第6分科会 Bグループ

XDDP によるデグレード防止効果の検証と その効果を高めるための方法

Simulation and Improvement for the prevention of degradation in XDDP

主査	足立　久美	株式会社デンソー
副主査	飯泉　紀子	株式会社日立ハイテクノロジーズ
副主査	清水　吉男	株式会社システムクリエイツ
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
	外谷地　茂	キヤノンＩＴソリューションズ株式会社
	長友　優治	株式会社ベリサーブ

研究概要

派生開発では，品質を確保する上で仕様変更による影響箇所を正確に見極めることが重要なポイントになる．そのため派生開発に必要な知識・スキル及び経験を充分に持ち，仕様変更による影響箇所を正確に見極めることができる熟練技術者が求められる．しかしながら我々のソフトウェア開発現場では，新製品開発やクレーム対応の状況により，派生開発に熟練技術者を割り当てることが難しく，変更モレ／変更ミスやデグレードの問題が繰り返し発生している．

そこで我々は XDDP（eXtreme Derivative Development Process）を導入し，熟練技術者に依存しなくても変更モレ／変更ミスやデグレードの問題を低減できないか検討を行った．その結果，開発経験から得られる知識・スキルの有無とその活用度合いが，XDDP のデグレード防止の効果に大きな影響を与えることが分かった．開発経験から得られる知識を補い活用させる仕組みとして，変更仕様から変更特性を抽出した「変更特性マトリックス」と，変更特性を手掛かりに関連する不具合から得られた教訓を確認できる「変更特性チェックリスト」を考案した．これにより，変更特性を手掛かりに膨大なチェック項目の中から変更仕様に関わるチェック項目だけを引き出せるようになるため，熟練技術者に頼ることなく XDDP によるデグレード防止効果を高めることができる．

Abstract

It is quite important for derivative development to judge correctly the part which was influenced by the change in specification when we ensure quality.Therefore, we need experienced technicians who have enough knowledgement and skills to judge correctly the part influenced by the change in specification.However, development of new products and dealing with complaints at our software development make those experienced difficult to be allocated to derivative development.In the results, we face those change-related problems such as degradation, mistakes on changes and forgetting changes again and again.

We studied to reduce the problems such as degradation, mistakes on changes and forgetting changes introducing XDDP even if we don't rely on experienced technicians.In the results, we learned that XDDP is not always effective enough for the prevention of degradation. The aim of this study is to enhance effectiveness for prevention of degradation with XDDP without relying on experienced technicians.By utilizing characteristics of changed content in derivative development, we created the method which can draw out only changed content related message from the existing checklist.

1. 研究動機

派生開発では仕様変更が頻繁に発生し，それらは短納期・低コストを要求される場合がほとんどである．また，繰り返される派生開発の中で設計書と実際のソースコードの乖離は次第に大きくなり，設計書が使い物にならなくなるなど，派生開発を取り巻く環境は劣悪である．このような環境の中でも，派生開発において，品質を確保するためには，仕様変更による影響箇所の正確な見極めが欠かせない．そのため現場では，派生開発に必要な知識・スキル及び経験を十分に持った熟練技術者を頼りに仕様変更による影響箇所の見極めを行っているのが実情である．

しかし実際の派生開発の現場では，新製品開発やクレーム対応などの影響により，常に熟練技術者を割り当てることが難しい．その結果，熟練技術者が不在となることで影響箇所の見極めを誤り，変更モレ／変更ミスやデグレードの問題が繰り返し発生してしまっている．中でもデグレードの問題については，これまで普通に機能していたものがある日突然機能しなくなってしまうということから，発生した際に深刻な問題となるケースが多い．そのため，派生開発の現場ではデグレード防止が大きな課題となっている．

そこで我々は，XDDP（eXtreme Derivative Development Process）を導入し，熟練技術者に依存しなくても変更モレ／変更ミスやデグレードの問題を低減することができないか研究を行った．XDDPとはそれぞれ視点の違った「3 点セット」の成果物の相互作用によって，担当者の思い込みや勘違いを低減するレビューの効果を引き出す方法である．研究の結果，XDDP は変更モレ／ミスには有効だが,デグレードの防止についての効果は熟練技術者が持つ「開発経験から得られる知識・スキル」の有無に大きく影響を受けることが分かった．そこで我々は，熟練技術者に頼ることなくXDDP によるデグレード防止効果を向上することを研究の目標とした．

熟練技術者に頼ることなく XDDP によるデグレード防止効果を向上するために，我々は派生開発における変更内容を分析し，以下の 2 つを組み合わせて使用する『気づきナビ』という手法を考案した．

・変更特性マトリックス：派生開発における変更内容の特性に着目し，共通するキーワードを抽出するしくみ

・変更特性チェックリスト：従来のチェックリストをキーワードごとに分類して改良したチェックリスト

この手法により，熟練技術者でなくても変更特性を手掛かりに仕様変更に必要な教訓を引き出すことができ，XDDP によるデグレード防止効果の向上を可能とした．通常，本手法のように新たな取り組みを組織内で始める場合は，新たな成果物の作成にかかる作業工数増加を嫌い，導入への抵抗感は大きなものとなる．しかし，本手法は組織内に蓄積された既存のチェックリストを利用するため，導入への抵抗感を小さくすることができた．

2. 現状分析

2.1. 不具合事例の分析

派生開発における不具合には「変更モレ／変更ミス」，「デグレード」，「潜在バグ」などがあるが，この中から主要因の問題を取り上げるため，本章では派生開発における不具合がどのような割合で発生しているのかを把握するために，各研究員から派生開発の不具合事例を収集し分類を行った．その結果を表 1 及び図 1 に示す．（各不具合の詳細は付録 A の「付表 1:分析を行った不具合事例」を参照）

表 1:不具合事例の分類結果

現象	計
A：変更モレ／変更ミス	20 件
B：デグレード	22 件
C：潜在バグ	7 件
D：対象外（※）	2 件
計	51 件

※導入時の手順ミスなど，開発工程に由来しない分析対象外の不具合

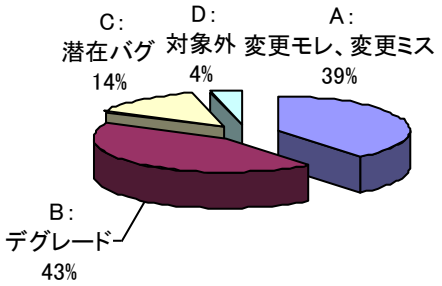


図1：不具合事例の分析結果（グラフ）結果

不具合事例の分類の結果，「変更モレ／変更ミス」と「デグレード」が共に全体の約 40%を占めており，これらの不具合を低減することが解決すべき課題であることが分かった．

2.2. XDDP による不具合防止効果のシミュレーション結果

本章では「変更モレ／変更ミス」と「デグレード」の問題を解決するために，XDDPを適用することで得られる不具合の防止効果を，不具合事例を用いたシミュレーションによって評価することにした．

なお，XDDP を適応しても不具合防止効果を得るためには技術者の経験・ノウハウに依存するところが多い．そこでそれらが不具合防止にどの程度影響するのかを把握するために，該当製品の開発経験や関連技術の知識に精通した「熟練技術者」と，それらの知識を持たない「非熟練技術者」とをシミュレーションのパラメータとした．これらの具体的な知識・スキルの違いは付表 2 となるが，非熟練技術者は熟練技術者に対して当該製品の開発経験から得られる知識・スキル（「ソースコードレベルの製品知識」「当該製品の顧客の業務・運用への知識」「当該製品の関連技術の知識」）が弱いものとする．

XDDP による不具合防止効果のシミュレーション結果を表 2 に示す．（全ての検討結果は付録 A の「付表 1: 分析を行った不具合事例」を参照）

この表により，熟練技術者プロジェクトでは「変更モレ／変更ミス」と「デグレード」の防止効果に大きな差異はなく，どちらも 80%以上である．それに対して非熟練技術者のプロジェクトでは「変更モレ／変更ミス」の防止効果は 53%を確保できたが，「デグレード」の防止効果は 20%に留まることがわかった．以上のことから「変更モレ／変更ミス」と「デグレード」のうち，非熟練技術者プロジェクトでの「デグレード」の防止が難しいということが言える．

表 2:XDDP の不具合防止シミュレーション結果

状況	現象	総計	XDDP 対象領域内	シミュレーション結果		XDDP 対象領域外 (※)
				防止可	防止不可	
ケース① 熟練技術者プロジェクト	A:変更モレ／変更ミス	20 件	19 件	17 件(89%)	2 件(11%)	1 件
	B：デグレード	22 件	20 件	16 件(80%)	4 件(20%)	2 件
ケース② 非熟練技術者プロジェクト	A:変更モレ／変更ミス	20 件	19 件	10 件(53%)	9 件(47%)	1 件
	B：デグレード	22 件	20 件	4 件(20%)	16 件(80%)	2 件

※：開発工程終了後の追加要件管理ミスなど，XDDP の対象とならない工程に原因があった不具合

2.3. デグレード防止に必要な知識・スキルの分析

2.2 のシミュレーション結果より，デグレード防止には熟練技術者が持っている知識やスキルが大きく寄与していることが分かった．そこで，本章では過去の不具合事例のうち 6 件を選び，デグレード防止に必要な知識・スキルを以下の観点で分析した．

・何（大カテゴリと小カテゴリの 2 階層で検討）をどのように変更したのか？

・何の検討(もしくは調査/確認/分析)が漏れていたか？なぜ漏れたか？

その結果を表 3 に示す．（全ての検討結果は付録 A の「付表 3:デグレード防止に必要な知識・スキルの分析」を参照）

表 3:デグレード防止に必要な知識・スキルの分析（抜粋）

No	何を変更したか？（大カテゴリ）	何を変更したか？（小カテゴリ）	どのように変更したか？	何の検討（調査/確認/分析)が漏れていたか？	なぜ漏れたか？
B-03	データ	データ構造	値の範囲を拡張	・データ範囲の妥当性	データの型は変更していないから安全という油断があった．
B-04	動作環境	Windows コンポーネント	別のソフトウェアをインストール	・Windows コンポーネントをバージョンアップすることでアプリケーションの動作に与える影響を確認していない など．	別の作業で Windows コンポーネントがバージョンアップされる可能性があることを想定できなかった．
B-09	データ	マルチロック	ローカル関数に既存のデータ（内部メモリ）を参照する処理を追加，関数内でデータをロック＆アンロック	・タスク全体/スレッド全体としてのマルチロックを確認していない	ローカル関数変更は安全という油断があった．

表 3 及び付表 3 から，デグレードの防止に必要な知識・スキルは以下の 3 点であることが分かった．つまり，非熟練技術者がデグレードを防ぐためには①～③の知識を得ることが必要であり，同時に①～③の知識を効率よく取得する手法がポイントとなる．なお本論文では，非熟練技術者がソースコードを読み込むだけでは得ることができない①～③を「ソースコード外知識」と定義する．

① データに関する情報（データ構造，データ範囲）」の知識　(B-03 等より)
② 制御に関する情報（関数呼び出し，イベント/タスクの流れ，データの排他制御）」の知識 (B-09 等より)
③ 「ソースコード以外で製品に影響を与えるもの（動作環境など）」の知識 (B-04 等より)

ここで，該当製品の熟練技術者にとって①～③は暗黙知である．これに対し非熟練技術者にとって①～②は設計書などを調査してベースソフトウェアのアーキテクチャを理解することで得られる知識である．しかし設計書が整備されていなければ，結局はソースコードを読み込み，データ構造や処理構造や制御構造を明らかにし，その上で DFD(Data Flow Diagram)やシーケンス図などを作成することでベースソフトウェアのアーキテクチャを理解しなければ得られない知識である．まして③は「動作環境の知識」を得るための調査方法を知らなければ，その存在さえ気づくことが難しい知識である．

2.4. XDDP による不具合防止効果シミュレーション結果のまとめ

非熟練技術者プロジェクトでもデグレードを防止するためには，非熟練技術者が「ソースコード外知識」を効率よく取得する手法がポイントとなる．派生開発では「ソースコード外知識」を得る手段は設計書であるが，設計書の記述内容が十分でなかったり，過去の派生開発の積み重ねで設計書と実際のソースコードが乖離していたりするなど，非熟練技術者が「ソースコード外知識」を得ることは難しい．

しかしながら多くの組織には，過去の不具合事例を元にした開発経験から得られたノウハウを積み上げて作成されているチェックリストがある．過去の不具合事例には「ソースコード外知識」の見落としにより発生したデグレードも多いと推測される．それらへの対策が記載されたチェックリストには過去の派生開発時に見落としていた「ソースコード外知識」が集積されていると考えた．そこで我々は非熟練技術者がチェックリストから容易に「ソースコード外知識」を引き出せる手法を考案すれば，過去に発生したデグレードの発生を低減することが可能であると考えた．

3. XDDP によるデグレード防止効果を高めるための方法

3.1. チェックリストの問題点と課題

現状のチェックリストはチェック項目を羅列したものであり，チェック項目の利用シーンや目的，観点によって整理されておらず，チェック項目が肥大化しやすい傾向にある．また，チェックリストを利用する対象者の想定も特定プロジェクトや特定製品を対象としておらず，組織内の様々なプロジェクトや製品を対象としているため，幅広い人が理解できるように記述内容を抽象化して応用性を持たせている．しかし抽象的な内容では，経験やスキルのない非熟練技術者はその内容を適切に理解することが難しく，チェックリストを有効に利用できていない．

これらの問題点について研究員の所属する各社で実際に利用している設計チェックリストを収集し，分析した結果(付録 B)，解決すべき課題は以下の 2 点（「表 4:チェックリストの課題」を参照）に集約した．

表 4:チェックリストの課題

No	課題	分析結果
1	膨大なチェック項目の中から必要なチェック項目を効率良く見つけ出すしくみをつくる.	付録B 付表5のNo.1
2	何を変更して発生した不具合であるのかがわかる情報をチェック項目に追加する.	付録B 付表5のNo.2, No.3

3.2. 解決策

3.2.1. 解決策のポイント

膨大なチェック項目の中から必要なチェック項目を効率良く見つけ出すしくみをつくるためには，派生開発の中で発生する変更仕様とそれに関係したチェック項目を結びつけるためのキーワードが必要であると考えた．そこで，変更仕様書から変更内容を簡潔な言葉として抜き出したものをキーワードとすることにした．このキーワードを以後「変更特性」と呼ぶ。変更仕様から変更特性を抽出する例を図 2 に示す．

（変更仕様）:「○○**画面に**××情報の**入力項目を追加**し，△△ボタン押下時に**必須チェックを追加**する．」
（変更特性）⇒「**画面入力項目の追加**」と「**必須チェックの追加**」

図 2:変更特性の抽出例

変更特性は何をどの様に変更したかを簡潔に示すキーワードであるため，既存のチェック項目に情報として付加することができれば，そのチェック項目内の教訓が何を変更したことによって発生した不具合から得られたものかをわかるようにすることができる．

3.2.2. 気づきナビ

我々は，「変更特性」を用いて，

・変更特性マトリックス：変更仕様から変更特性を抽出するしくみ
・変更特性チェックリスト：変更特性を手掛かりに，その変更特性が原因で発生した不具合から得られた教訓を確認できるしくみ

を考案することによって，変更特性を手掛かりに膨大なチェック項目の中から変更仕様に関わるチェック項目だけを引き出す手法を考えた．以後これを『気づきナビ』と呼び，その機能と特徴を表 5 に記す．

表 5:変更特性マトリックスと変更特性チェックリストの機能及び特徴

気づきナビの要素	機能	特徴
変更特性マトリックス	変更仕様から変更特性を抽出する	列項目に当該組織であらかじめ定義されている変更特性を持つ
変更特性チェックリスト	変更特性を手掛かりにその変更特性が原因で発生した不具合に関する教訓を確認できる	変更特性マトリックス上で定義されている変更特性ごとにチェック項目が用意され，変更特性が原因で発生した不具合に関する教訓がまとめられている

この『気づきナビ』は， XDDP の 3 点セットの1つである変更要求仕様書と組み合わせて利用することがポイントである．XDDP の変更要求仕様書は，以下の特徴を持っている．

・要求と仕様を階層関係で表現することで変更仕様をモレなく洗い出すことができる．
・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，何がどのように変更されるのかを理解しやすい．

次にこれらの特徴を踏まえた上で，『気づきナビ』の使い方について説明をする．変更特性マトリックスの行項目に変更要求仕様書の変更仕様を並べ，変更仕様ごとに変更特性マトリックスの列項目に定義された変更特性が含まれるかを確認していくことで，変更仕様から変更特性をモレなく容易に抽出することが可能になる．また，抽出された変更特性を手がかりに変更特性チェックリストを確認することで，変更仕様に必要なチェック項目だけを見つけ出すことができる．

『気づきナビ』を導入することによって以下の効果が得られる．導入前後のイメージを図 3 と図 4 に示す．

・導入前：変更仕様ごとにチェックリスト上の全チェック項目を確認する．
・導入後：変更仕様から抽出された変更特性に関係のあるチェック項目だけ確認する．

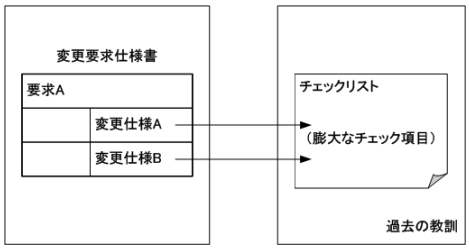


図 3:『気づきナビ』導入前

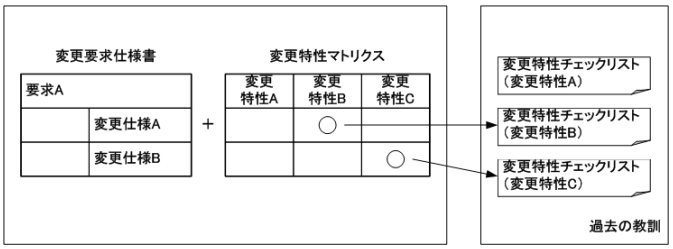


図 4:『気づきナビ』導入後

3.2.3. 変更特性マトリックス

変更特性マトリックスは，変更仕様とその変更仕様に含まれる変更特性の関係を表すことで，変更仕様における変更特性の分析を可能とするものである．変更特性マトリックスのフォーマットを図 5 に示す．

変更特性 / 変更仕様	カテゴリA			カテゴリB			カテゴリC		
	変更特性A1	変更特性A2	変更特性A3	変更特性B1	変更特性B2	変更特性B3	変更特性C1	変更特性C2	変更特性C3
○○の△△を××に変更する。									
○○に××を追加する。									
△△の○○を××に変更する。									

図 5:変更特性マトリックスのフォーマット

(1) 変更特性マトリックスの列項目

- **カテゴリ**：画面入力項目，共有モジュール，データ，制御，処理といった仕様中で変更・追加される対象となるものを分類したもの．カテゴリはプロジェクトごとに定義されることを想定している．例えば，サードパーティ製品を使用しているプロジェクトでは，カテゴリに「サードパーティ製品」があっても構わない．
- **変更特性**：カテゴリに対して変更・追加・削除などの動詞を追加したもの．例えば「画面入力項目」というカテゴリがあった場合，変更特性は画面入力項目変更，画面入力項目追加，画面入力項目削除などになる．

(2) 変更特性マトリックスの行項目

XDDP の 3 点セットの1つである変更要求仕様書によって抽出された変更仕様を記述する．

3.2.4. 変更特性チェックリスト

変更特性チェックリストは，既存のチェックリストに変更特性情報を追加することで作成される．変更特性チェックリストの作成例を「付表 11:変更特性チェックリストの例」に示す．

3.2.5. 使用手順

(1) 不具合から得られた教訓の活用

XDDP の作法に従い，変更要求仕様書を作成する．作成された変更要求仕様書に各組織，プロジェクトで運用している変更特性マトリックスを組み合わせる．変更要求仕様書上の変更仕様ごとに変更特性マトリックスの横軸に定義されている変更特性が変更仕様中に含まれるかどうかを検討し，含まれる場合は該当する変更特性の欄に○印をつける．○印がついた変更特性に紐づく変更特性チェックリストを確認し，変更仕様の見直しが必要であれば見直しを行う．イメージを図 6 に示す．

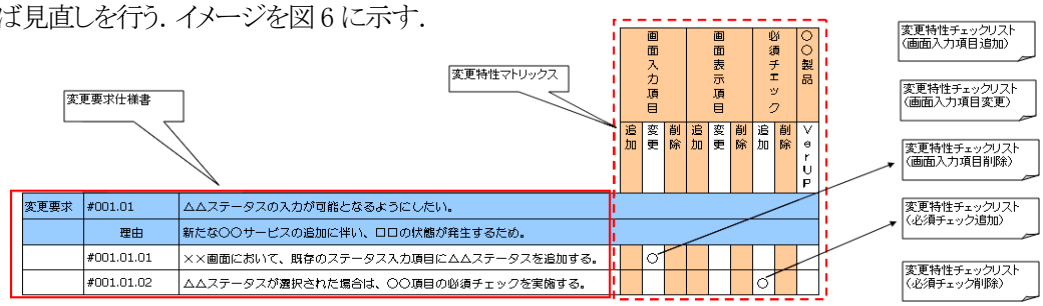


図 6:気づきナビによる変更特性チェックリスト項目への誘導

(2) 不具合から得られた教訓の蓄積

本手法を用いた開発の中で不具合が発生した場合は，不具合を発生させた仕様を変更要求仕様書上で特定し，対応する変更特性を変更特性マトリックス上で確認することで，どの変更特性によって発生した不具合であるのかを明らかにすることができる．これにより，変更特性の分析が行われた不具合から得られた教訓を変更特性チェックリストにグルーピングして蓄積することが可能になる．イメージを図 7 に示す．

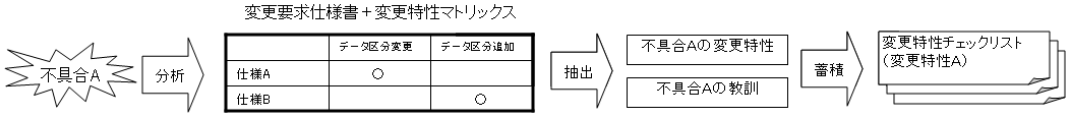


図 7:不具合から得た教訓の蓄積

4. 解決策の検証

4.1. 検証方法

『気づきナビ』の効果の検証は，表6に示すように，過去のデグレード事例に対するシミュレーション検証(①)

と，実際の XDDP のプロジェクト（組み込み系開発：仕様数：21 件）での検証（②）について行った．

表 6:『気づきナビ』の効果の検証方法

	対象	目的	方法
検証①	「変更特性チェックリスト」	変更特性により変更内容に対応したチェック項目にたどり着き，そこに有効な情報があった場合に，デグレード防止に関与するか否かを検証する．	1)過去のデグレード事例（付表8）に対し，デグレード防止のための変更特性とチェック項目を作成する． 2)次回以降の派生開発で，作成したチェック項目にたどり着けた場合，デグレード防止に関与するか，否かをシミュレーションする．
検証②	「変更特性マトリックス」「変更特性チェックリスト」	変更特性を利用すれば，確認が必要なチェック項目の数が少なくなることを検証する．	XDDPのプロジェクトに「変更特性マトリックス※」と「変更特性チェックリスト※」を適応し，変更内容に対するチェック項目を「チェックリストのカテゴリ」で抽出した場合と「変更特性マトリックスの変更特性」で抽出した場合で比較する．
		変更特性を利用すれば，経験に依存することなく，チェック項目を抽出できることを検証する．	XDDPのプロジェクトに「変更特性マトリックス※」と「変更特性チェックリスト※」を適応し，変更内容に対する熟練者技術者のチェック項目と非熟練技術者のチェック項目を比較する．
		変更特性チェックリストによって新たな欠陥に気づけることを検証する．	XDDPのプロジェクトに「変更特性マトリックス※」と「変更特性チェックリスト※」を適応し，変更内容に対する熟練者技術者のチェック項目と非熟練技術者のチェック項目を比較する．

※「変更特性マトリックス」と「変更特性チェックリスト」は既存のチェックリスト[3]から作成する．

4.2. 検証結果

検証①（過去のデグレード事例のシミュレーション）では 6 事例を検証した（各事例の詳細な結果は付表 8 を参照）．その結果，6 事例全てにおいて『気づきナビ』を利用することによって，技術者がデグレード防止のための回避策や注意事項，確認事項の教訓を取得でき，またそれらがデグレード防止に効果的であることが確認できた．検証②では実際に XDDP を適用したプロジェクトにおいて『気づきナビ』に期待される 3 つの効果が得られるか検証した．その結果，3 つの効果についていずれも当初期待した結果を得られた．表 7 に結果を示す．

表 7:『気づきナビ』に期待される効果と確認された効果

No	期待される効果	確認された効果
1	変更特性を利用すれば，確認が必要なチェック項目の数が少なくなる	確認するチェック項目の数が『気づきナビ』導入前に比べて約 17%少なくなった．（詳細は付録 C の「付表 9」参照）．
2	変更特性を利用すれば，経験に依存することなく，チェック項目を抽出できる	熟練技術者と非熟練技術者で確認したチェック項目がほぼ同数になった．
3	変更特性チェックリストによって新たな欠陥に気づく	非熟練技術者が見落としがちな異常処理に関する抜けに気づいた．

4.3. 考察

検証①（過去のデグレード事例のシミュレーション）の結果では，検証した 6 事例全てについて，技術者がデグレード防止のための回避策や注意事項，確認事項の教訓を取得できるということが確認できた．しかし，チェックリストの内容によっては，今回の検証結果と同等のデグレード防止に関する有益な情報を得られない可能性がある．これは今後の課題である．

検証②の結果（表 7）より，「熟練技術者に頼ることなく XDDP によるデグレード防止効果を高める」という目的に対して期待した効果が得られたと考える．表 8 に検証②の各結果に対する考察を記載する．

表 8:『気づきナビ』の検証結果に関する考察

No	検証結果	検証結果に対する考察
1	確認するチェック項目の数が『気づきナビ』導入前に比べて約 17%少なくなった	検証結果よりも高い効果を期待していたが，検証で利用したチェックリストには，設計共通のチェック項目が多く，製品固有のチェック項目が含まれていないことから，このような結果になったと考えられる．
2	熟練技術者と非熟練技術者で確認したチェック項目がほぼ同数になった	変更特性に従って機械的にチェック項目を確認することにより，知識・スキルに依存することなくチェック項目の確認が行えるようになったためと考えられる．
3	非熟練技術者が見落としがちな異常処理に関する抜けに気づいた	

また，検証を行っていく中で，デグレードの問題に限らず，変更ミスや変更モレといった他の問題に関しても多くの気づきがあることがわかった．これは，『気づきナビ』の本質が変更特性を抽出して変更仕様とチェック項目

(過去の教訓)の紐付けを行うことにあるため，チェック項目の内容がデグレードに関するものでなくてもデグレードの問題と同様の効果が発揮されたものと考えられる.

5. まとめ

5.1. 取り組みと結果

派生開発における変更内容の特性に着目した「変更特性マトリックス」と既存のチェックリストに改良を加えた「変更特性チェックリスト」を用いる手法『気づきナビ』を考案し，過去のデグレード不具合事例に対するシミュレーション検証と実際の XDDP を導入したプロジェクトでその効果を確認した.

その結果,『気づきナビ』を用いれば，技術者の知識・スキルに依存することなく，変更内容に必要なチェック項目を確認することができ，XDDP によるデグレード防止効果を高められることがわかった. また，通常新たな手法を組織内に展開する場合，新たな成果物の作成にかかる工数増加が敬遠され，導入への抵抗感は大きなものとなるが，本手法は組織内で運用されている既存チェックリストの整理のしかたを少し変えるだけなので，導入への抵抗感を小さくする効果も期待できる.

XDDP はそれぞれ視点の違った「3点セット」の成果物の相互作用によって担当者の思い込みや勘違いを低減し，レビューの効果を最大限引き出すことによって大幅な品質向上が可能になる手法である. しかしながら，その効果は「影響箇所の気づき」や「成果物の書き方や文章力」に左右され，さらには人の知識・スキル，姿勢の影響を受けてしまう. これは本研究で取り上げた熟練技術者と非熟練技術者の違いと本質を同じにする問題である.

本研究は『気づきナビ』によって「熟練技術者に頼ることなく XDDP によるデグレード防止効果を高める」ことに対して一定の成果をあげることができた. 同時に本研究のもう一つの成果として「XDDP 導入効果の均質化」についても一つの方向性を示すことができたと考える.

5.2. 今後の課題

派生開発における変更要求は多種多様なため，変更要求から新たな変更特性を見つけた場合，その変更特性を「変更特性マトリックス」，その設計の定石や注意点を「変更特性チェックリスト」に追加することで,『気づきナビ』はそれぞれの製品によりフィットしたものになっていくことが期待される(図 8 を参照). この有効性検証が今後の課題である. またチェックリストの工夫によって「XDDP 導入効果の均質化」を一層進化させることができると考えている.

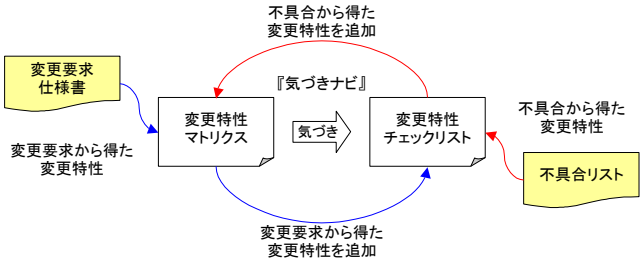


図 8:『気づきナビ』の成長モデル

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] 古畑 慶次:XDDP で現場を変える！～プロセス改善事例のご紹介～
CASE フェスタ 2010 ーソフトウェアエンジニアの集いー

[3] 組込み製品の品質を高めるための暗黙知の抽出・利用方法,
ソフトウェア品質管理研究会 第 23 年度(2007年度)分科会成果報告，日本科学技術連盟

付録 A

付表 1:分析を行った不具合事例

No	不具合現象	分類	XDDP の不具合防止シミュレーション結果				
			ケース① 熟練者 プロジェクト	3 点セット			ケース② 非熟練者 プロジェクト
				変更 要求 仕様書	TM	変更 設計書	
A-01	あるタイミングで印刷した場合, 指定した順番に印刷されない. （印刷順不正）	B：デグレード	○		○		○
A-02	１ページに収まるデータにも関わらず，２ページ出力された. （改行位置不正）	B：デグレード	○		○		○
A-03	指定したデータが出力されない. （出力データ不正）	B：デグレード	○			○	×
A-04	状態１中に，状態２となった場合，動作コマンドを送信できてしまった.	C：潜在バグ	対象外				対象外
A-05	条件１の場合，○○中かつ○○中に，メニューを開いたら，プログラムエラーが発生した.	B：デグレード	○	○	○		○
A-06	ステータス画面において，不正な情報が表示される. （表示不正）	C：潜在バグ	対象外				対象外
A-07	印刷日時において，印刷された日時でなく，データ作成日時だった. （印刷データ不良）	C：潜在バグ	対象外				対象外
A-08	処理不一致によりデータ未送信が発生した.	B：デグレード	×				×
A-09	複数範囲のデータを指定した場合，データ表示されるまで多大な時間がかかった.	B：デグレード	○	○			○
A-10	動作異常１と動作異常２の条件が重なった場合，停止モード中のままとなった.	C：潜在バグ	対象外				対象外
A-11	メインメニュー画面中のボタン名を変更したが，画面中の見出しには，古いボタン名が表示されている.	A：変更漏れ，変更ミス	○			○	○
A-12	画面で項目を選択（マスタで最大値を入力済み）すると，登録処理で異常終了する.	A：変更漏れ，変更ミス	○	○			○
A-13	帳票に，画面には存在しない項目が存在する. 基本設計書と異なる.	B：デグレード	対象外				対象外
A-14	画面上の項目が未入力であるエラーメッセージが，画面によって異なる.	C：潜在バグ	対象外				対象外
A-15	マスタメンテナンスの一覧項目がずれている.	A：変更漏れ，変更ミス	×				×
A-16	過去日付のデータで承認依頼取消を行なうと承認依頼できない.	A：変更漏れ，変更ミス	○		○		×
A-17	一定バイトずつデータを分割する時の境が全角文字にあたったとき，正しく分割されない.	C：潜在バグ	対象外				対象外

No	不具合現象	分類	XDDP の不具合防止シミュレーション結果				
			ケース① 熟練者 プロジェクト	3 点セット			ケース② 非熟練者 プロジェクト
				変更 要求 仕様書	TM	変更 設計書	
A-18	承認依頼後，データ取込を行っても再承認依頼待ちにならない	A：変更漏れ，変更ミス	対象外				対象外
A-19	マスタメンテナンスで，「表示しない」設定にした項目が画面のリストに表示される.	A：変更漏れ，変更ミス	○			○	×
A-20	画面にて，金額を表示する部分が「0」で表示されている.	B：デグレード	○		○		○
A-21	一定時間帯にログ監視が稼動しない.	A：変更漏れ，変更ミス	○			○	×
A-22	日跨ぎのジョブがスケジューラから稼動できない.	B：デグレード	×				×
A-23	サーバ間のファイル転送が行なえない.	D：対象外	対象外				対象外
A-24	WEB 画面から実施するバッチ処理が他ユーザとバッティングした時に，システムエラーになった.	B：デグレード	○	○			○
A-25	システム管理処理が異常終了した.	D：対象外	対象外				対象外
A-26	画面から追加登録されたデータが重複して連動された.	A：変更漏れ，変更ミス	○			○	×
A-27	黒データと赤データについて画面表示がおかしい.	C：潜在バグ	対象外				対象外
A-28	計算式中で補助単位を二重に適用してしまうケースがある.	A：変更漏れ，変更ミス	○			○	×
A-29	未対応の情報を入力すると Runtime Error になる.	B：デグレード	○	○			○
A-30	START および STOP コマンドで同一データを指定すると，指定したデータと，その次のデータが出力される.	A：変更漏れ，変更ミス	○	○		○	○
A-31	秒の表記が大文字の S 表記と小文字の s 表記混在で出力される.	A：変更漏れ，変更ミス	○	○			○
A-32	ユーザ向けのエラーメッセージを出すべき箇所で，開発者向けのメッセージにしてしまっていた.	A：変更漏れ，変更ミス	○	○		○	○
A-33	停止中のシステムに対してデータ取得を行なうと Runtime Error が表示される.	A：変更漏れ，変更ミス	○	○			×
A-34	あるソフトウェアで出力したファイルをそのソフトウェアの派生ソフトウェアに入力した場合に仕様の不一致によりエラーが発生する.	B：デグレード	○	○			○
A-35	必要な他システムが無くてもエラー無しに起動しなければならなかったが，組み込まれたライセンスソフト部分でエラーになった.	B：デグレード	○			○	○
A-36	ファイルに'#'が記述された場合に，コメント扱いにならず，エラーが発生する. そのソフトウェアの派生ソフトウェアでは，'#'はコメント扱いのため，開発対象のソフトウェアでも同様にコメントとして使用することが求められる.	B：デグレード	○	○			○
A-37	文字列のコピーが正しくできない.	B：デグレード	対象外				対象外

No	不具合現象	分類	XDDP の不具合防止シミュレーション結果				
			ケース① 熟練者 プロジェクト	3 点セット			ケース② 非熟練者 プロジェクト
				変更 要求 仕様書	TM	変更 設計書	
A-38	通話中のセッション B を変更ができない.	A：変更漏れ，変更ミス	○	○			×
A-39	通話中のセッション A を変更ができない.	A：変更漏れ，変更ミス	○	○			×
A-40	通話履歴情報が表示されない.	A：変更漏れ，変更ミス	×				×
A-41	XML データが，ヘッダのみの場合，ヘッダ情報がセーブされない. （ヘッダ＋コンテンツの場合はセーブ OK）	A：変更漏れ，変更ミス	○			○	○
B-01	追加した参照処理(共通関数呼び出し)のパラメータに，期待していたデータ範囲を下回る値が渡され，エラーになった.	B：デグレード	○	○			○
B-02	Windows サービスの再インストール時に変えた設定で，システムの一部機能が正常に動作しなくなった.	B：デグレード	×				×
B-03	コードの桁数を変更したところ，桁数依存だった処理が動かなくなった.	B：デグレード	○			○	○
B-04	○○ツール(VB アプリケーション)の画面を表示した所，「メモリ不足」とのエラーになった.	B：デグレード	○	○			○
B-05	現場実機を立ち上げた時に履歴データが欠測表示となった.	A：変更漏れ，変更ミス	○	○			○
B-06	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.	B：デグレード	○			○	○
B-07	データを No 順に出力する機能で，途中の No から入力していた場合，それより前の No が出力されない.	A：変更漏れ，変更ミス	○			○	○
B-08	○○機能を追加した所，画面を表示したままにすると，画面の通信状態が通信異常，正常を繰り返すようになった.	A：変更漏れ，変更ミス	○	○			○
B-09	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.	B：デグレード	○		○		○
B-10	フィルタリングルールをカスタマイズする機能で，「～のいずれかを含まない」「～を全て含まない」が逆になっていた.	B：デグレード	×				×

付表 2:熟練技術者と非熟練技術者の定義

No	知識・スキル	熟練技術者	熟練技術者
1	ITSS：IT アーキテクト，IT スペシャリスト， アプリケーションスペシャリスト，ソフトウェアデベロップメント ETSS：システムアーキテクト，ソフトウェアエンジニア	レベル 4 以上	レベル 1～3
2	ソースコードや設計書などからベースソフトウェアを調査しアーキテクチャを理解する知識	ある	ある
3	当該製品の開発経験	ある	なし
4	ソースコードレベルの製品知識	ある	なし
5	当該製品を利用する顧客側の業務・運用の知識	ある	なし
6	当該製品の関連技術※の知識	ある	なし

※エンタープライズ系：プラットフォーム（OS，VM），データベース，Web サーバ，通信（IP，TCP，HTTP）など
組み込み系：プラットフォーム（OS，VM），ストレージ，UI，情報処理，通信など

付表 3:デグレード防止に必要な知識・スキルの分析

No	何を 変更したか？ （大カテゴリ）	何を 変更したか？ （小カテゴリ）	どのように 変更したか？	何の検討 （調査/確認/分析）が 漏れていたか？	なぜ漏れたか？
B-01	データ	データ処理	既存の共通関数の呼び出しを追加	・共通関数に渡すパラメータの範囲	共通関数は実績があるから安全という油断があった.
B-02	動作環境	Windows サービス	再インストール	・Windows サービスを再インストールすることでシステムの動作に与える影響を確認していない ・システムの動作を保証するための確認事項が未確定	再インストールすることでサービスの動作が変わってしまうことを想定できなかった.
B-03	データ	データ構造	値の範囲を拡張	・データ範囲の妥当性	データの型は変更していないから安全という油断があった.
B-04	動作環境	Windows コンポーネント	別のソフトウェアをインストール	・Windows コンポーネントをバージョンアップすることでアプリケーションの動作に与えるに影響を確認していない ・アプリケーションの動作を保証するための確認事項が未確定	別の作業で Windows コンポーネントがバージョンアップされる可能性があることを想定できなかった.
B-06	データ	データ処理	既存のデータ（内部メモリ）を参照する処理を追加	・データ範囲の妥当性	既存データ参照は安全という油断があった.
B-09	データ	マルチロック	ローカル関数に既存のデータ（内部メモリ）を参照する処理を追加，関数内でデータをロック＆アンロック	・タスク全体/スレッド全体としてのマルチロックを確認していない	ローカル関数変更は安全という油断があった.

※B グループの不具合事例から「B:デグレード」かつ「熟練技術者でないと防止できないもの」について分析した.

付録 B

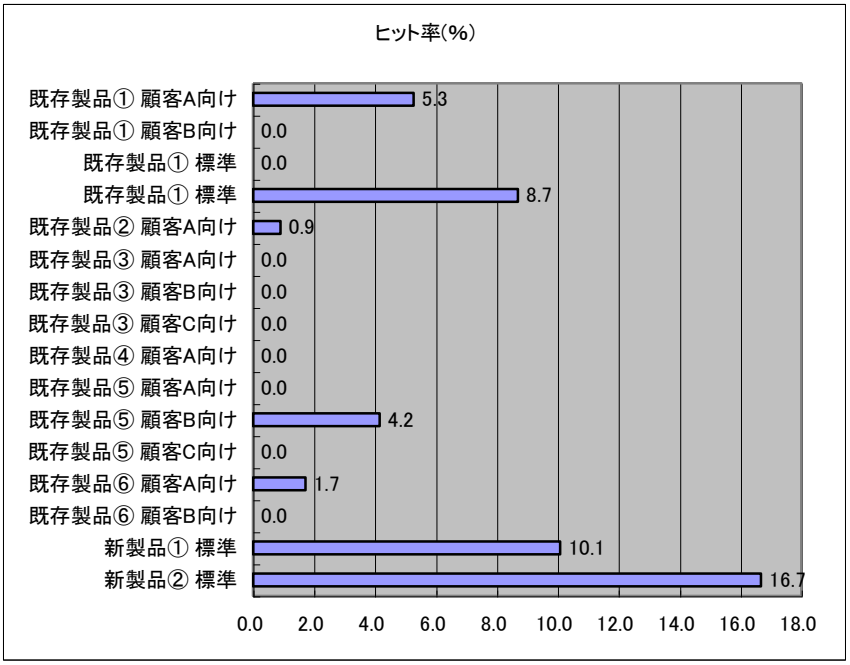
1. チェックリストの分析

1.1. チェックリストの利用状況

現状のチェックリストは，チェック項目を羅列したものであり，目的や観点を理解した上で利用することが前提になっている．そのため，知識や経験のある知識者はチェックリストを有効に利用できているが，経験やスキルのない無知見者はチェックリストを有効に利用できていない．そこで，これらの問題点と課題を明らかにするために，研究員が利用している設計チェックリストを収集し分析した．

まず，各プロジェクトにおける設計チェックリストのヒット率※を調べた．設計チェックリストには「設計時の定石（注意点）」と「過去の不具合の教訓」があるが，過去の不具合の教訓が現状のプロジェクトにどのように活かされているのかを把握するため，「過去の不具合の教訓」に着目して利用状況を分析した．その結果，既存製品を元にした標準や顧客向けの開発（派生開発）の利用率は 4～5%であるが，新製品開発では10%を越えおり，派生開発のチェックリストの利用率が新製品開発よりも低いことが分かった．

※ヒット率＝今回の開発で関係ありと判定したチェック項目数／チェックリストの項目総数



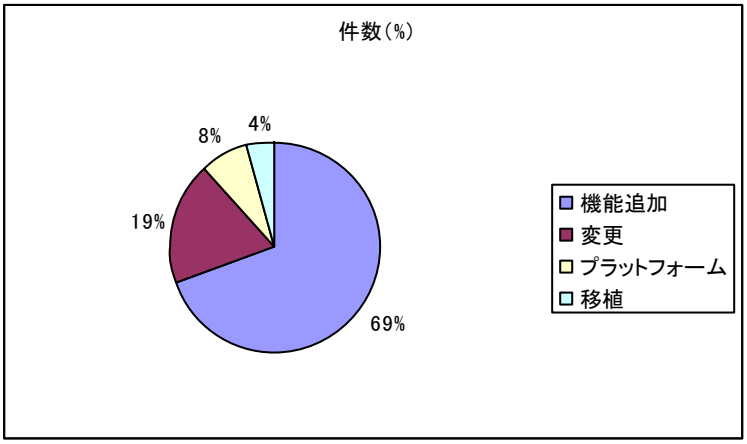
付図 1：各プロジェクトにおける設計チェックリストのヒット率

次に，記述内容からチェック項目を「新製品開発・派生開発：機能追加」，「派生開発：変更」，「新製品開発：プラットフォーム」，「派生開発：移植」の 4 種類に分析した．その結果を「付表 4:チェックリスト項目の分類」に示す．

ソフトウェア開発現場の不具合事例を分析した結果によると，「追加機能，変更機能が動作しない（新規バグ）」と「既存機能が動作しなくなった（デグレード）」は同程度で発生している．再発防止としてひとつの不具合にひとつのチェック項目があると仮定すると，「新製品開発・派生開発：機能追加」の件数と「派生開発：変更」と「派生開発：移植」を合計した件数は同程度と想定していたが，「派生開発：変更」と「派生開発：移植」を合計した件数は「新製品開発・派生開発：機能追加」の件数の約 1/3 であった．

付表 4：チェック項目の分類

No	分類	用途	記述内容	件数
1	新製品開発 派生派生開発: 機能追加	新製品開発において他の製品と同じ機能を新規設計する場合に利用する. または，派生開発において追加機能を設計する場合に利用する.	○○機能を新規作成/追加した場合，△△とする. △△には具体的な注意事項や設計方法が記述されている.	69%
2	派生開発: 変更	派生開発において仕様変更する場合のデグレード防止に利用する.	○○の仕様を変更した場合，△△とする. △△には具体的な注意事項や設計方法が記述されている.	19%
3	新製品開発: プラットフォーム	新製品開発においてプラットフォームの設計に利用する.	新規開発の場合，設計の定石（初期化処理，異常処理など）を確認する.	8%
4	派生開発: 移植	派生開発において他の製品の既存ソースコードを再利用する場合に利用する.	既存のソースコードを再利用する場合，△△を確認する. △△には具体的な注意事項や設計方法が記述されている.	4%



付図 2:チェック項目の分類

更に，「派生開発：変更」と「派生開発：移植」の件数が少ない要因について分析を行った．その分析結果を「付表 5:「派生開発：変更」と「派生開発：移植」の件数が少ない要因」に示す．

付表 5：「派生開発：変更」と「派生開発：移植」の件数が少ない要因

No	要因	問題点
1	「派生開発：変更」に関する不具合（デグレード）の教訓を製品共通のチェックリストに残すためには，チェック内容に注意点や回避策などを具体的に記述する必要があるが，製品共通のチェックリストに具体的なものが増えると，チェックリストが膨大となるため，製品個別のチェックリストに残している.	チェックリストにデグレードを防止するための注意点や回避策などを具体的に記述すると，チェックリストが膨大となり，チェックリストを利用する時の負荷が高くなる.
2	「派生開発：変更」に関する不具合（デグレード）の教訓を，製品共通のチェックリストに残すためには，応用性を向上させるためにチェック内容を抽象化するが，既存のチェックリストに抽象度の高いチェック内容（例えば，ハードウェア変更時は，ソフトウェア全体を調べてドライバを設計する．）があると，チェック内容は抽象度の高いものに集約される.	知識や経験のある有知識者はチェックリストを有効に利用できているが，経験やスキルのない無知見者がチェックリストを有効に利用できていない.

1.2. チェックリストの表現の分析

経験やスキルのない無知見者がチェックリストを有効に利用できていない問題点を把握するために，「変

更」と「移植」のチェックリストの表現について分析した. その分析結果を「付表 6:「変更」と「移植」のチェックリストの表現の問題点」に示す.

付表 6:「変更」と「移植」のチェックリストの表現の問題点

No	分類	区分	目的の記述	チェックリストの内容	問題点
1	変更	ファイル	なし	ファイル変更時, そのファイルをアクセスしている処理をすべて洗い出し, 改造が必要かどうか設計する.	無知見者にとっては, チェック項目に抽象度が高いものがある. そのため, 変更内容に照らし合わせた場合, 具体的に何に対応させたらよいのか分からない.
2	変更	ドライバ	あり	ハードウェア変更時は, 特定のハードウェア構成時に異常が起きる作り込みを防止するため, ソフトウェア全体を調べてドライバを設計する.	同上.
3	変更	機能仕様	あり	○○機能を変更する場合, △△処理が遅くなることを防止するため, ○○機能のすべてのパターンについて設計する.	問題なし (無知見者であっても気づける)
4	変更	初期化処理	あり	初期化処理に関数を追加する場合, 関数の参照するデータの種別(初期値／前回値)を間違えないため, コールドスタート時に呼び出されるのか, ウオームスタート時に呼び出されるのかを確認する.	同上.
5	変更	画面遷移	なし	○○機能を変更する場合, △△の画面遷移を確認する.	無知見者にとっては, チェック項目の内容を理解できないものがある. そのため, 変更内容に合致したチェック項目であることを認識できない.
6	移植	移植設計	あり	既存製品からソフトウェアを移植する場合, 未完成の機能を利用することで, 不完全な処理を含む作り込みを防止するため, 移植元のドキュメント調査, 設計者へのヒアリングを行う.	項目が具体的なレベルで捉えられていない.

チェックリストの表現を分析した結果では, チェック項目の抽象度が高いこと, チェック項目の目的が分からないことが主な原因であることが分かった.

1.3. チェックリストの問題点と真の原因

51 項の分析結果より, 現状のチェックリストの問題は 3 点, 真の原因は 2 点に集約された.

付表 7: チェックリストの問題点と真の原因

No	問題点	真の原因
1	チェックリストにデグレードを防止するための注意点や回避策などを具体的に記述すると, チェックリストが膨大となり, チェックリストを利用する時の負荷が高くなる. そのため, 変更内容に必要なチェックリストを選択するのに時間がかかる.	膨大なチェック項目の中から必要なチェック項目を効率良く見つけ出すしくみがない.
2	無知見者にとっては, チェック項目に抽象度が高いものがある. そのため, 変更内容に照らし合わせた場合, 具体的に何に対応させたらよいのか分からない.	何を変更して発生した不具合であるのかがわかる情報をチェック項目に追加するしくみがない.
3	無知見者にとっては, チェック項目の内容を理解できないものがある. そのため, 変更内容に合致したチェック項目であることを認識できない.	

付録 C

付表 8：不具合事例への気づきナビの使用シミュレーション結果　(1/2)

No	不具合現象	変更内容	変更内容分類(1次)	変更内容分類(2次)	真の原因	直接原因分類(1次)	直接原因分類(2次)	変更特性	変更特性チェックリスト	デグレードを防止への寄与が見込める？
B-01	追加した参照処理(共通関数)のパラメータに，期待していたデータ範囲を下回る値が渡され，エラーになった.	既存の共通処理を呼び出す処理を追加した.	ソースコード処理	既存の共通処理を呼び出す処理を追加	データの範囲設定間違い	データ	値（値の範囲）	関数呼び出しを追加	呼び出し元がパラメータとして渡すデータの範囲と，呼び出し先で期待しているデータ範囲が合致していることを確認する.	見込める
B-02	Windows サービスの再インストール時に変えた設定で，システムの一部機能が正常に動作しなくなった.	Windows をサービスの再インストールした.	動作環境	Windows サービス パラメータ設定	Windows サービスのパラメータ設定漏れ	動作環境	Windows サービス パラメータ設定	Windows サービスのインストール	Windows サービスを利用する時は，サービスが動作する時のオプションの有無を確認する. オプションがある場合は，設定値，設定手順を確認する.	見込める
B-03	コードの桁数を変更したところ，桁数依存だった処理が動かなくなった.	コードの桁数を変更した.（キー項目の桁数変更）	データ	データ構造	追加機能で不具合が発生するデータ構造が入り込んでいることに気づけない	データ	データ構造	データの範囲変更	データの取りうる範囲を変更する場合，データを参照している先の処理に影響がないことを確認する.	見込める
B-04	○○ツール(VB アプリケーション)の画面を表示した所，「メモリ不足」とのエラーになった.	アプリケーションの利用しているコンポーネントが（知らないうちに）バージョンアップされていた.	動作環境	Windows コンポーネント	アプリケーションの使っている Windows コンポーネントに気づけない	動作環境	Windows コンポーネント	Windows コンポーネントのインストール	Windows アプリケーションは実行環境に必要なコンポーネントを管理していること. コンポーネントの種類，バージョン，インストーラが分かっていること.	見込める
							Office コンポーネント	Office コンポーネントのインストール	実行環境に Office 標準コンポーネントがインストールされていることを確認する. バージョンも確認する.	

付表 8：不具合事例への気づきナビの使用シミュレーション結果　(2/2)

No	不具合現象	変更内容	変更内容分類(1次)	変更内容分類(2次)	真の原因	直接原因分類(1次)	直接原因分類(2次)	変更特性	変更特性チェックリスト	デグレードを防止への寄与が見込める？
B-06	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.	既存のデータ(内部メモリ)を使って新しい機能を追加した.	ソースコード処理	既存のデータを参照する処理を追加	内部メモリで保持している情報の範囲の認識間違い(制約に気つけない)	データ	値（値の範囲)	既存のデータを参照する処理を追加	元々存在するデータを参照して新しい機能を作成する場合，データの取り得る範囲を確認し，処理を作成する.	見込める
B-09	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.	同期処理	関数内にファイルをロックする処理を追加	関数内にファイルロック処理を追加した時に呼び出し元でファイルロックしていることに気づけない	同期処理	マルチロック	関数内にファイルをロックする処理を追加	関数内にファイルをロックする処理を追加する場合，呼び出し元でファイルをロックしていないかチェックする．ロックしている場合は，マルチロックとなるように処理を変更する. ※マルチロックとは，ロックが必要な複数のファイルがある場合に，ファイルを順にロックしていき，ひとつでもロックできないファイルがある時は，取得済みのすべてのファイルロックを一旦開放するという一連の処理繰り返すことにより，最終的にすべてのファイルロックを取得する方法である.	見込める

付表 9：変更特性チェックリストによるチェック項目総数の削減効果

プロジェクト	カテゴリで 確認した時の チェック項目総数　X	変更特性で 確認した時の チェック項目総数　Y	チェック項目総数の 削減効果 (1－Y／X)＊100
熟練技術者(経験 15 年)	63 項目	52 項目	17%
非熟技術練者(経験 1 年)	63 項目	54 項目(内 2 は選択ミス)	14%

※変更要求仕様書の仕様数:21 件

付録 D

1．変更特性マトリックスの作成

(1) カテゴリ・変更特性の検討

当該製品の派生開発で実際に発生した不具合の原因となった変更特性を，過去の不具合情報から抽出する．抽出する変更特性は，具体的で他の変更仕様においてもキーワードとして抽出されやすいものを選択する必要がある．また，新たに不具合が発生し，そこで抽出された変更特性がこれまでのカテゴリ・変更特性に分類できない場合は，新たなカテゴリ・変更特性を追加することで対応する．

変更特性マトリックスは，特定の組織・プロジェクトの中で繰り返し利用し，その都度，カテゴリや変更特性の種類・表現を見直すことで，より実際の組織・プロジェクトにフィットしたものになっていくことが期待できる．

付表 10：変更特性マトリックスの例

			変更特性	・Wakup/Sleepの処理変更□	・Wakup/Sleep割り込み処理変更	・異常判定追加(リセット条件追加)	・移植(共通Libの流用もしくは変更)	・移植(データ、配列などを利用するロジックを流用)	・移植(イニシャル処理を流用)	・イニシャル処理の追加および変更	・イニシャル処理時のCPUレジスタ/入出力ポート設定変更	・グローバル変数のアクセス権限変更	・グローバル変数の追加および変更	・グローバル変数の読み出し/書き込み	・グローバル変数の排他制御追加および変更	・時刻管理処理変更	・タイマ 割り込み周期変更	・タスク間通信メッセージ、シーケンス変更	・通信メッセージ、通信シーケンス 追加および変更	・データ、フラグ追加	・データ追加□	・データ追加	・データ範囲変更	・不揮発性メモリのデータ追加	・モジュール追加	・モジュール変更	・移植	・移植(処理タイミングを流用)	・割り込み処理変更	・関数パラメータ 追加および変更	・関数呼び出し追加	・関数追加	・関数戻り値追加および変更	・揮発性メモリのデータ読み出し/書き込み処理追加および変更	・共通データ変更(共通データ経由でデータの受け渡し)	・周辺 SIOのI/F変更(読み出し/書き込みタイミング、データ量の変	・周辺 SIOの変更(不揮発性メモリ、ADCなどの変更)	・周辺チップ(不揮発性メモリ、ADCなど)変更 (読み出し、書き込みタイミング、データ量の変更)	・処理構造　ステップが多い処理の追加	・処理構造　ループ追加(forループ、whileループなど)	・同期待ちが発生する関数呼び出し追加	・不揮発性メモリのHW変更
要求	XXXX	○○の範囲を△△から□□に変更する。																																									
	XXX-01	○○の条件判断は『XX以下ならばAAとする』としていたが、『YY以下ならばAAとする』ように変更する。																																									

2. 変更特性チェックリストの作成

変更特性チェックリスト，既存のチェックリストに変更特性情報を追加することで作成する.

付表 11：変更特性チェックリストの例

No	大項目	小項目	変更特性	目的	チェック項目
1	モジュール追加，モジュール変更，移植	～	・モジュール追加 ・モジュール変更 ・移植	・母体の既存バグ，潜在バグが派生製品に引き継がれてしまうことを防止するため	・母体製品の開発時に発生した不具合リストを調査し，本製品の品質目標に適用する.
2	モジュール追加，モジュール変更，移植	～	・モジュール追加 ・モジュール変更 ・移植	・母体の制約が派生製品に引き継がれてしまうことを防止するため	・流用母体や一部流用機能に，制限事項や前提条件，使用上の注意などがある場合，本製品の制限事項とするのか，回避するのか，明確にする.
3	移植	～	・移植 （データ，配列などを利用するロジックを流用）	・データ，配列などを利用するロジックを流用した場合，データの不整合を防止するため	・データ，配列などを利用するロジックを流用して，データの不整合が発生しないか確認/検討する.
4	移植	～	・移植 （処理タイミングを流用）	・処理タイミングを流用した場合，タイミングなどの問題を防止するため	・処理タイミング（イベント通知，タスク間通信など）を流用して，タイミングなどの問題がないか確認/検討する.
5	移植	～	・移植 （共通 Lib の流用もしくは変更）	・共通 Lib を流用もしくは変更する場合の既存のデグレードを防止するため	・共通 Lib を流用もしくは変更する場合，新規，既存両方の問題がないことを確認する.
6	モジュール追加，モジュール変更，移植	～	・モジュール追加 ・モジュール変更 ・移植	・モジュールを追加および更新した場合のデグレードを防止するため	・モジュールの影響範囲が分かる関係図を準備する. ・追加または変更したモジュールへ影響あるモジュールで構成する機能を明確にする.
7	モジュール追加，モジュール変更，移植	～	・モジュール追加 ・モジュール変更 ・移植	・担当者の変更があった場合，各工程の初期と末期で，I/F で接点があるモジュールに修正が発生した場合など，修正した内容に不整合が発生することを防止するため ・フラグ ・データ ・イベント通知 ・タスク間通信 ・I/F 呼び出し など	・同時期に修正されるモジュールやお互いに関わるインターフェースを明確にする. ・修正され，影響があるモジュール，インターフェースの組み合わせを確認する.
8	HW 変更	～	・イニシャル処理時の CPU レジスタ/入出力ポート設定変更 ・周辺 LSI の変更（不揮発性メモリ，ADC などの変更） ・周辺 LSI の I/F 変更（読み出し/書き込みタイミング，データ量の変更）	・HW の変更による SW の変更モレ／変更ミス，デグレードを防止するため	・タイミング，データ量などを明確にする. ・吸収方法，影響範囲を明確にする. ・過去の変更時の不具合リストを調査し，本製品の品質目標に適用する.

9	HW 変更	～	・周辺 LSI の変更（不揮発性メモリ，ADC などの変更） ・周辺 LSI の I/F 変更（読み出し/書き込みタイミング，データ量の変更）	・以前からわかっている HW トラブル対応が引き継がれないことを防止するため	・以前から存在する HW の制限事項がそのまま引き継ぐ場合，対策済であることを確認する．
10	Reset	異常系	・データ，フラグ追加 ・関数パラメータ追加および変更 ・関数追加	・定義しているデータ，フラグが規定外に化けた場合の異常処理の作り込みモレを防止するため	・パラメータの整合性を確認する． ・チェックの結果，不整合が発覚した場合のシナリオ（動作停止 or 再起動～再復旧）を明確にする． ・製品のポリシーとして，リセットの発生条件，停止条件を明確し，それに則るようにする．
11	Reset	異常系	・異常判定追加（リセット条件追加） ※判定順序に注意 ※OR 条件，AND 条件に注意	・過去のトラブルから，リセットの発生条件などを見直している出力側と整合を合わせるため	・想定されるリセットが発生する条件，理由が明確になっている資料を準備する． ・過去のトラブルから，発生条件を見直す． ・どのように検証するか手順を決める．
12	Reset	異常系	・通信メッセージ，通信シーケンス追加および変更	・不完全なデータ受信，異常時の処理方式としてリセット以外の処理（受信データ破棄，処理保留など）の検討モレを防止するため	・リセット以外の処理（受信データ破棄，処理保留など）があるかを確認できる資料を準備する． ・過去のトラブルから，発生条件を見直す．
13	Reset	異常系	・異常判定追加（リセット条件追加） ・周辺チップ（不揮発性メモリ，ADC など）変更（読み出し，書き込みタイミング，データ量の変更）	・想定しないリセット（WDT，ノイズリセット），処理途中の緊急停止（電源断含む）やキャンセルにより，揮発性メモリ（データや配列）が初期化されてしまうことを防止するため ・複数ノードが連携して動作している場合，中間ノードがリセットすることで，ノード全体として，通信シーケンス，揮発性メモリ（データや配列）が不整合となることを防止するため	・状態遷移条件に抜けモレがないことを確認する． （緊急停止，処理キャンセルのときの遷移は特に注意） ・復帰処理を明確にする． ・復帰できない場合に，他装置が感知するのか，自分でリセットをかけて，復旧するなどの処理を明確にする． ・現時点で想定していない異常な状態になった場合（その他トラブル）の処理を明確にする． （HW/SW になんらかの異常が起きるから，想定していない状態になる．そういった場合，無応答やとんでもない値になったりするので，その場合の処理を明確にすること）
14	定型シーケンス	初期化	・移植（イニシャル処理を流用）	・イニシャル処理を流用した場合，HW 初期化（レジスタ，ポート，不揮発性メモリ，周辺チップなどの初期化），データ初期化，データ復元といった，初期化シーケンスの変更モレ／変更ミスを防止するため	・前任機との変更点を明確する． ・電源投入時に初期化する項目と不揮発性メモリから読み込む項目の存在有無を明確にする． ・変更箇所の局所確認と初期化シーケンスを通した状態を確認する．
15	定型シーケンス	初期化	・データ追加 ・イニシャル処理の追加および変更	・イニシャル処理における，想定しないリセット（WDT，ノイズリセット），処理途中の緊急停止（電源断含む）やキャンセルにより，イニシャル処理の状態遷移がひとつの状態に固定されてしまうことを防止するため	・データ化けが起きている場合の起動処理を明確にする． ・起動中の停止など，一度正常以外の立ち上げ方をしたあとでも，正常に立ち上げることができるようにする． ・起動中の停止などで，データ書

					き込み異常などが起きている場合の起動処理を明確にする.
16	定型シーケンス	Wakeup/Sleep	・データ追加	・Sleep 前と後とでデータが不一致となることを防止するため	・Sleep 前に保存するデータと比較し，過不足が発生しないロジックにする.
17	定型シーケンス	Wakeup/Sleep	・Wakup/Sleep の処理変更	・Sleep 移行処理の変更モレ／変更ミスにより，Sleep モードに移行できず，電池が消耗してしまう作り込みを防止するため	・Sleep 状態へ移行する条件，時間などを明確にする.
18	定型シーケンス	Wakeup/Sleep	・Wakup/Sleep 割り込み処理変更	・Wakup/Sleep の割り込み処理の確認モレを防止するため	・Wakup 状態へ移行する条件，外部要因（割り込み，イベントなど）を明確にする.
19	定型シーケンス	割り込み処理	・割り込み処理変更 ※タイマ割り込みに注意	・割り込み処理の割り込み禁止，解除の変更ミスを防止するため	・割り込み時，割り込み解除，割り込み禁止の処理を明確にする.
20	定型シーケンス	割り込み処理	・割り込み処理変更 ※タイマ割り込みに注意	・割り込み処理の処理時間変更により，システム動作が劣化することを防止するため ・多重割り込みにより，割り込み中の割り込みにより，割り込み時間を保証できなくなる作り込みを防止するため	・優先順位と戻り処理を明確にする.
21	定型シーケンス	割り込み処理	・割り込み処理変更 ※DMA 割り込み，シリアル通信の送信割り込み／受信割り込みなどに注意	・多重割り込み（割り込み中の割り込み）により，割り込み処理が規定時間よりも長くなる作り込みを防止するため	・多重割り込みを考慮する.
22	定型シーケンス	割り込み処理	・Wakup/Sleep 割り込み処理変更	・Wakup/Sleep の割り込み処理の確認モレを防止するため	・Sleep 状態へ移行する条件，時間などを明確にする. ・割り込み時，割り込み解除，割り込み禁止の処理を明確にする.
23	タイミング	～	・ループ追加（for ループ，while ループなど） ・ステップが多い処理の追加 ・同期待ちが発生する関数呼び出し追加	・割り込み時間が長くなることで，システム動作を保証できなくなることを防止するため ・タスク処理の同期待ちが長くなることで，システム動作のパフォーマンスが劣化することを防止するため	・処理時間に影響がある既存モジュールの存在有無を示す資料を準備する. ・存在する場合，シーケンス図など，処理タイミングを検討した資料を準備する.

24	タイミング	～	・ループ追加（for ループ，while ループなど） ・ステップが多い処理の追加 ・同期待ちが発生する関数呼び出し追加 ・タイマ割り込み周期変更	・割り込み処理に処理時間のかかる処理を追加した場合，割り込み時間が規定時間以上に長くなり，システム動作を保証できなくなることを防止するため ・タスクに処理時間のかかる処理を追加した場合，スケジューラのディスパッチが規定時間以上に長くなり，システム動作を保証できなくなることを防止するため	・モジュールを更新したことで，モジュールの処理時間がどのくらいになったか，予測もしくは実測する． ・変更した処理時間のメインループへの影響がないことを確認する． ・カウントタイミングに(増減の)マージンを取ること．
25	タイミング	～	・時刻管理処理変更	・RTC がなく，通信で外部から時刻を取得する場合，通信断などにより，時間が遅れた場合の復旧処理のモレを防止するため	・同期ズレを補正する必要があるかどうか明確にする． ・補正した結果の影響を考慮する． ・時間などが前に戻る処理で，問題が生じるかどうか考慮する． ・補正条件を定義する．
26	I/F	～	・データ追加 ・関数呼び出し追加 ・関数パラメータ追加および変更 ・関数戻り値追加および変更	・追加した変数を関数のパラメータにする場合，パラメータの参照方法が間違っていることで，他のデータ領域を壊してしまう（データを書き換えてしまう）ことを防止するため	・返却値を引数として渡されたモジュールの引数が仕様と合致していることテストする．
27	I/F	～	・データ追加 ・関数呼び出し追加 ・関数パラメータ追加および変更 ・関数戻り値追加および変更	・関数のパラメータ追加および変更，関数の戻り値追加および変更の場合，境界値，異常値，データ化けなどの試験モレを防止するため	境界値，異常値，データ化けなどをテストする．
28	I/F	～	・データ範囲変更 ・データ追加	・データ範囲変更，データ追加などの場合，有効データ範囲，メモリサイズを超えることを防止するため	・コンパイラのチェックツールを利用する． ・可能であれば，子プロセスや，さらに受け渡し先のプロセスで不正なデータとならないかチェックを行う．
29	I/F	～	・タスク間通信メッセージ，シーケンス変更 ・共通データ変更（共通データ経由でデータの受け渡し）	・データ範囲変更，データ追加などの場合，有効データ範囲，メモリサイズを超えることを防止するため	・コンパイラのチェックツールを利用する． ・可能であれば，子プロセスや，さらに受け渡し先のプロセスで不正なデータとならないかチェックを行う．
30	データ	平行実施	・グローバル変数の読み出し/書き込み ・グローバル変数の追加および変更 ・グローバル変数のアクセス権限変更 ・グローバル変数の排他制御追加および変更	・メモリ領域へのアクセス権限がある場合，アクセス権限の問題でデータの書き込み/読み出しができなくなる作り込みを防止するため	・アクセス権限の基準が策定され，どの定義が適用されるか明確にする．
31	データ	既存機能への影響	・グローバル変数の読み出し/書き込み ・グローバル変数の追加および変更 ・グローバル変数のアクセス権限変更 ・グローバル変数の排他制	追加および更新したモジュールにより，他の機能のデグレードを防止するため	・データシーケンス図，タイムチャートを準備する． ・追加モジュールリストを準備する．

			御追加および変更		
32	データ	既存機能への影響	・グローバル変数の読み出し/書き込み ・グローバル変数の追加および変更 ・グローバル変数のアクセス権限変更 ・グローバル変数の排他制御追加および変更	追加および更新したモジュールにより，他の機能のデグレードを防止するため	・追加モジュールの動作タイミングと競合する対象（モジュール，メモリ空間，データ）を明確にする. ・同時書き換え，同時利用を確認する. ・同時書き換えがないことが望ましいが，もし発生した場合は動作を明確にする（例えば，リセットなど）
33	不揮発性メモリ	～	・不揮発性メモリの HW 変更 ・揮発性メモリのデータ読み出し/書き込み処理追加および変更	・不揮発性メモリへの書き込み/読み出しが集中することで，データ保存/復元が正常に行われないことを防止するため	・過負荷時や他モジュールの処理遅延によるタイミングのずれを考慮する. ・同時読み出し，同時書き込みの対応を考慮する. ・データが読み出せない，書き込めない（壊れていた場合）の対応を考慮する.
34	不揮発性メモリ	～	・不揮発性メモリのデータ追加	・不揮発性メモリへのデータ格納場所がずれることで，揮発性メモリのデータが不揮発性メモリにデータ保存後，データ復元すると不一致となることを防止するため（書き込み位置と読み出し位置が一致していること）を保証するため	・追加したモジュールの不揮発性メモリへのアクセスが存在するかどうかを明確にする. ・データサイズとアドレスを確認する. ・新規に使用する領域が未使用領域を使用していることを確認する.

変更特性チェックリスト作成ガイド

・デグレードの具体事例をチェック項目にする.
・チェック項目には注意点や回避策（変更方法）を具体事例で記述する.
・デグレードの現象に注目するのではなく，デグレードの原因となった変更方法（What，Where，How）に注目する. What はファイル，タスク，ハードウェアなどの具体事例を記述する.
・意図，目的がないと，何のためにチェックするのか理解できないため，チェック項目には意図，目的を記述する.
・母体のアーキテクチャの問題，部分理解による判断ミス，判断漏れなどを明確にするため，網羅性を追求するためのチェック項目は追加しない.
・同じデグレードを再発させないため，チェック項目は追加のみとする.

「気づきナビ」の効果を高めるポイント（参考）

「変更特性マトリックス」のカテゴリと変更特性

・組み込み系の場合：プラットフォームの構成は固定されており，アプリケーション数も少ないため，カテゴリと変更特性を一緒にした「何をどのように変更したのか」を変更特性としたほうがよい.
・エンタープライズ系の場合：OS やミドルウェア，データベースなどプラットフォームの構成は複雑であり，

アプリケーション数も多いため，カテゴリを階層化したほうがよい．

「変更特性チェックリスト」のチェック項目

・チェック項目には注意点や回避策（変更方法）を具体的事例で記述した方が，非熟練技術者にとって理解しやすい．

・チェック項目を作るときは，デグレードの現象に注目するのではなく，デグレードの要因となった変更方法（What, Where, How）に注目して作成した方が，変更特性マトリックスとの繋がりが明確になる．