者に不足した情報が付表2で示されたドキュメントのどの部分に記述すべき情報であるのかを検討した.その結果を表3と表4に示す.

表3:後任担当者に不足していた情報(変更方法の誤りの場合)

不足した情報のパターン	具体的な情報	その情報をどこから入手するのか?
A-1)新規開発時の仕様が明らかになっていない.	・画面仕様 ・ユースケース,ユースケースシナリオ	【ソフトウェア要求仕様書】 5. ユースケースとユースケースシナリオ
A-2)リソースに与える影響が明らかになっていない.	・リソースの割り当て,及びその理由	【システム・アーキテクチャ設計書】 6. システムで扱うデータ 【ソフトウェア・アーキテクチャ設計書】 4.1 メモリ構成/レイアウト 【ソフトウェア詳細設計書】 3.3 リソース定義 3.6 共通定義
A-3)動作環境(OS,DB,サードパーティ部品など)の選定条件,理由が明らかになっていない.	・選定条件,及びその理由	【システム要求仕様書】 4. 制約条件
A-4)変更しようとする処理に対して入力されるデータの属性(データの型,データ範囲など)が明らかになっていない.	・既存データ利用時の制約,及びその理由 データ名称,データの型,データの初期値,データの範囲,データの値の意味	【システム・アーキテクチャ設計書】 6. システムで扱うデータ 【ソフトウェア・アーキテクチャ設計書】 4.1 メモリ構成/レイアウト 【ソフトウェア詳細設計書】 3.3 リソース定義 3.6 共通定義
A-5)変更しようとする処理に対して入力が必要なデータがすべて列挙されていない.	ソフトウェア設計を理解する資料 ・DFD ・フローチャート	【システム・アーキテクチャ設計書】 6. システムで扱うデータ 【ソフトウェア・アーキテクチャ設計書】 4.1 メモリ構成/レイアウト 【ソフトウェア詳細設計書】 3.3 リソース定義 3.6 共通定義
A-6)変更箇所の状態遷移が明らかになっていない.	変更箇所の処理構造を理解する資料 ・遷移図(ステートマシン図) ・フローチャート	【ソフトウェア詳細設計書】 3.1 プログラムユニット詳細処理 3.2 状態管理

表 4：後任担当者に不足していた情報（影響箇所の誤りの場合）

不足していた情報パターン	何の情報があれば防止できたのか？	その情報をどこから入手するのか？
B-1)変更しようとする処理から出力されるデータの属性（データの型，データ範囲など）が明らかになっていない．	・既存データ利用時の制約，及びその理由 データ名称，データの型，データの初期値，データの範囲，データの値の意味	【システム・アーキテクチャ設計書】 6. システムで扱うデータ 【ソフトウェア・アーキテクチャ設計書】 4.1 メモリ構成/レイアウト 【ソフトウェア詳細設計書】 3.3 リソース定義 3.6 共通定義
B-2)変更しようとする処理から出力されるデータを利用する他の処理が明らかになっていない．	・データにアクセスする処理（読み出し，書き出し）	【システム・アーキテクチャ設計書】 6. システムで扱うデータ 【ソフトウェア・アーキテクチャ設計書】 4.1 メモリ構成/レイアウト 【ソフトウェア詳細設計書】 3.3 リソース定義 3.6 共通定義

2.3. ドキュメントに記述されていない後任担当者に不足していた情報

表 3 と表 4 より後任担当者に不足していた情報は全て仕様書や設計書に記述できることがわかった．しかし我々が開発現場で利用しているドキュメントには必要な情報が十分に記述されているとは限らない．そのため後任担当者によるトラブルが発生していると考えられる．そこで何が記述されていないのかを我々の開発現場で利用されているドキュメントで調査した．その結果を表 5 に示す．

表 5：ドキュメントに記述されているもの/記述されていないもの

ドキュメント	記述されているもの	記述されていないもの
システム要求仕様書	1. 概要 2. システム構成 3. 機能概要 6. 機能詳細	4. 制約条件 5. ユースケースとユースケースシナリオ 7. 性能・品質等非機能要求詳細
システム・アーキテクチャ設計書	1. 概要 2. システム構成 3. 機能ブロック概要 4. 制御方式 4.1 制御シーケンス 4.2 ユースケースと機能ブロックの対応 5. 機能ブロック詳細 6. システムで扱うデータ	4.3 性能見積 7. 例外一覧
ソフトウェア要求仕様書	1. 概要 2. システム構成 3. 機能概要 6.機能詳細 7.インタフェース詳細	4. 制約条件 5.ユースケースとユースケースシナリオ 8.性能・品質等非機能要求詳細
ソフトウェア・アーキテクチャ設計書	1. 概要 2. システム構成 3. ソフトウェア構成 4. 制御方式 4.1 メモリー構成/レイアウト 4.2 ソフトウェア制御方式 5. 機能ユニット詳細	4.3 性能見積
ソフトウェア詳細設計書	1. 概要 2. プログラムユニット機能/構成設計書 2.1 プログラムユニット一覧表 2.2 プログラムユニット構成図 4. プログラムユニット・インタフェース設計書 4.1 シーケンス図 4.2 インタフェース詳細	3. プログラムユニット設計書 3.1 プログラムユニット詳細処理 3.2 状態管理 3.3 リソース定義 3.4 ハードウェア制御方法 3.5 システム初期化処理 3.6 共通定義 5. メモリ使用量

表5より，我々の開発現場で利用されるドキュメントには，制約条件，ユースケースとユースケースシナリオ，性能・品質等非機能要求詳細，仕様や設計の理由などが記述できていないことがわかった．

次に我々は上記の内容がドキュメントに記述されていない理由を明らかにするために，我々がドキュメントの調査を実施したプロジェクトの熟練担当者に対してインタビューを実施した．その結果を表6に示す．

表6：マイスター情報がドキュメントに記述されていない理由

問題	内容
ドキュメント作成時の問題	熟練担当者はドキュメントの利用者として自分を想定してドキュメントに情報を残している．自分で知っていることは，頭の中で補完できるため，ドキュメントには記述しない．
レビュー時の問題	設計の理由や背景といった「マイスター情報」は，熟練担当者間で暗黙の了解として共有されている．そのため，ドキュメント上に記述がなくても暗黙的に了解されて指摘されない．また指摘があったとしても，口頭またはメールで完結し，その内容は正式なドキュメントに反映されない．

2.4. 現状分析のまとめ

現状分析で明らかになったドキュメント上に記述されていない情報をエンタープライズ系システムと組込み系システムで分けて整理するとそれぞれに特徴があった．エンタープライズ系システムでは，業務的な背景によって決定された仕様や設計の理由に関する情報が記述されていないことが多かった．また組込み系システムでは，製品の要求を満たすために決定された仕様や設計の理由に関する情報，および性能・品質等非機能要求の理由・背景に関する情報が記述されていないことが多かった．

派生開発では通常，設計書に記述されている情報に不足がある場合は，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）を実施してシステムに関する情報を補っている．しかし，現状分析で明らかになったドキュメント上に記述されていない情報は，熟練担当者の頭の中にだけあるため，どれもスペックアウトでは明らかにすることができない情報である。従って，熟練担当者から業務を引き継ぐ後任担当者がこれらの情報を入手するためには，熟練担当者に口頭で尋ねて確認するしかない．熟練担当者の退職や他部署への移動等があると，後任担当者はこれらの情報を入手できなくなる．

仕様や設計の理由に関する情報は，現時点の派生開発を実施する上では，特に必要にならない情報である．しかし，将来，同じ箇所の変更を行うことになった場合，その仕様や設計の理由を正しく理解しないで変更すると変更方法を誤り，思わぬ場所に影響を与えることになる．

我々は，このようにドキュメントに記述されておらず，熟練担当者の頭の中だけにある変更方法や影響箇所などの正当性を示す根拠となるような情報を「マイスター情報」と定義した．

3. XDDPによるマイスター情報を残すための方法

3.1 本研究の課題

我々の現場では，新規開発やクレーム対応等により，熟練担当者から後任担当者へ「マイスター情報」を引き継ぐための時間を確保することが難しい．また「マイスター情報」は熟練担当者の頭の中にしかないため，熟練担当者がプロジェクトに在籍し，派生開発を行なっている間に対策を打つことが重要であると考えた．本論文の課題は以下の2つである．

(1) 現在，保守を担当している熟練担当者が自ら「マイスター情報」をドキュメント上に残せるようになる．
(2) 熟練担当者間でのドキュメントレビューにおいて，「マイスター情報」が記述されていないことを確認し，指摘できるようになる．

3.2. 解決策

3.2.1. 解決策のポイント

現在，保守を担当している熟練担当者が「マイスター情報」をドキュメント上に残せるようにするためには，熟練担当者に「マイスター情報」を機械的に残させるしくみが必要である．また，熟練担当者間のドキュメントレビューにおいて，「マイスター情報」が記述されていないことを確認し，指摘できるようになるには「マイスター情報」の記述有無が一目でわかるしくみが必要である．

そこで我々は XDDP の成果物である変更要求仕様書とﾄﾚｰｻﾋﾞﾘﾃｨ･ﾏﾄﾘｸｽに着目した．これらの成果物には理由欄が備わっており，我々の課題を解決するためのしくみが準備されている．またこれら既に用意されている成果物を改造することで，開発現場への導入負荷を軽減する．次章からそれぞれの成果物による課題解決のしくみを解説する．

3.2.2. 「マイスター情報」を引き出すしくみ

■変更要求仕様書

我々は仕様や設計の理由を引き出すしくみとして変更要求仕様書を選んだ．変更要求仕様書に記述する変更仕様はソースコード(関数仕様)のレベルで記述されるので，仕様と設計を同時に扱える．また変更要求仕様書には以下の特徴があり，熟練技術者から「マイスター情報」を引き出すしくみとして活用できる．

- 要求と仕様を階層関係で表現するため変更仕様を決定したひとつの根拠として要求を捉えることができる．
- 変更仕様と理由（「今回の変更によって効果を上げたいこと」や「変更されないと困ること」など）をセットで捉えることで適切な変更仕様を引き出すことができる．

3.2 章の課題(1)(2)を解決するためには，上記の変更要求仕様書で十分であるが，変更仕様は「変更前(before)／変更後(after)」形式で変更内容が記述されており，「before」に対する問題を「after」で解決する効果を狙っている．そこで，理由欄を変更前の仕様の理由を記述する『変更前理由』欄と変更後の仕様の理由を記述する『変更後理由』欄に分け，変更前の仕様の理由も記述できるようにした．ｲﾒｰｼﾞを図1に示す．

また，変更前理由や変更後理由を記述していく中で，ベースの仕様書、設計書などに記述されていない理由が明らかになった場合、変更後理由の先頭に◎をつけ，変更要求仕様書の内容をベースの仕様書、設計書にマージする時の目印とする．

■ﾄﾚｰｻﾋﾞﾘﾃｨ･ﾏﾄﾘｸｽ

我々は非機能要求を引き出すしくみとしてﾄﾚｰｻﾋﾞﾘﾃｨ･ﾏﾄﾘｸｽを選んだ．ﾄﾚｰｻﾋﾞﾘﾃｨ･ﾏﾄﾘｸｽの列には単純にソースコードを並べるのではなく，システムのモジュール構成を表現できるように，タスクごとにモジュールをグルーピングする．そしてﾄﾚｰｻﾋﾞﾘﾃｨ･ﾏﾄﾘｸｽ下部にタスクやモジュールで実現すべき非機能要求の有無を記述する欄を追加し，以下の効果が得られるようにした．ｲﾒｰｼﾞを図1に示す．

ﾄﾚｰｻﾋﾞﾘﾃｨ･ﾏﾄﾘｸｽに○をつけたタスクやモジュールに対して，

非機能要求が未調査（欄が未記入）であれば，

- 変更箇所の非機能要求を調べ,ﾄﾚｰｻﾋﾞﾘﾃｨ･ﾏﾄﾘｸｽ下部の欄に有無を「○／×」で記述する．モジュールがタスクの非機能要求に従う場合は「→」で記述する．なお非機能要求が有の場合は制約などをガｲﾄﾞに記述する．

非機能要求が調査済（欄が「○」）であれば，

- ｶﾞｲﾄﾞから非機能要求を前もって知っていることで，変更仕様を実現するための具体的な変更方法を決定するにあたり，適切な変更方法を決定できる．

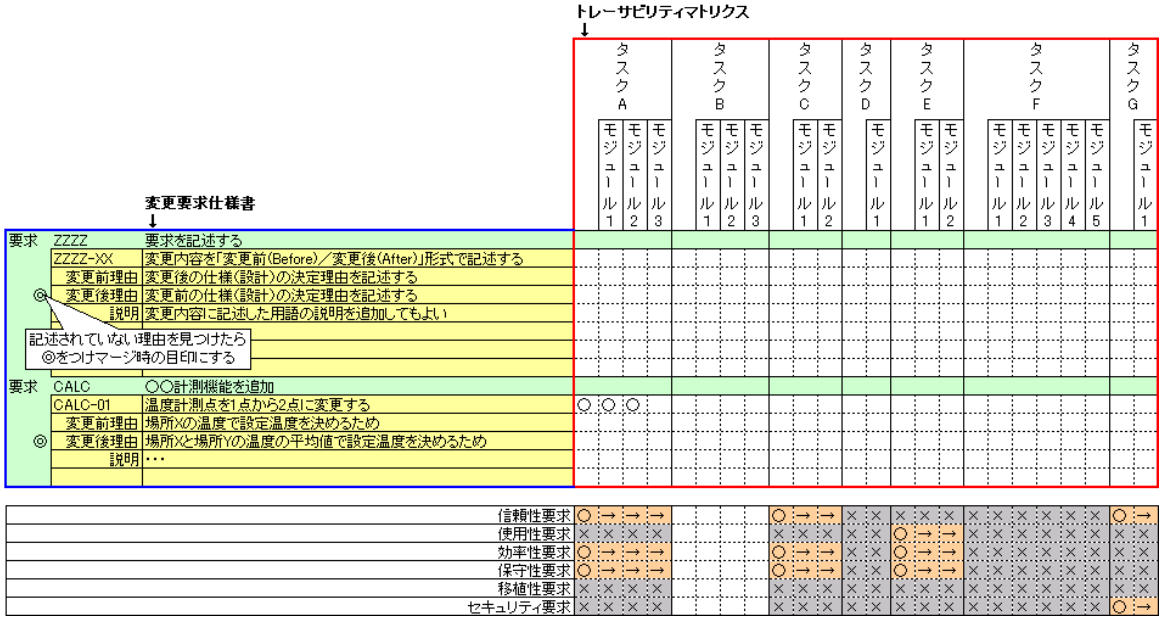

図1:変更要求仕様書とトレーサビリティマトリクス例

3.2.3. 「マイスター情報」を残すしくみ

変更要求仕様書は最後にベースの仕様書や設計書にマージする．この時に変更要求仕様書の変更後理由の先頭に◎がついた仕様や設計の理由をマージする．理由を記述する場所は特に規定はしないが，後から読み手に気づきを与え易いように，仕様や設計と同じページ内にマージすることが望ましい．ただし仕様や設計の項目がたくさんある場合は，どれが仕様でどれが理由なのか分からなくなるので別ページにマージすることが望ましい．

4. 解決策の検証

4.1. 検証方法

本研究で改良した変更要求仕様書とトレーサビリティ・マトリクスを過去実施した派生開発にシミュレーションの形で適用し、その効果を検証した。

4.2. 検証結果

シミュレーションの結果，我々が改良した変更要求仕様書とトレーサビリティ・マトリクスを用いれば，現担当者自ら「マイスター情報」を意識して，ドキュメントに情報として残せることがわかった．に実験の中でドキュメント上に残すことができた「マイスター情報」を表7に示す．

表7:熟練担当者自ら残すことができるようになった情報

システム形態	残すことができた情報の種類	残すことができた情報の具体例
エンタープライズ系	仕様や設計を決定した業務背景	2011/11/24が○○対応の本番リリース日であったがリリース後2週間の間は，過去申込データについては,旧判定処理を行わなければならなかったため新/旧判定処理を両方とも残した.
組み込み系	性能品質等非機能要求	○○管理タスク処理は325msec周期で実行されるので,50msec以内に処理を完了させること.

4.3. 検証結果の考察

検証の結果，変更要求仕様書の新フォーマットを使うことで，熟練担当者は変更仕様の仕様（設計）の理由に気づき，情報を追加することができた．これによって「マイスター情報（仕様（設計）の理由）」を引き出すという目

的に対して，期待した効果が得られたと考えられる．また後任担当者は変更前の仕様（設計）の理由と変更後の仕様（設計）の理由を比較することで，仕様（設計）の変遷を明確化できることを確認した．設計理由を記述させるようにしたことで，熟練担当者に後任担当者視点が加わる効果が発揮され，変更前後の理由を記述することができたと考えられる．そして今回の変更内容を元々の設計理由に照らし合わせ，その妥当性をセルフレビューする効果も得られたと考えられる．

次にトレーサビリティ・マトリクスの新フォーマットにおいても，タスクやモジュールの非機能要求を記述することができた．こちらも「マイスター情報（非機能要求）」を引き出すという目的に対して，期待した効果が得られたと考えられる．また前もってガイドで非機能要求を知ることで，変更仕様を実現するための具体的な変更方法を決定するにあたり，最初から非機能要求を考慮した変更方法を決められる効果を確認できた．特にパフォーマンスやリソースなどの難しい非機能要求であっても，それら前提にできることを積み上げればよく，効率的に変更方法を決められたためであると考えられる．

5. まとめ

5.1. 結果

本研究では，仕様や設計の理由といった情報が設計書上に残されていないことによって，これまでシステムを担当していた熟練担当者から後任担当者に業務が引き継がれた際に変更箇所，変更方法，影響箇所の特定を誤り，不具合を発生させていることを明らかにした．

我々は XDDP の変更要求仕様書とトレーサビリティ・マトリクスを改良して，現在システムを担当している熟練担当者が後任担当者のために「マイスター情報」をドキュメントに残せるしくみを考案した．そのしくみを利用することにより，現在システムを担当している熟練担当者が機械的に「マイスター情報」をドキュメントに残すことができるようになった．また，ドキュメントレビューの担当者もドキュメント上の「マイスター情報」に関する記述有無を容易に判別することが可能になり「マイスター情報」の不足を指摘できるようになった．その結果，熟練担当者から業務を引き継ぐ後任担当者は，ドキュメント上の「マイスター情報」を参照することによって，変更や影響箇所の正当性を検証することが可能になり，不具合の発生を防ぐことができると考えられる．

5.2. 今後の課題

派生開発の品質向上という観点において，本研究は，「マイスター情報」を残す方法までの提案に留まった．そのため，ドキュメント上に残されるようになった「マイスター情報」を実際の派生開発の中で利用してその効果を確認することができていない．今後は，残された「マイスター情報」を実際の派生開発の中で利用してその効果を検証していくことが課題である．

また「マイスター情報」を残すために本研究で提案した変更要求仕様書とトレーサビリティ・マトリクスは，派生開発を実施するタイミングで利用するものなので，派生開発が実際に行われた箇所に関わるドキュメントしか「マイスター情報」を残すことができない．しかし将来の変更箇所や変更内容は想定することは不可能なので，本来であればプロジェクト内にある派生開発に関わる全てのドキュメントに対して「マイスター情報」が追加されていることが望ましい．そのためには，派生開発が実際に行われなくてもプロジェクト内のドキュメントを見直し，「マイスター情報」を効率的に残していくしくみが必要である．この点についても今後の課題である．

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，技術評論社，2007
[2] 組込みソフトウェア向け 開発プロセスガイド，ソフトウェア・エンジニアリング・センター，翔泳社

付録

付表 1:後任担当者が発生させた不具合事例

No	不具合現象 (結果)	直接原因	不足した情報のパターン	具体的な情報
1	バージョンアップしたら,メニューに登録していた情報が表示できなくなった.	既存データ(文字列)を参照する機能を追加した.だだし,そのデータがNULL(空文字列)になる場合があることに気づかなかった.	変更しようとする処理において入力となるデータの属性 (データの型,データ範囲など) が明らかになっていない.	・既存データ利用時の制約,及びその理由 データ名称,データの型,データの初期値,データの範囲,データの値の意味
2	ファイルのファイルロックタイムアウトにより,リスタートする.	関数内にファイルをロックする処理を追加したが,その関数の呼び出し元でも別のファイルをロックしており,知らない内にデットロック構造になってしまった.	変更方法に依存した注意点が明らかになっていない.	変更方法に依存した注意点(過去の失敗事例から得た教訓)
3	処理Aのタイマーが設定される条件はステータスがXという条件がシステム設計書に書いてなかったため,処理Bの追加でステータスをXからYに書き換える処理を追加してしまった.	状態遷移を追加したら,既存処理を呼び出す状態に遷移できなくなり,既存処理が動作しなくなった. 設計書に書かれていない処理がソースコードにあることに気づかないで,状態遷移の設計を間違えた.	変更箇所の状態遷移が明らかになっていない.	変更箇所の処理構造を理解する資料,またはスペックアウトの観点・方法
4	通信負荷が高くなるとリスタートする.	タスク間通信用メモリがメモリフルになったことでリスタートした. 今回メモリフルになった部分はタスク間通信用のメモリで.過去の不具合修正の時にメモリを拡張しようとしたが拡張できていなかった.	リソースに与える影響が明らかになっていない.	・リソースの割り当て,及びその理由
5	毎日のデータ収集が1日以上かかり (通常は2時間程度) 収集が追いつかない.	データ拡張したが,データレプリケーションを想定して,リソースの再設計をしなかった.	リソースに与える影響が明らかになっていない.	・リソースの割り当て,及びその理由
6	サーバにアクセスしログインした履歴が,同時刻に重複して記録される.重複を解消したい.	ボタン押下時の処理は,入力の排他制御を考慮すべきであったが,抜けてしまった.	新規開発時の仕様が明らかになっていない.	・画面仕様 ・ユースケース,ユースケースシナリオ
7	クライアントPCからのマルチアクセスがあるとjdbc-odbcが停止してしまい,に画面表示が失敗する.	jdbc-odbcはマルチスレッドで利用可能と思い込んでいたが,実は利用できなかった.	動作環境 (OS,DB,サードパーディ部品など) の選定条件,理由が明らかになっていない.	・選定条件,及びその理由
8	カレンダーマスタの参照処理を追加した結果,カレンダーマスタに保持していない過去の日付を参照しようとしてシステムエラーが発生するようになった.	契約の契約開始日は1991年からのデータが存在しているが,カレンダーマスタについては,1994年からしかデータを保持していなかった.そのため,1994年より前に契約が開始された契約で本処理を行った際にカレンダーマスタ上に存在しないデータを参照しようとしてシステムエラーとなった.	変更しようとする処理において入力となるデータの属性 (データの型,データ範囲など) が明らかになっていない.	・既存データ利用時の制約,及びその理由 データ名称,データの型,データの初期値,データの範囲,データの値の意味
9	入金状況一覧の「入金」「残高」の値,未収残高一覧の「○○費未収残高」の値が不正となっている.	未収残高一覧の作成は,複数入金パターンごとにデータを収集して行っており,今回の改修で追加となったテーブルについても追加で読み込んで処理する必要があったが漏れ	変更しようとする処理において入力となるデータがすべて列挙されていない.	ソフトウェア設計を理解する資料 ・DFD ・フローチャート

		てしまった.		
10	ある金融商品の契約についてシステムで「運用開始年月」という項目を保持している.この金融商品は1年ごとに契約を継続するかしないかを顧客が選択する.継続契約を申し込んだ顧客の当該契約における「運用開始年月」について,ある担当者は再契約を行った月を出力しようとしていた.業務要件上は,一番最初に本契約の運用が開始された日付を出力すべきであった.	画面に「運用開始年月」と表示する仕様だったので,そこに表示するデータは当然ＤＢ上の「運用開始年月」であると思い込んでしまった. ※DB上では契約ごとに以下2つの運用開始年月を持っている. ①初回運用開始年月：当該契約が一番最初に運用された年月 ②運用開始年月：当該契約が継続された年月	変更しようとする処理において入力となるデータの属性（データの型,データ範囲など）が明らかになっていない.	・既存データ利用時の制約,及びその理由 データ名称,データの型,データの初期値,データの範囲,データの値の意味
11	ワークフローの管理ステータスの状態遷移に関するテストケースの記述に誤りがあった.テスト結果欄に遷移不可能なステータスに遷移可能である旨が記載されていた.	状態遷移を記述した設計書はあったが,テストケース作成者が設計書を調査していなかった.	変更箇所の状態遷移が明らかになっていない.	変更箇所の処理構造を理解する資料,またはスペックアウトの観点・方法
12	追加した顧客検索機能にて全件検索を実施し,データエクスポート(CSV出力)を実行した際,3桁として定義されている出力項目「運用日数」に4桁のデータが発生してCSV作成処理で異常終了した.	運用日数の算出は［現在の日付から当該契約の運用開始日］を引いた値で算出している. 運運用開始日は1年ごとの契約更新により新しい運用開始日が設定されるので,通常は365日を超えることはない. しかし契約を取り消した顧客については,運用開始日が更新されないため,今回のケースでは,契約を取り消した顧客について［2008年9月1日から2011年8月12日］の運用日数が1075日となり,異常終了した.	変更しようとする処理において出力となるデータの属性（データの型,データ範囲など）が明らかになっていない.	・既存データ利用時の制約,及びその理由 データ名称,データの型,データの初期値,データの範囲,データの値の意味
13	顧客検索機能の結果一覧にて適切なソート順が設定されていなかった.	個々の契約テーブルをひとつのテーブルに統合する場合,顧客IDは契約テーブルごとに記録されているため,顧客IDに重複がある.そのため,契約者ごとにGROUP BYすることに気づかなかった.	新規開発時の仕様が明らかになっていない.	・画面仕様 ・ユースケース,ユースケースシナリオ
14	基幹システムの共通処理にて使用しているテーブルAにカラム追加した際に,他システム側バッチ処理にてアベンドした.	テーブルにカラムを追加したが,他システムにこのテーブルからSELECT-INSERTする処理があり,INSERTするテーブルにもカラム追加しなければならないことに気づかなかった.	変更しようとする処理において出力となるデータを利用する他の処理が明らかになっていない.	・データにアクセスする処理（読み出し,書き出し）
15	顧客の変動情報テーブルにて未使用と認識していたカラムA="1"のコード値を別の仕様で使い回したことにより,そのテーブルを参照していたバッチ処理がアベンドした.	未使用と認識していたカラムAには,カラムA="1"のコード値を前提とした変換処理が不要なロジックとして残っており,その変換処理でエラーが発生しバッチ処理がアベンドした. 本システムのリプレイス当初は,カラムA="1"のデータは仕様として盛り込まれてていたが,最終的には不要となり未使用となった.しかしロジックのみ残っており,対象データも発生しなかったため処理としては空振りし続けていた.	変更しようとする処理において出力となるデータを利用する他の処理が明らかになっていない.	・データにアクセスする処理（読み出し,書き出し）

付表2:ドキュメントの記載内容と記載項目例

ドキュメント	記述内容	記述項目例
システム要求仕様書	システム要求定義の作業で検討した,システムとして実現が求められる機能要求事項,非機能要求事項や制約条件などを記述	1. 概要 2. システム構成 3. 機能概要 4. 制約条件 5. ユースケースとユースケースシナリオ 6. 機能詳細 7. 性能・品質等非機能要求詳細 8. その他
システム・アーキテクチャ設計書	システム・アーキテクチャ設計の作業で検討した,要求事項の実現方法(ソフトウェア構成,制御方式など)を記述	1. 概要 2. システム構成 3. 機能ブロック概要 4. 制御方式 4.1 制御シーケンス 4.2 ユースケースと機能ブロックの対応 4.3 性能見積 5. 機能ブロック詳細 6. システムで扱うデータ 7. 例外一覧 8. その他
ソフトウェア要求仕様書	ソフトウェア要求定義の作業で検討した,ソフトウェアとして実現が求められる機能要求事項,非機能要求事項や制約条件などを記述	1. 概要 2. システム構成 3. 機能概要 4. 制約条件 5.ユースケースとユースケースシナリオ 6.機能詳細 7.インタフェース詳細 8.性能・品質等非機能要求詳細 9.その他
ソフトウェア・アーキテクチャ設計書	ソフトウェア・アーキテクチャ設計の作業で検討した,要求事項の実現方法(ソフトウェア構成,制御方式など)を記述	1. 概要 2. システム構成 3. ソフトウェア構成 4. 制御方式 4.1 メモリ構成/レイアウト 4.2 ソフトウェア制御方式 4.3 性能見積 5. 機能ユニット詳細 6. その他
ソフトウェア詳細設計書	ソフトウェア詳細設計の作業で検討した,プログラムユニットの構成,詳細処理およびプログラムユニット間のインタフェースなどを実装可能なレベルで記述	1. 概要 2. プログラムユニット機能/構成設計書 2.1 プログラムユニット一覧表 2.2 プログラムユニット構成図 3. プログラムユニット設計書 3.1 プログラムユニット詳細処理 3.2 状態管理 3.3 リソース定義 3.4 ハードウェア制御方法 3.5 システム初期化処理 3.6 共通定義 4. プログラムユニット・インタフェース設計書 4.1 シーケンス図 4.2 インタフェース詳細 5. メモリ使用量

※組込みソフトウェア向け 開発プロセスガイド[2]より抜粋