Vol.2021-SE-208 No.18
2021/7/9

派生開発のためのリバースエンジニアリング管理手法

井口 一騎[1]　中西 恒夫[2]　久住 憲嗣[3]

概要：我が国の開発現場で広く実践されている派生開発プロセス XDDP（eXtreme Derivative Development Process）では，既存ソフトウェアに対する機能追加や変更に係る部分をリバースエンジニアリングする工程は設けられてはいるものの，リバースエンジニアリングを体系的，かつ必要最小限で実施する方法論は述べられていない。本稿では，明らかになっていない既存ソフトウェアのつくりを想定したうえで変更要求仕様を記述し，そのような想定変更要求仕様を記述するうえで立てた仮説を書き出し，その仮説を検証することで変更要求仕様書を継続的に修正して完成させる，ゴール指向のリバースエンジニアリグの方法論を提案する。ケーススタディによる評価では，提案手法によって派生開発に要する全体の工数を削減することはかなわなかったが，手戻りやバグ修正に要する工数が絶対値でも割合においても大幅に抑えられ，工数のほとんどが派生製品のソフトウェア開発に直接的に必要な作業に使われたことが確認できた。

A Management Method of Reverse Engineering for the Extreme Derivative Development Process

Abstract: XDDP (eXtreme Derivative Development Process), which is widely accepted by domestic companies, defines a sub process to conduct reverse engineering for addition and modification to existing software, although it does not provide a scheme to do reverse engineering in a systematic and minimal manner. This paper proposes a goal oriented scheme for the reverse engineering. In the scheme, developers describe specification on existing software modification whose insides are not well understood with speculation, elicit hypotheses established to describe the specification, and correct the specification consecutively with validating the hypotheses. As the results of a case study, the scheme could not reduce total man-hours used for derivative development; however, it can guide developers to use man-hours only for development activities directly needed to realization of the derivative product and reduce man-hours used for rework and bug fix considerably.

1. はじめに

今日のソフトウェア開発現場，特に組込みソフトウェアの開発現場において新製品のソフトウェアをスクラッチから開発することは稀である。新製品に最も近い既存製品のソフトウェアを再利用し改修する，いわゆる派生開発を行うことが一般的である。派生開発のためのプロセスのひとつとして，清水によって提唱された XDDP（eXtreme Derivative Development Process）[1] は，我が国の組込みソフトウェアの開発現場で広く支持され，実製品の開発における多くのケーススタディが報告されている [2], [3], [4]。XDDP は，派生製品を開発するにあたり，既存ソフトウェアに対する機能追加と変更に関する要求を洗い出し，それら要求を実現するべくどこをどのように変更するのかを調査したうえで，機能追加と変更に関する仕様を定義する。既存ソフトウェアの全体像を把握することなく，機能追加と変更に関する部分の理解に集中することで派生製品開発に要する工数を抑えるプロセスである。

しかしながら，工期の短縮に大きく寄与し得るように思える派生開発も既存ソフトウェアの調査には相当の工数を費やし得る。派生元となる既存製品の要求からコードへ至るまでのドキュメント，すなわち要求仕様書や設計書がトレーサビリティが保証されたかたちで揃っている開発現場は決して多くはない。ドキュメントが残っていてもそれらの間のトレーサビリティが不十分であったり，齟齬があったり，さらに悪い場合には信頼できるドキュメントはコー

[1] 日産自動車株式会社
Nissan Motor Co., Ltd., Yokohama, Kanagawa, Japan
[2] 福岡大学
Fukuoka University, Fukuoka, Japan
[3] 芝浦工業大学
Shibaura Institute of Technology, Saitama, Japan

ドのみということすらある。また，ドキュメントに記された仕様や設計の意図が読み取れず，そのソフトウェアの開発者に尋ねようにも多忙や転職でそれもままならず，そのソフトウェアに関する知識が開発現場から失われていることもしばしばである。

XDDP では，こうした既存ソフトウェアを調査するリバースエンジニアリングの工程を「スペックアウト」と呼んでいるが，その方法論については具体的なものが示されていない。方法論を持たず欠陥混入の不安を持ちながらのリバースエンジニアリングは不用意な深入りとそれに伴う工数増大を招きがちである。また，リバースエンジニアリングは開発者の経験や勘に頼る私的な開発活動とされがちである。そうした開発体制ではリバースエンジニアリングの計画や過程が開発チームで共有されることもなくなるため，既存ソフトウェアに対して追加，修正された仕様や設計の根拠が記録される機会が失われ，将来，部分的に重複したリバースエンジニアリングが実施される無駄すら生じ得る。

このような XDDP におけるリバースエンジニアリングの問題を背景に，本稿では，開発者個人ではなく開発チームで実施する，体系立てられた既存ソフトウェアのリバースエンジニアリング管理手法を提唱し，ケーススタディによってその評価を行う。提案手法では，XDDP の変更要求仕様書を作成するにあたって，既存ソフトウェアがどのように動いているのかを想定した「想定要求仕様」を記述し，想定要求仕様に基づいてソフトウェアの構造や振舞いに関する「仮説」を導出する。想定要求仕様は派生製品のソフトウェアを開発するにあたって既存ソフトウェアに関するどのような知識が必要なのかを洗い出す役割を担う。想定要求仕様から導出，形式化される仮説をリバースエンジアリング管理のためのファーストクラスオブジェクトとすることで，リバースエンジニアリングの目的を明確にし，開発チームでリバースエンジアリングの過程を共有し，リバースエンジニアリングの範囲を無秩序に拡大することを防止する。

以下，本稿第 2 節では派生開発プロセス XDDP の概要を紹介し，そのリバースエンジニアリングにおける課題を述べる。第 3 節では，本稿で提案するリバースエンジニアリング管理手法のねらいとプロセスについて述べ，第 4 節ではケーススタディによるその評価と考察を述べる。第 5 節では関連研究を紹介する。最後に第 6 節で本稿を総括し，今後の課題について述べる。

2. 派生開発プロセス XDDP

本稿で前提とする派生開発プロセス，XDDP（eXtreme Derivative Development Process）の概要を図 1（原著 [1] より一部翻案）に示す。

XDDP では，派生製品開発のための既存ソフトウェアへの改修を大きく機能追加と変更に分離して捉え，それぞれのための工程を定義している。変更とは，派生製品の要求を実現するにあたり，既存製品の仕様，設計，実装，テスト等の成果物を変更することを意味する。一方，機能追加とは，派生製品の要求を実現するにあたり，既存製品の成果物を再利用するだけでは対応できず，新たに要求，仕様，設計，実装，テストを作成することを意味する。派生開発において機能追加がある場合はその追加分を既存ソフトウェアに受け容れるための変更が必ず生じる。

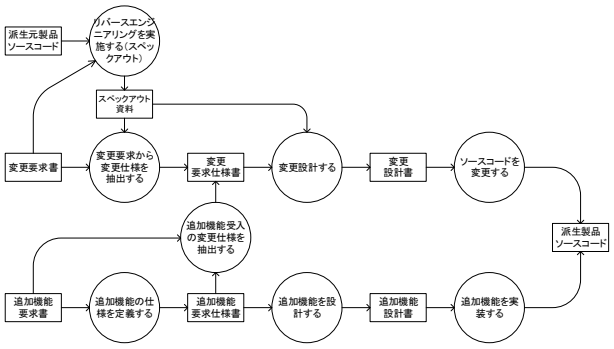

図 1 XDDP のプロセス概要

XDDP でもっとも重要な役割を担うドキュメントは変更要求仕様書である。変更要求仕様書は，顧客（あるいは企画）からの変更要求に対応するべく既存ソフトウェアの何をどのように変更するのか，開発チームでの合意を記述したドキュメントと見ることができる。変更要求仕様書は，USDM（Universal Specification Describing Manner）[5] による書式に則り，変更に関する要求と仕様を対にして記述する。記述の際には，差分を意識し，すなわち既存ソフトウェアの仕様の変更の前後を明らかにし，変更箇所の特定と合意形成を行う。

変更要求仕様書に添付するかたちでトレーサビリティマトリックスも記述される。トレーサビリティマトリックスは，変更要求仕様書に記載した変更仕様とそれに関係するソースコードの変更箇所（モジュールや関数，変数）の対応を記述した表である。トレーサビリティマトリックスにより，変更仕様を実現するためにどのモジュールに変更すべきなのかが一目瞭然となる。

変更要求を詳細化して変更仕様を抽出し，変更要求仕様書ならびにトレーサビリティマトリックスを記述するためには，既存ソフトウェアに関する知識が必要となる。そうした知識のうち明らかでないものについては，XDDP では，リバースエンジニアリングを実施してスペックアウト資料を書き出すべしとしているが，前節で述べたように，その具体的な方法論については与えられていない。しかし，著者らのソフトウェアプロダクトラインにおける産業界との経験 [6] からすれば，既存ソフトウェアのつくりを調査するこのリバースエンジニアリングの工程は，XDDP による開発工数の相当分を占める支配的なものとなるはず

である。本稿はこのリバースエンジニアリングの工程を議論の対象とする。

3. 派生開発のためのリバースエンジニアリング管理手法

3.1 提案するリバースエンジニアリング管理手法のねらい

本稿では，XDDP におけるリバースエンジアリングを実施する工程を体系化するリバースエンジニアリング管理手法を提案する。提案手法はリバースエンジアリングの結果のみならず，リバースエンジニアリングの目的，計画，過程，結果を追跡可能なかたちで可視化，記録する。これにより，i) これまで個人的な開発活動とされがちだったリバースエンジニアリングをチームによって計画，分担実施，レビューできる活動にする，ii) 目的のはっきりしない，あるいは目的を忘れた深いリバースエンジニアリングを避ける，iii) 私的に，あるいは暗黙的に知られていた知識をドキュメント化し，あるいは誤解や既存ドキュメントの誤りを正すことを図る。

図 2 は，既存ソフトウェアに関する開発チームの知識資産，ならびに開発者個人の理解を模式的に表したものである。派生開発対象となる既存ソフトウェアに関する「正しい」完全な知識の範囲（$A \cup B \cup C \cup D$）はいわば神のみぞ知るものである。既存ソフトウェアに関する開発チームの知識資産（$C \cup D \cup F \cup G$）と開発者個人の理解（$B \cup C \cup E \cup F$）はこれに重なるかたちで位置づけられる。開発チームのソフトウェア知識資産は，要求仕様書や設計書などドキュメント化されているもののみならず，ドキュメント化されることなく開発チームの多数のメンバによって暗黙的に合意，共有されている知識も含む。開発者個人の理解は，こうした知識資産に記述されているものもあれば，個人的にのみ理解されているものもある。

B は個人的には理解されているが，開発チームには知らされていない，共有モレとなっている知識である。D は開発チームで知られているのに，個人は知らない，理解モレとなっている知識である。

$E \cup F \cup G$ は既存ソフトウェアに対する誤った記述や理解である。E は個人的な勘違いや思い込み，無知による誤りである。$F \cup G$ は開発チーム全体での認識の誤りであったり，あるいはコードの改修後にドキュメントの修正が遺漏された結果，生じた誤りである。継続的に派生開発を行っている開発現場では，前者は稀であるが，後者はしばしば生じる誤りである。

開発において好ましくないのは C 以外の領域であり，A，B，D の領域は開発活動を通して減らす，E，F，G の領域は（本人たちは正しいと誤認していることであるので）誤認識の発見につながる施策を実施することがリバースエンジニアリングのプロセスを設計するうえで重要となる。

提案するリバースエンジニアリング手法では，リバースエンジニアリングの目的を明確にするゴール指向のアプローチにより，必要最小限の調査により，A の領域にある機能追加と変更に要する既存ソフトウェアの知識を抽出し記述することを図る（$A \rightarrow C$）。さらに，リバースエンジニアリングの結果だけをレビューするのではなく，その計画や過程を可視化，記録し，開発チームで共有することにより，個人的な活動とされがちなリバースエンジニアリングを開発チームで実施するプロセスに変える。これにより，私蔵されている既存ソフトウェア知識を開発チームのソフトウェア知識資産に変える（$B \rightarrow C$），誤った理解を正す（$E \rightarrow \emptyset$），理解していなかった知識を認識する（$D \rightarrow C$）ことを図る。また，リバースエンジニアリングにおける既存ソフトウェアの検証活動においては，既存ソフトウェアの知識資産のうち正誤が明らかになり，開発チームの誤った認識やソフトウェアの実態と齟齬を生じているドキュメントが正される効果（$F \cup G \rightarrow \emptyset$）が期待される。

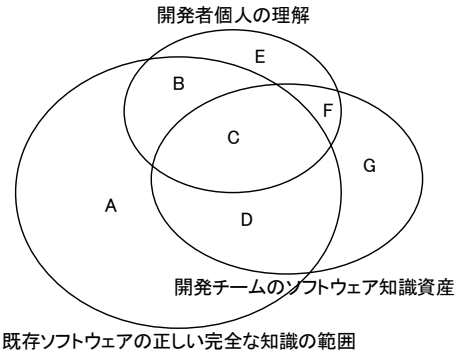

図 2 ソフトウェア理解モデル

3.2 リバースエンジニアリングプロセス

提案手法におけるリバースエンジニアリングのプロセスを図 3 に示す。このプロセスでは，リバースエンジニアリングを実施する前に，既存ソフトウェアに関する開発チームの知識資産と開発者自身の理解とを一致させ（$B \rightarrow C$，$D \rightarrow C$），そのうえで開発者自身が派生開発のために本当に必要とする知識（A の限られた範囲）を認識することによって，既存ソフトウェアについて部分理解すべき範囲を詳細化する。

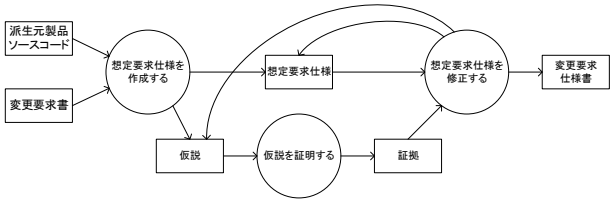

図 3 リバースエンジニアリングプロセス

想定要求仕様の作成：まず，変更仕様を作成するのに十分な知識が揃っていないことを承知のうえで，変更要求仕様書相当のドキュメントを記述し，これを「想定要求仕様」

と呼ぶ。リバースエンジニアリングを終えてから，あるいはリバースエンジニアリングをやりながら変更要求仕様を導出するのではなく，既存ソフトウェアに関する確たる知識のみならず想定や思い込みにすら基づいて，いわば裏付けのない変更要求仕様書を想定要求仕様として記述する。

開発者が既存ソフトウェアについてある程度よく知っていて，既存ソフトウェアの仕様が予測できるならば，想定要求仕様の記述はさほど困難なものとはならず，既存ソフトウェアの実態からもさほど乖離しない。しかし，そうではない場合，XDDP の変更要求仕様書に求められる水準に specify された想定要求仕様を記述することは容易ではない。そのため想定要求仕様の記述では，明らかになっていない既存ソフトウェアの仕様記述については不確実さや曖昧さを許す。その代わり，既存ソフトウェアについてどのような事柄が明らかになれば，それらを排除できるのかを把握できるようにしなければならない。そこで想定要求仕様の記述において，既存ソフトウェアについておそらくこのようになっているのだろうと想定した事柄を「仮説」として記述する。また，既存ソフトウェアのつくりが全くわからず，想定要求仕様を書けない場合については，これがわかれば既存ソフトウェアのつくりが想定できるようになるだろうと思われる事柄を「仮説」として記述する。

この工程のもうひとつの目的は，リバースエンジニアリングを実施する前に開発チーム内で想定要求仕様と仮説のレビューを行うことによって，既存ソフトウェアに関する開発チームの知識と開発者個人の理解を一致させること（$B \to C$，$D \to C$）である。

仮説の証明：既存ソフトウェアの知識に関する仮説を立てた後はリバースエンジニアリングを実施して仮説を証明する。仮説は想定要求仕様作成直後は不明の状態にあるが，リバースエンジニアリングによって証拠が集められ，真偽が明らかな状態となり，それが既存ソフトウェアの知識資産となる。漫然とリバースエンジニアリングをするのではなく，立てた仮説を証明するための活動としてリバースエンジニアリングを位置づけ，仮説の立証に寄与しない知識獲得を制限することでリバースエンジニアリングの工数増大を防ぐ。リバースエンジニアリングの過程で想定要求仕様の確定や修正のためのさらなる知識獲得が必要となった場合は，いきなりそのためのリバースエンジニアリングをするのではなく，それに関する仮説を改めて立ててから必要なリバースエンジニアリングを実施する。すなわち終始，リバースエンジニアリング工程を律するためのハンドルとして仮説を用いる。

証拠となるのは，ドキュメントやソースコードの解析結果，テストコードとその実行結果，仮説を証明するために行った実験のプロセスや結果などさまざまなものが考えられる。こうした証拠が開発チームで承認され仮説の真偽が決したとき，仮説に加えて証拠そのものもソフトウェア知識資産に加えられる。

この想定要求仕様を整理するために GSN[7] を用いる。GSN はシステムが達成しなければならない目的，機能，性質をどのように具現化するのか，その論証の主張，前提，戦略，証拠，そしてまだ論証されていないことを樹形図として体系的に記述する記法である。GSN の上位の主張を下位の主張に分解する表記に即して，上位の変更要求が下位の変更要求へ，あるいは想定要求仕様に分解される観点と過程を，GSN のゴールノードと戦略ノードを用いて記述する。リバースエンジニアリングが終わった後にこれらは USDM に転写され変更要求仕様書となる。さらに想定変更仕様で想定された事柄，つまり仮説が想定変更要求仕様のゴールノードの下位ゴールノードとして記述される。仮説は証明可能な粒度になるまで下位の仮説に分解される。最終的にリバースエンジニアリング活動による証明の結果，証拠が記録される。必要に応じて証明における前提が記述される。GSN は証明を重ねるたびに膨張していく傾向にあるため，適宜，仮説を論理的に統合したり，残す必要がない論証は削除する等して管理可能な規模に抑えるよう努める。最終的にできあがる GSN のうち，変更要求仕様書にされない範囲がドキュメント化されず失われがちであった既存ソフトウェアの知識資産であり，これを GSN のかたちで残すのが提案手法の遡及点である。

想定要求仕様の修正：その後，想定要求仕様の正しさを裏付けるようなリバースエンジニアリングを行い，裏付けのとれない事柄に関しては変更要求仕様の修正を行う。あらかじめリバースエンジニアリングを行う，もしくは同時進行で変更要求仕様を導出するのではなく，変更要求仕様を導出後にリバースエンジニアリングを行い，必要に応じて変更要求仕様の修正を行う。

仮説を証明する工程で獲得された既存ソフトウェアに関する知識に基づいて想定要求仕様を修正する。修正した想定要求仕様によって新たに仮説が生み出されることもあり得る。仮説が追加された場合は再びその証明を行い，その結果に基づいて想定要求仕様を修正する。仮説が新たに追加されず，仮説の証明の結果に想定要求仕様に対する反証が含まれない場合，想定要求仕様は変更要求仕様書として確定される。

仮説をすべて証明し，要求に対する実現可能性が保証されていることを確認した後，本来の XDDP のプロセスと同様にレビューを行う。この際に，変更要求仕様書だけではなく，リバースエンジニアリング時に構築した GSN を参照しつつ，レビューすることで，リバースエンジニアリングの過程をレビュワーやチームと共有することができる。そして担当者の要求仕様の意図の理解が容易になり，担当者の思い込みに気づくことやレビュワーやチームとの知識のすり合わせが可能になる。

4. ケーススタディ

提案するリバースエンジニアリング管理手法を評価すべく，実際のソフトウェアに対して XDDP による派生開発を施し，リバースエンジニアリングプロセスを含む各工程にかかった工数を比較，検討する。

4.1 題材

ケーススタディの題材として，オープンスタンダード命令セット RISC-V の基本整数命令セット RV32I[8] のエミュレータ，FuRISCV を用いる。FuRISCV は，第二著者によって Java で実装され，福岡大学工学部電子情報工学科におけるアセンブリ言語プログラミングを行う実験やコンパイラの講義等で実際に用いられている。図 4 に FuRISCV のダンプ画面を示す。FuRISCV では，RISC-V のレジスタや主記憶の内容を GUI 上でインタラクティブに参照，編集でき，主記憶上に置かれたプログラムをステップ実行し，レジスタやメモリが更新される様子を観察できる。

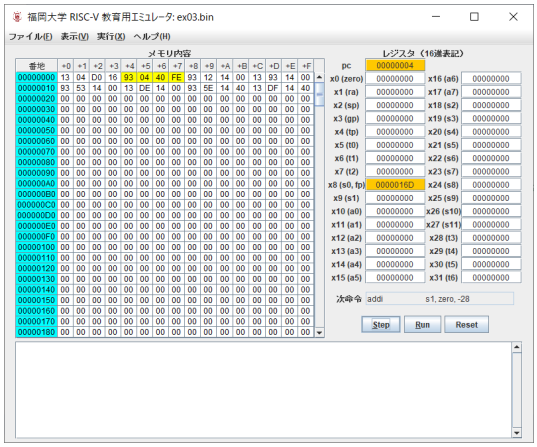

図 4 FuRISCV

4.2 ケーススタディのプロセス

ケーススタディでは，福岡大学工学部電子情報工学科に属する 2 名の被験者に上述の FuRISCV の XDDP による派生開発を依頼する。被験者の 1 名（以下，被験者 A）は情報工学を専攻する学部 3 年生（当時）であり，学科においてすでにソフトウェア工学と Java の講義，ならびウォータフォール型プロセスに基づくソフトウェア開発演習を履修している。もう 1 名（以下，被験者 B）は工学の学位を有する 30 代の情報系の教員であり，当然ながらソフトウェア工学と Java の知識と教育経験を有する。両名とも FuRISCV を演習や講義で使用した経験を有し，FuRISCV の使用方法についてはよく知っているが，その実装については全く知らない。

各被験者に依頼した，FuRISCV の派生開発要求は以下である。

> 派生開発要求
>
> 隣接する複数のメモリの番地を選択できるようにし，選択された範囲のメモリに対して逆アセンブル，挿入，削除をできるようにしたい。

この派生開発要求をさらに，XDDP で規定される USDM による変更要求仕様記述様式に従い，以下のような下位の派生開発要求に分割し，これらのそれぞれについて変更仕様を記述するところからケーススタディを開始する。

- 【下位派生開発要求 1】メモリブロックを複数選択し，操作を選択できるようにする。
- 【下位派生開発要求 2】選択されたメモリブロックに対して逆アセンブル，挿入，削除の操作を選択できるようにする。
- 【下位派生開発要求 3】逆アセンブルが選択された場合，選択された範囲のメモリブロックの内容に対応するアセンブリ言語命令文をコンソールに表示する。
- 【下位派生開発要求 4】挿入が選択された場合，0 で埋められたメモリブロックを選択された箇所に挿入し，選択された箇所以降のメモリブロックを挿入されたメモリブロックのバイト数だけ下にずらし，記憶可能なメモリアドレス数をはみ出した分は削除する。
- 【下位派生開発要求 5】削除が選択された場合，選択された箇所を削除し，選択された箇所以降のメモリブロックを削除されたメモリブロックのバイト数だけ上にずらし，メモリブロックの末尾に 0 で埋められたメモリブロックを削除されたメモリブロックのバイト数分挿入する。

被験者 A には，提案リバースエンジニアリング管理手法を用いた XDDP プロセスでの開発を依頼する。派生開発を実施している間，FuRISCV に関する知識や情報は一切与えないが，開発プロセスや各種開発資料の作成等に関する助言は与える。一方，被験者 B には提案リバースエンジニアリング管理手法を用いない XDDP プロセスでの派生開発を依頼する。

両被験者ともに，派生開発の各工程が終了するたびに XDDP の成果物である変更要求仕様書，変更要求仕様書とトレーサビリティマトリックス，変更設計書を提出するよう依頼する。さらにリバースエンジニアリング管理手法を用いる被験者 A には想定要求仕様と仮説を整理した GSN の提出も依頼する。コーディングは全工程の成果物の作成が終了した時点で行うように制限し，前工程への手戻りが発生する場合はその理由を報告するように求める。

4.3 結果

誌面の制約により掲載はしないが，両被験者が作成した

変更要求書，変更要求仕様書（トレーサビリティマトリクスつき），変更設計書に大きな差は見られなかった。しかしながら，被験者 B の変更要求仕様書には欠陥が見つかり，コーディング工程からの手戻りが発生した。

被験者 A が提案手法における仮説の整理，記述，証明に用いた GSN の一部を図 5 に示す。GSN の描き方として正しくない箇所が散見されるが，あえて被験者 A が作成したレビュー前のものを掲載する。被験者 A はもともと GSN に関する知識はなくこの GSN は初めて描くものであったが，おおよそ仮説立証の道筋が読めるものにはなっている。

工数に関する比較を表 1 に示す。表では，両被験者が変更要求仕様書作成，リバースエンジニアリング，変更設計書作成，コーディング，手戻り，バグ修正に費やした工数を比較している。手戻りとは，前工程の成果物に欠陥があり，それが後工程で見つかった際に，前工程に戻って成果物を修正することを意味する。バグ修正は，オリジナルの FuRISCV には今回の派生開発で無視できないバグがあり，その修正に要した工数である。多くの工程において被験者 A は被験者 B よりも多くの工数を費やしており，全体でも 550 分多くの工数を費やしている。しかし，手戻りとバグ修正については被験者 A は被験者 B よりも圧倒的に少ない工数で済んでいる。被験者 B の手戻りの原因は，要求仕様書作成時に FuRISCV のソフトウェアのつくりに関する知識が足りないことによるものであった。

表 1 工程別工数

	被験者 A	被験者 B
変更要求仕様書作成	6h05 (27%)	2h26 (18%)
リバースエンジニアリング	8h00 (35%)	5h44 (42%)
変更設計書作成	6h00 (26%)	0h40 (5%)
コーディング	2h30 (11%)	2h20 (17%)
手戻り	0h00 (0%)	0h55 (7%)
バグ修正	0h10 (1%)	1h30 (11%)
合計	22h45 (100%)	13h35 (100%)

各工程の工数を割合で見ると，被験者 A は変更要求仕様書と変更設計書の作成に半分を費やし，残りはリバースエンジニアリングとコーディングに費やし，手戻りとバグ修正にはほとんど工数を費やしていない。一方，被験者 B はリバースエンジニアリングと手戻り，バグ修正に 6 割以上の工数を費やしており，リバースエンジニアリングに使っている工数の割合は被験者 A よりも多い。

派生開発前, ならびに派生開発後の各被験者の, FuRISCV のソースコードに関する種々のメトリクスを表 2 に示す。計測には SourceMonitor[9] を用いた。算出されている複雑度は McCabe の循環的複雑度 [10] である。いずれの被験者も平均複雑度，平均ネスト数がわずかに増減しているのみであり，提案手法によってソフトウェアの複雑度が大きく改善されたり，悪化したりはしていないことがわかる。

表 2 派生開発前後の FuRISCV のソースコードメトリクス

メトリクス	オリジナル	被験者 A	被験者 B
クラス数	38	38	40
コード行数	3,441	3,793	3,645
平均メソッド数/クラス	3.55	3.63	3.70
平均複雑度	2.39	2.49	2.35
最大複雑度	22	22	22
平均ネスト	1.82	1.89	1.94
最大ネスト	9	9	9

4.4 検討

本稿で提案するリバースエンジニアリング管理手法は，XDDP における工程と作成しなければならない成果物を増すものであるが，この増加した工程と成果物の作成に要する工数と引き換えに，派生製品のソフトウェアの開発に実質的に寄与しない不要のリバースエンジニアリング，ならびに手戻りに要する工数を大幅に削減することを狙うものである。しかしながら，今回のケーススタディにおける，両被験者の工数の絶対比較では，本手法による派生開発の工数削減を示すことができなかった。

被験者 B は FuRISCV のソフトウェアのつくりに関する知識不足によって，手戻りが発生しているのに対し，被験者 A はソフトウェア開発経験年数で劣るものの，手戻りとバグ修正に費やした工数は，絶対比較と割合による相対比較でも優っていた。この比較結果を見れば，被験者 A は相対的に少ない工数で，派生開発に必要な FuRISCV のソフトウェアのつくりに関する知識を十分に取得できていることが言えよう。また，被験者 B がリバースエンジニアリングに 4 割程を費やした割に，手戻りとバグ修正に 2 割近い工数を費やしているのに対し，被験者 A はリバースエンジニアリングの割合はさほど変わらずとも手戻りとバグ修正にはわずか 1%しか費やしていない。コーディングに要した時間は，被験者 A は被験者 B より少ない割合であり，絶対比較においてもわずか 10 分長いのみである。被験者 A は派生製品のソフトウェアの開発に必要な仕事「のみ」をしていることが，この比較からは推測される。

今回のケーススタディでは絶対的な工数比較では提案手法の有効性を示すことはできなかった。そもそも被験者の開発経験に大きな差があるため絶対的な工数比較では手法の効果を適切に測定できないものと考える。しかし，各工程の工数が全体に占める割合を相対的に比較することで，XDDP におけるリバースエンジニアリングの工数を削減し，派生開発の全体工数削減に寄与できる可能性を示した。多くの開発者が関わる大規模プロジェクトでは，手戻りのペナルティが大きくなることを考えれば，提案手法のねらい通り，リバースエンジアリング工数を手戻りとバグ修正に要する工数削減分で償却することが可能になるものと思われる。

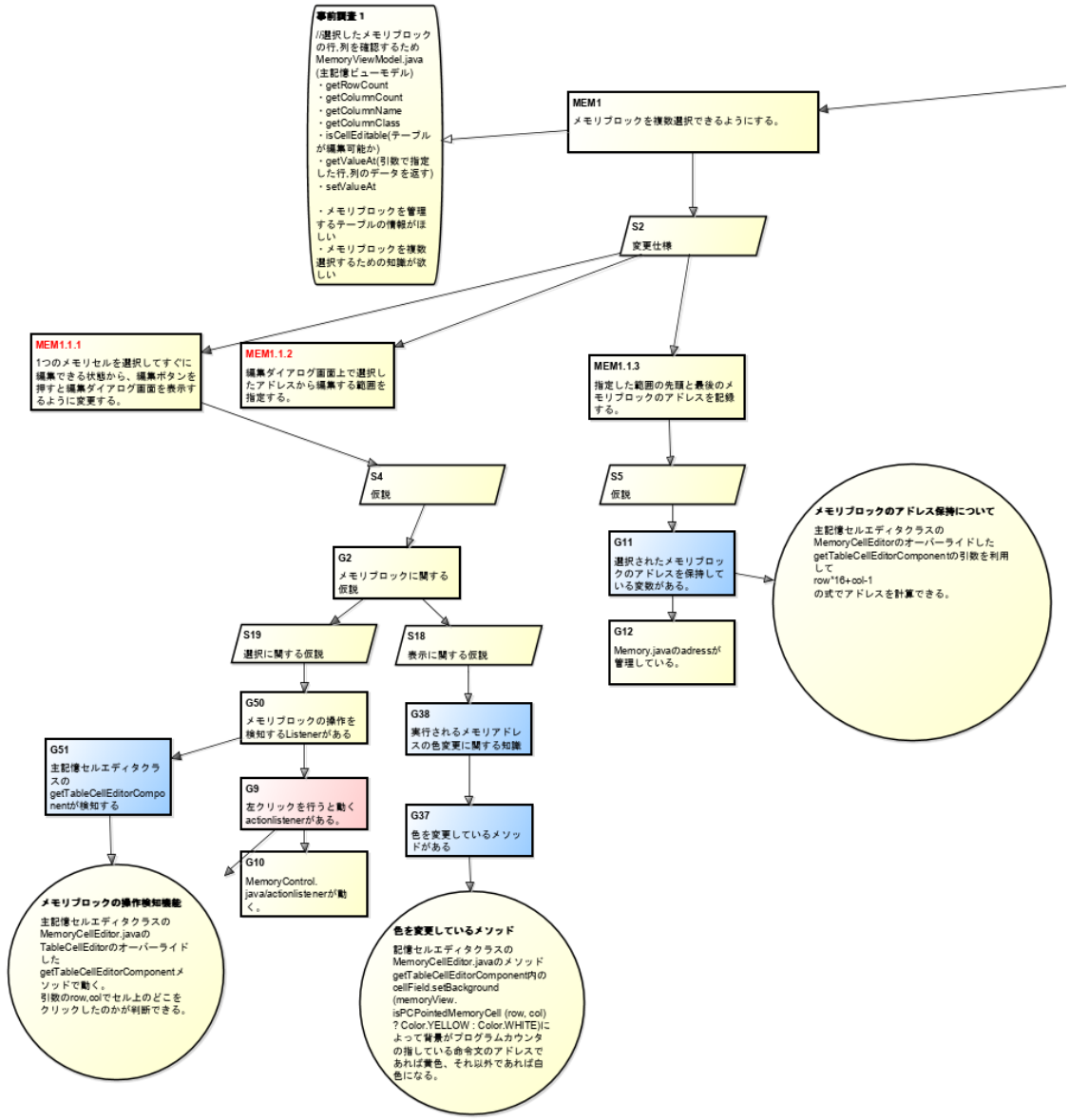

図 5 ケーススタディで作成された GSN（一部，レビュー前のもの）

5. 関連研究

XDDP のスペックアウト工程におけるリバースエンジニアリングの改善事例 [2], [11] では，変更要求仕様の不正確さに起因するソフトウェアの不具合と工数の増加が述べられている。両事例とも，不十分な既存ソフトウェアの調査（本稿で言うリバースエンジニアリング）を変更要求仕様の不正確さにつながる原因のひとつとして挙げており，既存ソフトウェアを俯瞰したうえで合理的な調査を実施するプロセスを定義することの必要性が主張されている。

文献 [2] では，担当者に既存製品ソフトウェアに対する知見がないプロジェクトにおける既存ソフトウェアの調査プロセスを定義している。定義された調査プロセスは，既存ソフトウェアの設計書を入力とし，処理フローを抽出して USDM 形式で記述し，それをもって既存ソフトウェアを俯瞰する。必要な箇所のみ詳細な調査を実施し，最終的に変更要求仕様書が完成される。

文献 [11] では，既存ソフトウェアの調査を事前調査，変更箇所調査，影響範囲調査の 3 ステージに分離し，それぞれの調査の目的を明確にし，そのプロセスを与えている。事前調査は既存ソフトウェアを俯瞰的に調査して調査のテーマを定めることを目的とし，変更箇所調査，影響範囲調査はその名の通り目的を有する。

XDDP のリバースエンジニアリングにおける問題認識，ならびに既存ソフトウェアの俯瞰しプロセスに基づいてリバースエンジニアリングを実施するアプローチについては，提案手法はこれら先行研究と変わらない。しかしながら，これら先行研究は品質の高い変更要求仕様書の完成を

専らの目的としているのに対し，提案手法は GSN を用いて調査の目的，計画，過程，結果を体系的，かつ追跡可能なかたちで記述し，既存ソフトウェアの形式知として資産化するところにちがいがある。

また，先行研究は設計書やソースコードを既存ソフトウェア調査の出発点とするアプローチであるのに対し，提案手法は想定変更要求仕様を出発点とし，そこから既存ソフトウェアを必要に応じて調査し，想定変更要求仕様を完成された変更要求仕様に修正していくアプローチを採っている。これらアプローチの良否はプロジェクトの条件によって変わってくるものと思われるが，提案手法はより高い抽象度でリバースエンジニアリングを計画するアプローチであり，既存ソフトウェアの現状に束縛され過ぎない変更を議論できる余地を残している。そのため，既存ソフトウェアが特定の製品に過剰に最適化されたかたちになっている場合には，それを他の派生製品に再利用しやすいように汎化する改善することが可能となると考えている。さらに，既存ソフトウェアの知識が部分的にある場合は，それを想定変更要求仕様として容易に取り込むことで，それが本当に正しいのか検証の対象に挙げたり，リバースエンジニアリングの範囲をトップダウンで絞ったりすることも可能である。もっとも既存ソフトウェアの知識が全くない場合については，先行研究と変わらず既存ソフトウェアからの調査が必要となる。

6. まとめ

本稿では，派生開発プロセス XDDP において既存ソフトウェアを体系的，かつ必要最小限にリバースエンジニアリングする方法論を提案し，ケーススタディによってその評価を行った。

提案手法は，リバースエンジニアリングを実施する前に，既存ソフトウェアのつくりを想定のもと，裏付けのない想定変更要求仕様を記述する。リバースエンジニアリングは，想定のために立てた既存ソフトウェアに関する仮説を証明するためにのみ実施し，それによりリバースエンジニアリング範囲の不用意な拡大と工数増大を避ける。リバースエンジニアリングの目的，計画，過程，結果をすべて GSN で記述し，開発チームで共有し資産化する。GSN のレビューによって開発者個人に私蔵されている，開発者個人に十分に知らされていない，あるいは開発チームや開発者個人で誤解している既存ソフトウェアに関する知識を明らかにし，将来の派生開発に再利用できる確たる知識に変える。ケーススタディによる評価では，提案手法によって派生開発に要する全体の工数を削減することはかなわなかったが，手戻りやバグ修正に要する工数が絶対値でも割合においても大幅に抑えられ，工数のほとんどが派生製品のソフトウェア開発に直接的に必要な作業に使われたことが確認できた。

今後の課題としてより規模を大きくしたケーススタディによる評価が挙げられる。本稿で実施したケーススタディは，一人の開発者によって派生開発が行われており，チーム開発における提案手法の効果は評価できていない。また，リバースエンジニアリングにおいて仮説の証明の過程を GSN として適性に記述するには，開発者個人の論理的思考能力や習熟が必要と見られ，記述のパターンを整備していくことが実務上重要に思われる。

謝辞 本研究は科研費（No.24500052）の助成を部分的に受けている。

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 年.
[2] 中井 栄次,「XDDP 適用による無知見プロジェクトのプロセス改善」, 派生開発カンファレンス 2010, 2010 年 6 月.（https://affordd.jp/conference2010/xddp2010_P4.pdf, 最終アクセス日: 2021 年 6 月 7 日）
[3] K. Kobata, E. Nakai, and T. Tsuda, “Process Improvement Using XDDP: Application of XDDP to the Car Navigation System,” *Proc. 5th World Congress for Software Quality*, 8 pages, Nov. 2011.
[4] 櫻庭 恒一郎,「XDDP 導入による派生開発プロセス改善とその効果」, SPI Japan 2012, 2012 年 10 月.（http://www.jaspic.org/event/2012/SPIJapan/session2A/2A2_ID014.pdf, 最終アクセス日: 2021 年 6 月 7 日）
[5] 清水 吉男, 要求を仕様化する技術・表現する技術, 改訂第 2 版, 技術評論社, 2010 年.
[6] 西浦 洋一, 浅野 雅樹, 中西 恒夫,「フィーチャ指向アプローチによる自動車ボディ系製品のプロダクトライン開発への移行事例」, 情処論, Vol. 61, No. 2, pp. 417–428, 2020 年 2 月.
[7] T. Kelly and R. Weaver, “The Goal Structuring Notation: A Safety Argument Notation,” *Proc. Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.
[8] A. Waterman and K. Asanović (eds.), *The RISC-V Instruction Set Manual*, Vol. I: Unprivileged ISA, Dec. 2019. (Available at: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf, Accessed 7 June 2021)
[9] SourceMonitor, http://www.campwoodsw.com/sourcemonitor.html（最終アクセス日: 2021 年 6 月 7 日）
[10] T. J. McCabe, “A Complexity Measure,” *IEEE Trans. on Software Engineering*, Vol. SE-2, No. 4, pp. 308–320, Dec. 1976.
[11] 飯泉 紀子, 足立 久美, 清水 吉男, 宇田 泰子, 夛田 一成, 川井 めぐみ, 伊藤 友一,「変更の影響範囲を特定するための『標準調査プロセス』の提案」, ソフトウェア品質管理（SQiP）研究会第 30 年度第 6 分科会 A チーム成果報告, 2015 年 2 月.（https://www.juse.or.jp/sqip/workshop/report/attachs/2014/sqip6-a.pdf, 最終アクセス日: 2021 年 6 月 7 日）