SPI Japan2025 発表概要集

セッション順番 ＞ 発表順番に記載しています。

1A1　変化に負けない技術者へ！品質技術者リスキリングプロセス構築の取り組み～ 知識をつなげて知恵を生み出す品管部門を作ろう！～　小林一郎（ＳＯＭＰＯシステムズ株式会社）

＜タイトル＞

変化に負けない技術者へ！品質技術者リスキリングプロセス構築の取り組み

＜サブタイトル＞

～ 知識をつなげて知恵を生み出す品管部門を作ろう！ ～

＜発表者＞

氏名(ふりがな)： 小林一郎（こばやしいちろう）
所属： ＳＯＭＰＯシステムズ株式会社　業務品質部

＜共同執筆者＞

氏名(ふりがな)：
所属：

＜主張したい点＞

DX やクラウド関連など、次々に発表される最新のテクノロジーや、それに対応するための開発ライフサイクルの多様化など、システム品質に影響を及ぼす要素の変化に適応できる品質技術者を増やすためのリスキリングプロセス構築の取り組みを行った。本プロセスを経た品質技術者の支援を受けた社内プロジェクトから効果的と評価される結果も出てきており、品質向上効果が期待できるため紹介する。

＜キーワード＞

品質管理、品質検査、DX、クラウド、外部サービス、アジャイル開発、ウォーターフォール開発、SQuaRE

＜想定する聴衆＞

品質管理部門の方、つぎつぎに現れる最新テクノロジーの品質管理に困っている方

＜活動時期＞

2024 年 4 月から 2025 年 3 月

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

☐ 着想の段階(アイデア・構想の発表)
■ 改善活動を実施したが、結果はまだ明確ではない段階
☐ 改善活動の結果が明確になっている段階
☐ その他(　　)

＜発表内容＞
1.背景
私が所属する当社の品質管理部門である業務品質部は、当社で発生した 1 年分のシステム障害の傾向を分析し、抽出した弱点を克服するための対策を計画立てて実施する PDCA サイクルを回す改善活動を続けている。2023 年度の障害発生数は、改善活動開始前の 2021 年度と比較して約 40%減となったが、特徴的な傾向として、外部サービス（社外の事業者から提供を受けたシステム）に起因した障害が全体の約 60%を占めていた。該当障害の個別事象を分析した結果、たとえば、特定クラウドサービスでのスケールイン（サーバー台数などを減らしてリソースを最適化する処理）のタイミングで通信の瞬断が発生し、処理中の業務アプリケーションが異常終了するケースなど、採用した外部サービスのみに限定された障害原因が複数存在することがわかった。当社は、「オンプレミス（自社持ち）のメインフレーム（大型汎用コンピュータ）環境でのウォーターフォール型開発」（以降「従来型開発」と表記）を 30 年以上続けており、品質管理プロセスも従来型開発向けとしては成熟しているものの、前述のような特定の外部サービスに限定された不具合を開発中に検知できない等、従来型開発以外のシステム開発における品質管理プロセスが十分ではないことを新たな弱点として認識し、業務品質部はこの克服を 2024 年度の品質向上施策として実施することとし、事業計画として経営承認された。
本発表は、未経験の技術を用いて構築されるシステムの品質管理を適切に行うためのプロセス構築の取り組みについて、当社の品質技術者の 1 人であり、施策担当の私が紹介するものである。

2.改善したいこと
今回改善したい問題は、外部サービスに起因した障害の発生である。発生した障害の再発防止策を都度策定・実施していく対策も考えられるが、今後も外部サービスで採用される最新テクノロジーや、外部サービスに適合するためのシステム開発ライフサイクル（アジャイル開発など）の変化スピードはますます速くなっていくことを鑑みると、既存の再発防止策に当てはまらない新出の不具合パタンによる障害発生は有り得ると想定できることから、対策としては不十分と考え、障害の発生をシステム開発中に検知する未然防止の取り組みを行うこととした。改善効果の目標値としては、外部サービスに起因した障害件数が前年度より 50%減となること、および、当部が品質検査を行う全プロジェクトに対するアンケートにおいて、当部の品質検査が効果的であるという回答の割合が前年度の 76%を上回ることとした。

3.改善策を導き出した経緯
改善策を導き出すにあたり、障害未然防止策として、当社未経験の外部サービスで採用された最新テクノロジーや開発ライフサイクルなどの新出の技術要素（以降「未経験技術」と表記）に合わせた開発標準の改定を検討したが、次々に発表される未経験技術の新しいパタンが発生するたびに標準の改定を行うことで、改定作業に追われた品質技術者の品質管理業務に割く時間が減少してしまい、それが原因でこれまで検知できていた品質低下要因の見逃しが起きてしまう可能性があるため、現実的ではないと判断し実施しないこととした。前述した再発防止策の実施、開発標準の改定による未然防止策の実施ともに有効な改善策とはならないことから、「モノ・コト」観点のプロセス改善には限界があると考え、発想を変えて、未経験技術に対して必要な知識を学び、適切な品質管理、特に品質検査による不具合摘出が行える品質技術者を生み出すという、「ヒト」つまり人材育成観点のリスキリングプロセス構築を改善策として提案し、施策として承認された。施策チーム体制としては、私がリーダーを担当し、立候補した部内の品質技術者 3 名をメンバとする 4 名体制にて取り組みを開始した。

施策チームにて改善策の議論を行ったところ、未経験技術に対する品質検査技術の向上を目指すことは当然だが、ベテラン技術者が陥りがちな、「アインシュテルング効果」（過去の成功経験に固執して、他の考え方や解決策を見落としてしまう認知バイアスの 1 種）から脱却することも必要という意見がメンバから出された。私自身が従来型開発の品質管理手法に固執してしまった経験があることや、他の品質技術者との会話でも同様の傾向を感じたケースが複数あったこともあり、必要性に共感し、メンバ全員の納得感も得られたため取り組むこととした。改善策とした具体的な取り組みプロセスは以下の３つである。
(1) 当部品質技術者の弱点抽出
(2) 弱点を克服するための研究と品質検査実務への適用によるリスキリング

(3) 品質技術者のマインド変革（アインシュテルング効果からの脱却）

4.改善策の内容

(1) 当部品質技術者の弱点抽出

「１．背景」で述べたとおり、外部サービスに起因した障害数が増加しているという傾向は、われわれ品質技術者が品質検査で検知できなかった不具合の傾向であると考え、 外部サービスに起因した障害を検証のうえ弱点を抽出することとした。

(2) 弱点を克服するための研究と品質検査実務への適用によるリスキリング

前述の弱点を克服するためには外部知識の導入が必要と考え、社外研修やセミナー、コミュニティ（以降「社外研修等」と表記）に品質技術者が参加して学び、それをもとに品質検査観点を作成のうえ品質検査実施要領に追加することとした。これにより、未経験技術に対する品質検査技術のアップデートおよび業務実践によるリスキリングが可能となる。

(3) 品質技術者のマインド変革（アインシュテルング効果からの脱却）

当部の品質技術者は社内審査にて認定された専門職である。審査内容にはヒューマンスキル（コミュニケーション能力やリーダーシップ、品質のプロとしての志など）も含まれているため、アインシュテルング効果が働くとしても、過去の自分の成功経験のパタン以外は専門職としての責任が果たせないという意識が働いているだけであり、過去の成功経験と同等のものが得られれば自分の品質検査のパタンが広がり始め、自然にアインシュテルング効果から脱却できるという仮説を立て、これを実行することとした。

5.改善策の実現方法

当部品質技術者の弱点抽出については、外部サービス起因の過去障害情報全件を施策チームにて分析し、分析結果をもとに、現状の品質検査では検知することができないと品質技術者全員の認識が一致した不具合事象を列挙のうえ、同種の事象を汎化したものを弱点として定義した。弱点を克服するための研究と品質検査業務への適用については、施策チームにて弱点に関連すると思われるテーマの社外研修等を探して受講していたが、当社のシステム開発の特徴（従来型開発が主体だが、外部サービスの利用が増えてきている）に当てはまる内容のものが少なかったため、テーマに加えて講師の経歴（当社のシステム開発の特徴と同じ状況を経験していると思われる方）に着目する方針にて研修やセミナーを探して受講したところ、有用な知識が得られるようになった。しかし、得られた知識はシステム開発を対象としたものが多く、これを品質管理の観点に置き換える必要があるため、品質技術者全員にて納得できるまで議論と修正を繰り返したうえで品質検査観点としてチェックリスト化し、品質検査業務の実施要領にチェックリストの使用を追加のうえ検査業務を開始した。

具体的な弱点と対応策となる品質検査観点は以下の 4 点である。

① クラウドサービス固有の障害を、開発中の欠陥として検知できない

・品質検査観点：過去障害の再発防止策が適用されているか。障害設計が適切に行われているか

・補足（社外研修等から得た知識と品質検査観点への置き換え）

クラウドサービスなどのブラックボックスに類するシステムは新出の障害パタンの予測が難しく、リスク先出型の品質　管理計画によるコントロールは不可能という知識を得た。したがって、過去の障害パタンの再発を防ぐことに加え、　障害発生時のシステム復旧手順などの備えが適切に設計されていること（障害としてユーザーに認知される前にシステムを復旧させることが目的）を検査する観点とした。

② アジャイル開発（スクラム）を採用したプロジェクトが、ユーザー要件を実装することに注力してしまい、異常系処理や非機能要件の考慮が不十分になっていることを、開発中に検知できない

・品質検査観点：プロダクトバックログに国際規格 SQuaRE（国際標準化機構が定めた品質要求と評価の 規格）の品質特性に相当する定義が網羅されているか。

・補足（社外研修等から得た知識と品質検査観点への置き換え）

経験の浅いスクラムチームは異常系処理や非機能要件まで考慮が及ばないケースがあり、プロダクトバックログ　の情報から異常系処理や非機能要件が考慮されているかを確認することが有効

という知識を得た。当社のスクラムチームは成熟したチームが多くない可能性を考慮し、不足がある場合は SQuaRE の品質特性モデルをもとに体系的に指導し、検査とセットで異常系処理や非機能要件を含むシステム品質に必要な特性について開発メンバに理解を深めてもらう現場育成を行うことが、中長期的な品質向上として有効と判断し、これを観点化した。

③ ハイブリッド開発（アジャイル開発システムとウォーターフォール開発システムの併存）を採用したプロジェクトのシステム間インターフェースが確定していないことを開発中に検知できない
※例：ウォーターフォール開発システムでの外部設計フェーズ終了後に、同システムとインターフェースを取るアジャイル開発システムのスプリントが開始されるため、外部設計フェーズが終わってもインターフェースが確定しない。
・品質検査観点：プロジェクトが、システム間インターフェースの変更管理運用について、常に最新の状態にて一
元管理できているか
・補足（社外研修等から得た知識と品質検査観点への置き換え）
ハイブリッド開発プロジェクトの管理は、アジャイル開発とウォーターフォール開発双方に精通した管理要員が開発プロセスを定義する必要があるという知識を得た。当社では該当する要員が必ずしも調達できるとは限らないため、各システムの接点であるインターフェースの状態を、開発ライフサイクルの種類に関係なく一元管理することが適切と判断し、これを観点化した。

④ アジャイル開発における開発チームの適切さを判断できず、品質確定のポイントが不明瞭となり、不具合が作り込まれたことを開発中に検知できない。
※例：ウォーターフォール開発経験しかないメンバが多いスクラムチームが組成され、案件やチームに合った品質管理に必要な手順を十分に検討しないまま開発を進めてしまう。
・品質検査観点：開発チームのメンバ構成が妥当となっているか（プロダクトオーナーは役割を果たせるか、開発メンバの能力や適性は適切か）
・補足（社外研修等から得た知識と品質管理観点への置き換え）
アジャイル開発はウォーターフォール開発と比べて、開発ルールを創造・改善できる要員の割合が多い必要があるという知識を得た。これを受け、アジャイル開発チームのメンバ構成が、開発ルールを創造・改善できる要員の割合が少なく、開発ルールに沿って作業する要員の割合が多くなっていないこと（ウォーターフォール開発チームと同様の要員構成になっていないこと）を確認することが必要と判断し、観点化した。

品質管理技術者のマインド変革については、前述の仮説に基づき、従来型開発における品質管理技術で未経験技術の検査観点作成が可能かどうかをあらためて徹底的に議論し、議論が行き詰まったタイミングで社外研修等より得た知識を説明のうえ、品質検査観点作成に向けた議論を深めて検査観点の定義に到達するという、ワークショップ形式での検討を行うこととした。これにより、各品質管理技術者が自分の技術に不足があったという事実を認めて受け入れたのちに、外部の新しい知識をつなげた議論によってあらたな品質検査観点を作成する体験をすることで、過去の成功経験に相当する「知恵」を生み出す楽しさと、この知恵を用いて今までにできなかった品質向上が実現できるというワクワク感が生まれることによって、アインシュテルング効果から脱却できると考え、これを実行した。

6.改善による変化や効果
未経験技術に対する検査観点を適用した品質検査を開始したが、まだ開始後 1 年未満であり、当社のプロジェクトは 1 年以上の期間となるものが多く、現時点では本番稼働をむかえたシステム数が少ないため、障害件数の推移による改善効果の検証は十分に行えていないものの、後述のプロジェクトへのアンケートには、外部サービス障害の未然防止効果がある旨の回答が寄せられており、一定の効果は出ていると考える。また、品質技術者が今回構築したリスキリングプロセスを用いて、自ら新たな弱点を設定したうえで改善活動を開始する行動の変化が見られるようになった。今回構築したプロセスの初回実施として、品質技術者全員が参加する形で、外部知識と他の品質技術者の知識をつなげて知恵を生み出すプロセスを体験

できたことにより、未経験技術に対する「苦手意識」よりも、新しい品質技術が身に付けられ品質向上につなげることができる「楽しさ」が勝るというムードが醸成されつつあることから、アインシュテルング効果から脱却する効果も表れてきたと考える。

7.改善活動の妥当性確認

改善活動の妥当性については、当部の品質検査の効果について当社の全プロジェクト（常時 10 プロジェクト以上が稼働している）の PM、PL、PMO、品質管理担当者を対象にアンケートを行っており、改善活動実施前のアンケート結果は、効果ありの回答が全回答者の 76%であったが、改善活動実施後のアンケート結果では、効果ありの回答が全回答者の 94%となった。改善活動実施前のアンケート結果で効果なしの回答は未経験技術によるシステム開発を行っていたプロジェクトからのものであり、改善活動実施後のアンケート結果では効果なしの回答はなく（効果あり以外の 6%は「どちらでもない」という回答）、フリー記述欄に、クラウドサービス障害の未然防止に役立つ支援が得られた旨のコメントも寄せられており、改善活動については妥当であったと考える。

今回構築したリスキリングプロセスは一定の効果が確認できたものの、現時点では改善活動実施後に本番稼働をむかえたシステムが少ないため、障害発生数の推移による妥当性確認までには至っていない。今後は引き続き改善活動の妥当性検証を行い、不足していた点やあらたな改善点を検知した際には適切な改善施策を提案・実行していく PDCA サイクルを回し続けていきたい。

参考情報

[1] ISO/IEC25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation(SQuaRE) - System and software quality models

1A2 メンタルモデル構築による要件定義能力の向上　服部悦子（住友電工情報システム株式会社）

＜タイトル＞

メンタルモデル構築による要件定義能力の向上

＜サブタイトル＞

＜発表者＞

氏名(ふりがな)：服部　悦子　(はっとり　えつこ)
所属：　住友電工情報システム株式会社　システムソリューション事業本部

＜共同執筆者＞

氏名(ふりがな)：　中村　伸裕　(なかむら　のぶひろ)
所属：　住友電工情報システム株式会社　ＱＣＤ改善推進部

＜主張したい点＞

システムエンジニアの頭の中にある目にみえない業務理解をメンタルモデルとして認識し、メンタルモデルの構築を意識して進めることで要件定義の能力、及び 成果物としての質が向上する可能性がある。今回、メンタルモデル構築を組み込んだ要件定義トレーニングを実施し、要件定義の経験が少ないシステムエンジニアの要件定義能力向上が確認できた。

＜キーワード＞

要件定義，メンタルモデル，業務フロー，状態遷移図, ER 図,
REBOK, CMMI, IPA ユーザのための要件定義ガイド，花束問題,
要件定義研修

＜想定する聴衆＞

プロジェクトマネージャ、プロジェクトリーダー、システムエンジニア、
要件定義担当者、IT 教育担当者

＜活動時期＞

2024 年 12 月～2025 年 5 月 (継続予定)

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
□ 改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1.背景

住友電工情報システム株式会社は、住友電気工業株式会社(以下 住友電工)の情報子会社でエンタープライズ・システムの構築を行っている。住友電工グループ各社の業務システムを標準化されたツールとプロセスで、コストを抑え、品質よく開発することが組織としてのミッションである。

これまでシステム開発における企画、要件定義フェーズは親会社の情報システム部の責任範疇であったが、ここ数年は我々にも求められるように変化してきた。しかし我々はシステムレベルの要件定義以降に特化して標準化を推進してきた組織であり、業務要件定義については標準化もできておらず、教育もしないまま一部メンバーの経験則で進めている状態であった。その結果、次の問題が発生している。

問題１． 業務を含めた全体の要件定義がうまく進まない

問題２． 外部設計に着手してから、要件定義の不十分さに気づき手戻りが多発する

問題３． システムテスト・本番稼動以降で、業務上不可欠な改善要望が多発する

これらの要件定義に起因する問題で品質不良・納期遅延・コスト超過が発生しているが、要件定義不足というだけで、何がどう不足だったのか、何をすべきだったのか原因分析できておらず、根本的な対策をとれていない。

よって、まずは我々の要件定義スキルを向上させる必要があると判断し、要件定義の研修を実施することにした。

2.改善したいこと

要件定義スキルの向上、システム開発プロジェクトの QCD 目標の達成

及び、そのための効果的な研修プログラムの確立

3.改善策を導き出した経緯

研修内容を検討するにあたり、「メンタルモデルの構築によって要件定義能力が高まる」という仮説を立てた。[1]

一般的に、メンタルモデルとは、頭の中にある「ああなったらこうなる」という行動のイメージを表現したものであるが、今回は、システムエンジニアの頭の中にあるシステム化対象業務への理解を指し、ER 図や業務フロー、状態遷移図などの記述モデルが統合されたものとして考えている。

具体的には、要件定義の際に利用部門から業務内容をヒアリングし、その情報を基に ER 図や業務フロー、状態遷移図などの記述モデルを作成する。この際、業務に対するメンタルモデルが十分に構築されていれば、そこからある側面で切り出し、記述モデルを作成することが可能になる。切り出す側面とは、データに着目するとデータモデルとして ER 図、業務の流れに着目すると業務フロー図などである（図 1）。

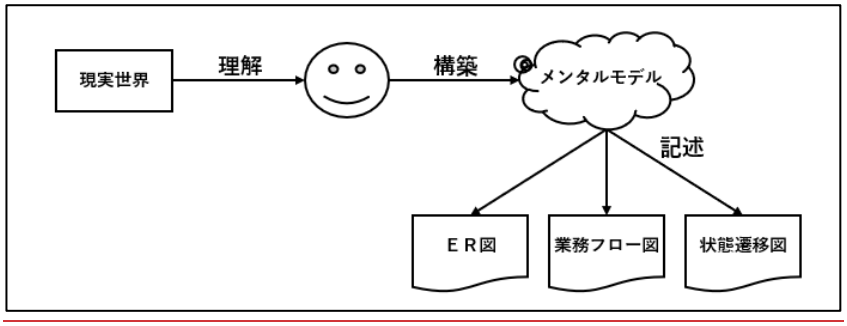


図 1 メンタルモデル

私たちが考えるメンタルモデルは複合的で多次元であり、1 つのメンタルモデルから生成された複数のモデルは、整合して質が高くなることが期待される。さらに、ER 図や業務フロー図に書き出すことでメンタルモデルの曖昧な部分に気づき、メンタルモデルをさらに成長させることができると考えている。

もし ER 図や業務フロー図、状態遷移図などをバラバラに作成した場合、これらの間に矛盾があっても発見が難しくなる。しかし、メンタルモデルを持つことで、さまざまなモデルを効率的に矛盾なく作成することができるという利点があると思われる。

こう考えると、現状において要件定義が上手くできる人とそうでない人の違いは、メンタルモデルが十分に構築されているかどうかに起因するのではないか。メンタルモデルが不十分なまま記述モデルを作成し、現実を十分に説明できないモデルや不整合のあるモデルを作成してしまっているのではないだろうか。これが、現在、私たちの組織で直面している問題を発生させているのではないか、と考えた。

したがって、「メンタルモデルの構築によって要件定義能力が高まる」という仮説のもとに、メンタルモデルを構築できるようになる研修プログラムを考えることにした。

4.改善策の内容

先述の通り、我々の組織は要件定義に関する知識・スキルとも不足していることから、研修は一般知識の学習(知識編)と演習によるスキル習得(スキル編)が必要と考えた。一般的に物事を学ぶには本を 3 冊読めばよいと言われていることから、知識としては要件定義に関する書籍や文献など 3 種類の学習をすることにした。スキル習得のための演習は、業務モデルの作成演習、書籍を使った演習、総合演習の 3 つとし、それぞれの演習の中でメンタルモデルを体感し、メンタルモデル構築を推進できるように準備した。また今回の研修では要件定義に関するエンジニアリングを中心に行うこととした。プロジェクト管理に先だってエンジニアリングを学ぶ理由は、エンジニアリングの観点で実施することが明確になっていないと管理もできないはずというところからである。

具体的な研修内容を次項に記載する。

5.改善策の実現方法

5-1.学習目標の設定

研修の実施に先だって以下の 6 つの学習目標を決め、それぞれの目標に対してスキル項目(計 43 項目)を設定した。

- ・要件定義に関する一般知識を取得し、実践で活用できる
- ・要件定義に必要なインプット(主に企画内容)を理解し、揃っているか判断できる
- ・要件定義書として作成すべきものを理解し、作成することができる
- ・業務要件を十分理解し、メンタルモデルを構築できる
- ・メンタルモデルを使って、要件定義の成果物の整合性・十分性・妥当性を確認できる
- ・要件定義を完了できるか、外部設計をするための材料が揃っているか判断できる

5-2.知識編の内容

以下 3 種類の学習を行うこととした。

① CMMI v1.3 要件開発(RD),要件管理(REQM) [2]
要件定義に関する能力を客観的に評価する視点を学ぶ目的で、CMMI の要件開発(RD),要件管理(REQM)を学習することとした。

② 要求工学知識体系(REBOK) [3]
要求を合理的に獲得・仕様化しシステム開発へとスムーズに接続することを目的とし体系化されていることから、要件定義を学習するには最適と考え学習することとした。

③ IPA ユーザのための要件定義ガイド [4]

本ガイドは IPA のユーザ企業メンバーで構成されたワーキンググループで作成されたもので、要件定義のさまざまな局面において発生する問題に焦点を当て、その解決の勘所と合わせて解説されている。我々の組織と同様にエンタープライズ・システム開発を行っている複数社の事例とともにノウハウが紹介されており、活用度が高いと考え学習することとした。

①～③について、講師より学習ポイントを解説し、書籍・資料を各自に読み込んでもらう期間(2～4 週間)を設け、簡単な確認テストを用意し回答してもらうことで学習を進めた。

5-3.スキル編の内容

以下 3 種類の演習を行うこととした。

演習 1．モデル作成演習

花束問題([5])について、受講者各自が業務フロー図、状態遷移図、ＥＲ図の 3 種のモデルを作成し、その後、他メンバーが作成したものと比較し、メンバー間で図の違いについてどう考えそのような図になったかお互いに説明し合う。これにより、メンタルモデルと記述モデルの関係を理解するとともに、自分自身が作成した記述モデルと他のメンバーが作った記述モデルの違いを認識することで、メンバー間でのメンタルモデルの差を認識してもらった。

演習 2．書籍演習

書籍「演習で身につく要件定義の実践テクニック」[6] より以下 2 種類のツールを実際の事例で作成することにした。

- ニーズ整理表

  書籍ではシステム化方針を決めるための情報が揃っているかを確認するツールとして紹介されている。我々の組織では、親会社の情報システム部でシステム企画が作成され、その後の要件定義から参画することが多いので、参画時点で要件定義のインプットとなる企画内容が十分揃っているか確認するためのツールとして紹介し、実際のシステム事例で作成してもらうことにした。

- 業務場面設定表

  書籍では業務フローを作成する前に業務場面を漏れなく抽出するためのツールとして紹介されており、社内にある実際のシステム事例で作成してもらうことにした。

これらの成果物についても各自が作成後、他メンバーと比較し、差分について議論することで各自のメンタルモデルの違いを認識し、かつ自身のメンタルモデルの不足があれば気づけるように演習を進めた。

演習 3．総合演習

本演習では仮想のガソリンスタンドを想定し、スマホアプリの提案を行うこととした。

現実のガソリンスタンド・ビジネスにおける環境の変化や、実在するガソリンスタンドアプリなどの事例を分析して、提案書を作成してもらうこととした。提案書には、業務フロー図、状態遷移図、ＥＲ図のモデル図を含めること、アプリ導入後の顧客視点でのメンタルモデルを良質なものに磨き上げることを指示し、演習に取り組んでもらった。

5-4.研修の実施

以下内容で研修を実施した。

期間：2024 年 12 月～2025 年 5 月
開催：計 10 回　各回 2 時間程度の講義・演習
形式：集合会議　又は　リモート会議
受講者：9 名　要件定義の経験が少なく、これから要件定義を担うメンバーを選出

研修教材は社内で新規に開発し、一部の講義は社外講師に委託した。

6.改善による変化や効果

6つの学習目標 及び スキル項目(計43項目)対して、受講者に受講前・受講後に以下5段階で評価してもらった。

表1．スキルレベル

スキルレベル	ポイント
知らない/できない	1
知っている	2
指導を受けながら実施できる	3
一人で実施できる	4
指導できる	5

結果、研修前後の評価結果は以下の通り。平均、中央値ともスキル向上を確認できた。

表2．研修前後のスキル評価

	研修前	研修後	(増減)
平均	1.59	2.77	(+1.18p)
中央値	1.46	3.00	(+1.53p)

N=9名、43項目

7.改善活動の妥当性確認

現在は研修終了直後であり受講者9名のアンケート結果しかないが、「受講していれば要件定義起因の問題が回避できそうか？」という問いに対し、9名中7名が「回避できそうだ」と回答した。他2名の内、1名は「回避できることがありそうだがすぐに思いつかない」、もう1名はＩＴ経験が少ない要員で「わからない」との回答であった。

また「メンタルモデルを意識する（メンタルモデルができていないことに気づく）ことで、要件漏れや誤りを回避でき、後工程での手戻りを防ぐことに繋がると感じる。」といった感想も得られ、要件定義スキル向上のための研修として妥当であったと考えている。

今後、受講メンバーの実際に開発業務において本研修内容の活用状況を確認し、要件定義の質がどうであったか、結果として開発プロジェクトのＱＣＤがどうなったか評価する予定である。加えて次の研修開催に向け準備を進める。

参考情報

[1] 中村 伸裕, ”[仮説] ソフトウェア・プロセスの質はメンタルモデルにあり ”,SPI Japan 2021

[2] カーネギーメロン大学 著 JASPIC CMMI V1.3 翻訳研究会 訳, "開発のための CMMI® 1.3 版", https://www.jaspic.org/activities/cmmi-dev-v1-3/

[3] 一般社団法人 情報サービス産業協会, "要求工学知識体系(REBOK)", 近代科学社

[4] 独立行政法人情報処理推進機構（IPA）, "ユーザのための要件定義ガイド 第2版 要件定義を成功に導く128の勘どころ", https://www.ipa.go.jp/archive/publish/tn20191220.html

[5] NPO 法人 IT 勉強宴会, "花束問題 1.2", https://www.benkyoenkai.org/contents/Practice/Bouquet1-2

[6] 水田哲郎, "演習で身に着く要件定義実践テクニック", 日経 BP 社

1A3　顧客価値共創への変革：システム運用保守における新たな「価値」の確立　相澤武（株式会社インテック）

＜タイトル＞
顧客価値共創への変革：システム運用保守における新たな「価値」の確立
＜サブタイトル＞

＜発表者＞
氏名（ふりがな）：相澤　武（あいざわ　たけし）
所属：株式会社インテック　品質革新本部　品質マネジメント革新部

＜共同執筆者＞
氏名（ふりがな）：植田　絵里（うえだ　えり）
所属：株式会社インテック　品質革新本部　品質マネジメント革新部

氏名（ふりがな）：廣川　敬久（ひろかわ　のりひさ）
所属：株式会社インテック　品質革新本部　品質マネジメント革新部

氏名（ふりがな）：原田　かおり（はらだ　かおり）
所属：株式会社インテック　品質革新本部　品質マネジメント革新部

＜主張したい点＞
システム運用保守に ITIL4 バリューストリームの考え方を導入することで、プロセスと人の連携を強化し、顧客価値共創を加速する「質で語られる信頼のトップブランド」の実現を目指している。本発表では、その活動内容と現在確認されている変化について紹介する。

＜キーワード＞
価値共創、バリューストリーム、継続的改善、人材育成、ITIL4

＜想定する聴衆＞
システム運用保守における品質マネジメントの新たなアプローチや、部門間の連携による改善事例に関心のある担当者。特に、既存プロセスの課題解決や顧客価値最大化を目指す方々。

＜活動時期＞
2024 年 4 月から継続中

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。
☐ 着想の段階（アイデア・構想の発表）
■ 改善活動を実施したが、結果はまだ明確ではない段階
☐ 改善活動の結果が明確になっている段階
☐ その他（　　）

＜発表内容＞

1.背景

当社では、グループ会社共通の基本理念のもと、「質で語られる信頼のトップブランド」の確立を目指し、継続的に「品質」、「生産性」、「技術力向上」に取り組んでいる。
この取り組みの中で、システム運用保守については、高品質サービスの提供を目指しているが、現状は、個々のプロセスの最適化に偏り、エンドツーエンドでの効率性や価値創出の可視化に課題を抱えていた。市場変化への迅速な対応と顧客提供価値の最大化を目指し、個々のプロセス改善に留まらず、顧客への価値提供という最終目標に向けて各活動を「つなぐ」ことが不可欠であると認識した。
この課題意識から、ITIL4 が提唱するバリューストリームの概念に注目した。これは、顧客への価値提供に焦点を当て、関連する活動を横断的に連携させることで、組織全体の最適化とアジリティ向上を実現する可能性を秘めており、真の価値共創を実現するためのアプローチであると考えた。

2.改善したいこと

現在のシステム運用保守サービス提供において、以下の具体的な課題を改善したいと考えている。
（１）顧客への価値提供の不明瞭さと最大化の障壁
　システム運用保守サービス提供プロセス全体が、顧客にどのような価値を、いつ、どのように提供しているのかが不明瞭
　であり、真の顧客満足度向上のための改善点が特定しにくい状況であった。
（２）継続的改善サイクルの未確立
　システム運用保守サービスのボトルネックや非効率性を体系的に特定し、継続的に改善していく仕組みが不足していた。改善活動そのものが単発で終わってしまうという「改善の断絶」が存在し、体系的なナレッジの蓄積も不十分であった。

3.改善策を導き出した経緯

上記の課題認識から、私たちはこれらの問題を根本的に解決するための新たなアプローチを模索した。まず、現状の主要なシステム運用保守サービス提供プロセスにおいて、具体的なサービス要求が顧客から価値として提供されるまでのフローを詳細に可視化する必要があった。
分析の結果、以下の根本原因が特定された。

・個々のプロセスは定義されているものの、プロセス間の引き渡しルールや責任範囲が曖昧な箇所が存在する。
・サービス全体を俯瞰し、ボトルネックを特定・改善する仕組みが不足している。

これらの根本原因を踏まえ、解決策を検討する中で、ITIL4 が提唱する「価値に着目する」といった原則と、サービスバリューストリームの概念に注目した。
個別最適化されたプロセスではなく、顧客への価値提供という最終目標から逆算して、システム運用保守サービス提供に関わる一連の活動を「バリューストリーム」として定義し、全体最適を図るアプローチが有効であるとの結論に至った。これにより、単なるプロセス改善に留まらず、顧客価値創出に直結する活動全体の「流れ」を抜本的に見直し、「つなぐ」ことで、新たな価値創造の仕組みを構築することが可能になると判断した。

4.改善策の内容

「2.　改善したいこと」で挙げた課題を解決するために、「3.　改善策を導き出した経緯」で導き出したアプローチに基づき、ITIL4 のバリューストリームの考え方を品質マネジメントシステムに導入する具体的な改善策を、プロセス面と教育面の 2 つの側面から以下に示す。
4.1 プロセス面
（１）主要なサービスバリューストリームの特定と可視化
　顧客の「需要」から「価値」への変換を促す主要なシステム運用保守サービスについて、関連する全ての活動を洗い出し、エンドツーエンドのバリューストリームマップを作成した。これにより、サービス提供の全体像を「可視化」した。各バリューストリームにおける顧客価値の定義と、その価値がどのように創造されるかを明確にすることで、顧客との「価値共創」の意識を高める。

（２）バリューストリーム内の活動と ITIL4 プラクティスの最適化
マップ化されたバリューストリームにおいて、各活動が ITIL4 のプラクティスとどのように関連しているかを明確にし、ボトルネックや無駄を特定する。
（３）継続的改善サイクルの組み込みと可視化
バリューストリームのパフォーマンス指標を定義し、定期的に測定・評価することで、継続的な改善サイクルを回す。測定結果は、月次のレビュー会議で共有することで、改善への意識を高めた。計画(Plan)、実行（Do)、評価（Check)、改善（Act）の PDCA サイクルをバリューストリームレベルで機能させ、継続的な学習と適応を促進した。

4.2 教育面
本改善策を実現し、組織に定着させるためには、教育面での取り組みが不可欠である。ITIL4 の資格取得推進も視野に入れ、以下の教育策を実施している。
（１）ITIL4 資格取得の推進と支援
ITIL4 ファンデーションの資格取得を奨励し、受験費用補助などのインセンティブ制度を導入した。これにより、社員の学習意欲と専門性向上を促した。 特定の役割（例：サービスオーナー、サービスマネージャー、プロジェクトマネージャー、運用責任者）を担う社員に対しては、ITIL4 マネージング・プロフェッショナル（MP）や ITIL4 ストラテジック・リーダー（SL）といった上位資格の取得も積極的に支援した。これにより、ITIL4 の深い知識と実践力を組織内に蓄積している。
（２）バリューストリーム思考のワークショップと実践演習
ITIL4 のプラクティスを具体的な業務にどう適用するかを議論し、部門横断での協働を促すワークショップを定期的に実施した。これにより、座学だけでなく、実践を通じてバリューストリーム思考を定着させ、部門間の「協働をつなぎ」を実践している。

5.改善策の実現方法

本改善策を実現するため、以下のステップで進めた。
（１）パイロットバリューストリームの選定とマップ化
まずは影響範囲が限定的で、効果測定がしやすいシステム運用保守サービスをパイロットとして選定し、そのバリューストリームを詳細にマップ化した。この際、現行のプロセスや課題を関係者間で共有し、共通認識を形成した。
（２）改善策の横展開と定着化
パイロットでの成功事例を水平展開し、他の主要なシステム運用保守サービスバリューストリームにも適用を進めている。継続的な改善活動を組織文化として定着させるため、定期的なレビュー会議を開催している。

6.改善による変化や効果

本改善策の導入により、以下のような変化や効果がで始めている。これらの変化は、活動間の連携がもたらす価値創造への貢献と言える。

（１）顧客価値の創出加速
顧客の要望から価値提供までのプロセス全体が可視化され、ボトルネックが解消されることで、サービスの提供速度が向上し、顧客への価値創出が加速する兆しが見えている。
（２）サービス品質の向上
バリューストリーム全体での品質管理が強化され、問題の早期発見・解決が可能となることで、サービス品質が向上し、顧客満足度の向上に繋がりつつある。
（３）従業員のモチベーション向上
自身が担当する業務がバリューストリーム全体の中でどのような役割を担い、顧客価値に貢献しているかが明確になることで、従業員のモチベーション向上とオーナーシップ意識の醸成が期待される。

7.改善活動の妥当性確認

今回の取り組みにより、顧客への価値共創に一定の成果が出始めていると考えている。
ITIL4 のバリューストリームの概念は、単なるツールやプロセスの導入に留まらず、組織文化の変革を促すものである。そのため、継続的な評価と改善がその妥当性を確立する上で不可欠であると考えている。
今後は、さらなるバリューストリームの特定と最適化、そして組織全体への展開を進め、真の「質で語ら

れる信頼のトップブランド」の確立を目指していく。

参考情報

[1]SPI Japan2021「事業活動と統合した品質マネジメントシステム確立に向けて」
[2]SPI Japan2022「事業活動と統合した品質マネジメントシステム「守り」と「攻め」の活動事例の紹介」

1A4　SPI Japan の活用方法　中村伸裕（住友電工情報システム株式会社）

＜タイトル＞

SPI Japan の活用方法

＜サブタイトル＞

＜発表者＞

氏名(ふりがな)：中村　伸裕
所属：　住友電工情報システム株式会社

＜共同執筆者＞

氏名(ふりがな)：
所属：

＜主張したい点＞

SPI Japan への参加はプロセス改善推進の重要な要素である

＜キーワード＞

SPI Japan、自動テスト、仕事と作業

＜想定する聴衆＞

プロセス改善推進者

＜活動時期＞

2024～2025 年

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
□ 改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1. 概要

筆者の所属する組織では 2007 年に富山で開催された SPI Japan に参加し、表 1 に示したとおり 2008 年以降は 2020 年(新型コロナ)を除き毎年 2～6 名が発表者として参加している。当初の目的は CMMI Level 5 達成のために社外事例を収集することであったが、しばらくするとプロセス改善者のスキルアップの場として活用するようになった。また、SPI Japan で聴講した基調講演、発表等を契機に社内のプロセス、ツールの改善を進めている本発表では当組織での SPI Japan の活用方法を紹介する。

表 1. SPI Japan の発表者数

No.	SPI Japan	発表者数
1	SPI Japan 2007	-
2	SPI Japan 2008	2
3	SPI Japan 2009	5
4	SPI Japan 2010	4
5	SPI Japan 2011	4
6	SPI Japan 2012	4
7	SPI Japan 2013	3
8	SPI Japan 2014	5
9	SPI Japan 2015	3
10	SPI Japan 2016	4
11	SPI Japan 2017	6
12	SPI Japan 2018	2
13	SPI Japan 2019	4
14	SPI Japan 2020	-
15	SPI Japan 2021	3
16	SPI Japan 2022	4
17	SPI Japan 2023	2
18	SPI Japan 2024	3
	合　計	58

2. プロセス改善推進者のスキルアップ

(1) 問題点

当組織でのプロセス改善は主に 6～8 名で構成されるワーキング・グループ（以下、WG）で行われており、解決すべき問題の設定、解決策の検討、手順書等の作成、施策の展開、評価といった流れで行っている。この中で解決策の検討、手順書等の作成等の期間が長く、活動が進むにつれて当初の問題を忘れてしまい、施策の実施が目的になることが多い。結果として問題設定から解決までのストーリーが頭の中に残らず、単に WG の宿題をやっているだけといった感じになっていることも多い。

(2) 解決策

実施した改善活動を SPI Japan で発表することで、問題から施策の実施までを論理的に語れるようにする。応募資料、発表資料のレビューは時間をかけて行っているが、問題から解決策の関係がロジカルに理解できていない人も多い。因果関係ではなく、連想ゲームのように繋がっている。IT 業界に多い比喩的な表現で具体的な現象、施策に対応していない言葉が適当に並んでいるケースも多い。これらの点を指導し、実施した改善活動をロジカルに理解してもらっている。

投稿までのスケジュールは表 1 のように定着している。

表 1. SPI Japan 発表スケジュール

月	活　動
10 月	SPI Japan 参加
3 月	定例会議の中で 10～20 分程度、発表内容のストーリーを検討。
4 月	ストーリーを書き出し始める
5 月	本格的に執筆。レビューは 1 時間/週のペース
8～9 月	プレゼン資料作成。レビューは 1 時間/週のペース

(3) 成果

プロセス改善は 2007 年頃から取り組んでいるが、18 年間途絶えることなく現在 5 つの WG が活動しており、改善活動が定着している。なお、改善活動は 1 年毎にメンバーを入れ替えることが多かったため、施策実施後、すぐ SPI Japan に応募している。効果を十分確認する前の発表となるため、成果の確認が弱くなるという課題が残っている。

3. 自動テストの改善

(1) 背景

筆者のグループでは Digital Assistant(以下、D.A.)[1][2][3] と名付けられたサービスを開発して開発部門のプロセス改善を推進している。上流成果物から該当工程の設計・開発を支援するものであるが、継続的に支援内容を強化しており、ソースの規模が日々拡大している。そのため、既存機能に不具合が生じやすい状態が続いていた。対策としてテストツール(Selenium)を使って自動テストを作成していたが、 機能改善に対して自動テストの保守が漏れているケースがあり、デグレ対策として十分に機能していない状況であった。

2024 年の SPI Japan で招待講演[4] で和田卓人氏から自動テストの講演を聴講した。この中では不具合の発生から解消までのリードタイム等のメトリクスを使って開発チームを評価しており、新しい価値の軸が得られた。この観点で自動テストの価値を再評価し、D.A. の開発で利用している自動テストを見直す切掛けとなった。

(2) 施策

D.A. はブラウザ上で動作するソフトウェアで、TypeScript で開発し、コンパイルして JavaScript で動作する。ブラウザを介した自動テストでは Selenium がよく知られており、D.A. の開発でも利用していた。しかし、Selenium による自動テストではソースコードのどの部分が実行されたか確認することができず、自動テストの善し悪しを評価することができなかった。JavaScript でカバレージが取得できるテストツールを調査した結果、Jest[5] で実現できることがわかり、導入することにした。自動テストはソース管理に利用している GitLab のパイプラインで実行する。ソースを push した際、自動的に自動テストが実行され、すぐデグレに気付けるようにする。更に SonarQube を導入し、カバレージを SonarQube に登録するようにした。この結果、カバレージの状況がいつでも確認できるようになり、自動テストの良し悪しが判断できるよう

になった。なお、自動テストの質に関する指標はカバレージだけでは不十分であり、コード量の対するテスト項目数等、今後測定していきたい。

また、講演の中でいつでもリリースできる環境になっていないことが問題であるとの内容があり、独自の仕組みを開発して 24 時間いつでもリリースできる環境をチームメンバーに開発してもらった。

[注釈] 具体的な内容はパラミックス中村さんが SJ2025 に投稿しています

(3) 成果

SPI Japan 2024 の参加を契機として D.A.の開発・運用環境の改善が進んだ。

(a) 自動テスト

Jest の加えて独自にテスト作成を効率化するライブラリを開発したことにより Selenium より効率的に自動テストが開発できている。Jest による自動テストは 2025/3 から開発を開始した。図 1 は 5 月末時点の自動テストの状況を SonarQube で表示したものである。箱はディレクトリを示し、箱の大きさはディレクトリに含まれるソースのライン数に対応している。カバレージが高くなると赤色から緑に変化する。視覚的に自動テストの整備が進んでいることが実感できる。テストを補強すべきソースが一目で確認できるようになった。自動テストの実装により統合テストで発見されるデグレは 2025/3 の 15 件から 2025/5 の 5 件と約 67%削減できた。

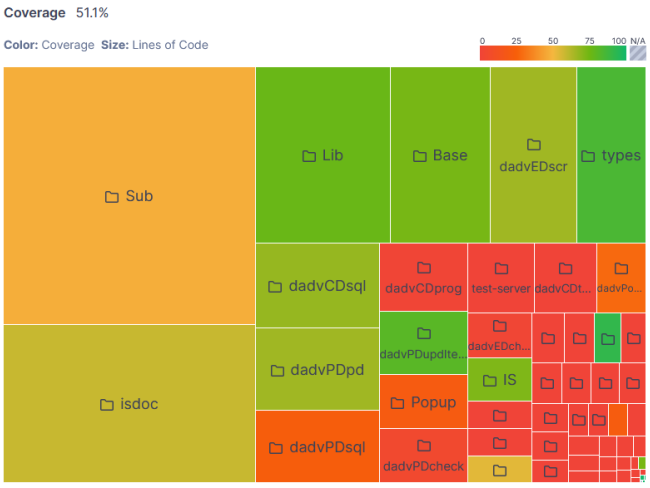


図 1. 自動テストのカバレージ

(b) リリース

D.A. は基本的にブラウザで動作するソフトウェアであるが、ファイルの入出力等、サーバー上の Node.js で稼働している機能もある。そのため、2 種類のソフトウェアを同期してリリースする必要があり、夜間等の利用者が少ない時間帯にリリースしていた。バグ修正等のリリースがいつでもすぐにできるよう、新旧 2 セットのソフトウェアを同時に稼働(Bule-Green Development)させ、利用者はメニューを選択した際、新バージョンに自動的に移動する方式にした。この仕組みにより、早ければ不具合発見から 1 時間以内に修正版がリリースできるようになった。利用者の修正待ちで使えない期間が短縮した。

4. 部下の指導

(1) 背景

人材育成は多くの組織にとって重要な課題であるが、筆者は ”人材育成” はできないと判断している。成長するかどうかは本人の意思次第であり、筆者ができるのは ”成長支援” である。成長したいと考えていない部下はいないと感じているが、同じ指導をしても成長のスピードはさまざまであり、成長に関する重要な要因が何か探していた。そんな中、SPI Japan 2024 の倉貫さんの基調講演[6]では ”仕事” と “作業” に違いについて説明があり、大きなヒントになった。

(2) 施策

”仕事” と “作業” の観点で社内の活動や過去の成長支援を振り返ってみるの “仕事” をしている人がかなり少ないことがわかった。仕事をしている人は顧客へ価値提供しているひとで、作業をしている人は単に成果物を作成しているひとである。両者は一見同じことをしているように見える。しかし、頭の中は表 2 に示すように大きな違いがあり、この差が成長速度の差になると実感できた。成長支援はまず “作業をしている人” を “仕事をする人” に意識改革させる必要かある。レビューの指摘では、”作業をしている人” は作成した成果物をレビューしてもらって指摘を直せば終了と考えている。レビュー者に考えてもらって自分の作業を完了させることを当然と思っている。日常会話で “仕事と作業” という言葉を頻繁に出すようにし、実施している作業を顧客に対する価値提供と関係づけられるように指導し、本人に ”仕事” を意識させている。

表 2. 仕事をしている人と作業をしている人の違い

	仕事している人	作業している人
活動のゴール	目標施策関連図を常に意識	聞いているけど作業とリンクしない
設計	完了基準：利用者が満足するか もっとよい方法はないか	完了基準：指摘の修正
コーディング	欠陥、CPU, Memory 使用効率、保守性、再利用性もっと良くできないか	テストが通れば OK コピーした無駄なコードに無関心
読書	月1冊以上 よい提案には幅広い知識が必要	作業できれば不要
習得習慣	常に ”なぜ？”を考える 因果関係を分析してモデル化	言葉を記憶 (ex) バグがあるのはテスト不足
知識	メンタルモデルに統合され、 適切な提案ができる	バラバラで活用されない知識 知識と作業は無関係
脳内メモリ	2週間程度先までロード 最終ゴールまでの道筋は常に意識	昨日と今日の作業のみ
準備作業	チーム全体の作業がスムーズに 進むよう裏で作業	なし
成長	毎月進化	作業に必要なスキル獲得後、停滞

(3) 成果

まだ成長を加速する成果は得られていない。しかしながら、作業を一生懸命残業してやっている開発者にどんなスキルが不足しているかを認識させることができるようになった。スタート地点に立ったようなものであるが、今後の道筋が見えるようになったことが成果である。

また、2 章のプロセス改善推進者のスキルアップは改善活動を作業ではなく仕事に引き上げる活動であったと考えることができる。

5.まとめ

本発表では SPI Japan の活用方法として(a)発表者として参加させることによるスキルアップの方法と (b)聴講者として獲得した情報を社内で活用した事例を示した。当組織で SPI Japan がうまく活用できている背景および要点を以下に示す。

(1) スキルアップ

親会社である住友電気工業では昇進論文の制度があり、2～3 年毎に業務成果を 7 ページの論文で提出する必要がある。指導者を含め多くの工数が必要であるが指導効果は大きい。子会社の住友電工情報システムではこの制度がなく、SPI Japan で代用している。文書を書いてもらい指導するというそれなりの工数を要するが、組織力強化、組織としてのスキル維持の観点から実施している。組織としてこの取り組みを評価してもらえているので継続できている。

(2) 技術導入

プロセス改善の主な 3 要素はプロセス(手順等)、人(スキル)、ツールであると考えている。私の部署ではこの 3 つの要素を進化させることでプロセス改善を推進している。例えばスキルアップの施策として教育管理システム Moodle を活用したオンライン教材[7]を提供している。このシステムはパソコンに Linux を導入し、OSS である Moodle をインストールしてテスト的に利用しだしたのが始まりである。OSS は無料で利用できため、社内の稟議資料作成の手間なしで試してみることができる。このように新しいツールを試せる環境およびスキルを持っていることが重要である。昨今、DX のキーワードで IT 部門以外でもローコード開発ツールを使って簡単なシステムを開発している時代になっている。プロセス改善推進部門でも手が動く要員が必須になると考えられる。

改善活動では経験のない技術や手法を導入する必要があり、知らない技術を知るための場として SPI Japan は重要である。単に聞くだけではなく、改善に繋げる事例としてこの発表が参考になれば幸いである。

参考情報

[1] 中村 伸裕, ”ソフトウェア・プロセス DX の提言”, SPI Japan 2022,2022
[2] 中村 伸裕, “デジタル・アシスタントによる測定の自動化”, SPI Japan 2023, 2023
[3] 野尻 優輝, “基幹システムにおける自動テストの課題と対策”, SPI Japan 2024, 2024
[4] 和田 卓人, "組織に自動テストを書く文化を根付かせる戦略（2024 版）", SPI Japan 2024, 2024
[5] JavaScript Testing Framework, https://jestjs.io/ja/
[6] 倉貫 義人, "いいソフトウェアを創るための組織と経営", SPI Japan 2024, 2024
[7] 岡本 優奈, “開発実習を含む集合研修の e-Learning 化と成果”, SPI Japan 2021, 2021

1B1　大規模な LowCode 開発を実践し得られたノウハウ･教訓　伊丹 淳（株式会社 NTT データグループ）

＜タイトル＞

大規模な LowCode 開発を実践し得られたノウハウ・教訓

＜サブタイトル＞

―

＜発表者＞

氏名(ふりがな)：伊丹 淳(いたみ　じゅん)
所属：　株式会社 NTT データグループ　ＩＴマネジメント室　システム開発担当

＜共同執筆者＞

氏名(ふりがな)：珍田 晋之介(ちんだ　しんのすけ)
所属：　株式会社 NTT データグループ　ＩＴマネジメント室　システム開発担当

＜主張したい点＞

大規模な LowCode 開発を実践する際に、開発プロセスをどのように設計＆準備すべきか工夫点をまとめた。
NTT データグループの基幹系システム更改で得られたノウハウの中で特に重要と考えるポイントを絞り紹介する。
また開発が進み開発メンバーの習熟が進む中で、開発プロセスで必要となる成果物も変わってくる。このあたりの考えについても、実際の開発メンバーの声（アンケート）を踏まえ解説する。

＜キーワード＞

大規模、LowCode、開発プロセス

＜想定する聴衆＞

LowCode 導入を検討しているプロジェクトのリーダ、または PMO

＜活動時期＞

2020 年 4 月～（継続中）

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
■ 改善活動を実施したが、結果はまだ明確ではない段階
□ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1.背景

近年のシステム開発においては SaaS 活用や LowCode 開発など DX 化の促進が求められるようになった。一方で多くのプロジェクト経験者が感じているように、これらの DX 開発を上手く使いこなすためには、プロジェクトの特性を踏まえた様々な工夫をしなければならない（途中で問題に気づいたとして、もなかなか軌道修正が難しい)。これらのうち、LowCode 導入時の開発プロセス定義の勘所について、大規模システム開発で得られた教訓を踏まえ紹介する。

2.改善したいこと

LowCode 開発にあたり開発プロセスを導入した（個人スキルに依存したブラックボックス化を防ぐため）

・ 日本国内でもさまざまな拠点があり、600 人以上が常に開発に関わるような基幹系システムの更改のため、LowCode が推奨するようなアジャイル拠りの開発では情報の正しい要件を正確に把握し、試験に繋げることが難しいと考えた。このため LowCode 開発用の開発プロセスを定義した。

※開発立上げ当初は、世界を見渡しても関連ノウハウがほとんど見当たらず、試行錯誤を繰り返しながら体系化した

・ 今回我々が採用した LowCode ツールに限らず、新たな DX 製品を積極採用しなければならないケースにおいては、この事前準備（技術検証、Fit&GAP 分析、PoC など）をどれだけやり切り正確に影響を見極めできるかで、開発の生産性や品質が大きく変わる。この期間において、特にどの点に注意して検証しルール化しなければならないのか、まとめる。

3.改善策を導き出した経緯

NTT データグループの基幹系システムの更改（初回）では、事前準備を十分に確保できなかった。限られた期間・予算・リソースの中で、基本構想や DX 製品選定（見極め選定）を進めていた。(従来の FW 等を活用した C/S システム開発のイメージで準備・計画し進めた）その結果、PJ 開始後に LowCode 導入に関する問題が現れた。

4.改善策の内容

① Fit&GAP
② 要件定義・外部設計フェーズのアクティビティ定義
③ 製造フェーズ（作り方をパターン化する、サンプルコードを PJ 全体で共有する）とにかくナレッジを増やす
④ 試験フェーズ（現行システムのデータをマスキング活用しながら、できるだけ試験の網羅性・充足性をあげる）

5.改善策の実現方法

開発プロセスのルール追加/変更として、随時展開。

プロジェクトで定める会議体の中で、これらに対する改善要望や問題・課題などを確認。フィードバックをもらう。

フィードバック結果を踏まえて、更なるルール改善等を実施。

6.改善による変化や効果

業務を跨る要件や設計の確認にあたり、拠り所をもって確認することができた。

（特に開発初期の段階では、技術レベル差を考慮し LowCode アプリを直接見て確認する方法は行わず）

HUB への製造に必要な情報を可視化することができた（これらを流用・参照し他チームのレベルも上がった）

△試験における要件の充足性・網羅性確認（特定の個人に依存することなく確認することができた）

＜生産性から見た推移＞
要件定義完了までに画面、データモデル、IF 設計は外部設計レベルまで進めなければならない。要件定義はこれら外部設計で本来やるべき設計を一部前倒しで進める必要があり、これらの進め方について（一部並行し進める必要があるためお客様や開発内部で合意する必要がある。
技術が安定しない（ノウハウが枯れていない）結合試験では想定外の問題（しかも長期間にわたり解消・解決になる）が発生するため、一定のバッファ期間を設けて計画する必要がある。
※LowCode のアーキテクチャの課題や環境、リリースの問題がクリアされない限りは、この期間で何かしら問題が現れるため、業務の試験期間の前にこれらを検証・確認するための疎通検証期間が必要。

7.改善活動の妥当性確認
・プロジェクトメンバーに対するアンケート確認。

参考情報
[1] https://www.nttdata.com/jp/ja/trends/data-insight/2021/030401/

1B2　スマートロックの QR コード対応開発における XDDP の適用　15 ヶ月の派生開発計画を 3 ヶ月に短縮した、マルチパートナー連携の成功事例　本田英稔（株式会社構造計画研究所/派生開発推進協議会）

＜タイトル＞

スマートロックの QR コード対応開発における XDDP の適用

＜サブタイトル＞

15 ヶ月の派生開発計画を 3 ヶ月に短縮した、マルチパートナー連携の成功事例

＜発表者＞

氏名(ふりがな)：本田英稔
所属：　株式会社構造計画研究所/派生開発推進協議会

＜共同執筆者＞

氏名(ふりがな)：
所属：

＜主張したい点＞

現在は垂直統合型の開発から、クラウドサービスを連携する（つなぐ）ことで新しい価値を提供する時代になっている。マルチパートナー(クラウドサービス各社・製造担当社)が関わる超短納期開発において、USDM(要求仕様記述法)を全ステークホルダー共通の「錨(アンカー)」として要求と影響範囲を可視化することが、迅速な合意形成と手戻り防止に極めて有効である。これにより、計画を 80%（15 ヶ月→3 ヶ月）短縮するという超上流での要求変更への対応を実現した。

＜キーワード＞

XDDP、派生開発、USDM、トレーサビリティマトリクス、プロジェクト管理、短納期開発、ステークホルダー管理、IoT

＜想定する聴衆＞

プロジェクトマネージャ、品質保証担当者、複数のサプライヤーが関わる製品開発の担当者、IoT システムの開発者

＜活動時期＞

今回の発表は 2021 年 11 月～2022 年 6 月

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
□ 改善活動を実施したが、結果はまだ明確ではない段階
☑ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1.背景

スマートロックは、IoT デバイス、クラウド、スマホアプリなど、技術領域が多岐にわたるサービスである。このようなシステムで派生開発を行う場合、サービスのビジョンを実現するための領域ごとの変更要求、追加機能要求、および領域間の連携が重要となっている。
今回発表する事例は、ホテル事業者への導入が決定したスマートロック「RemoteLOCK」に対し、ユーザへの体験価値を高めるために QR コード対応を超短期間での派生開発の例である。 この派生開発では、自社のクラウド、ファームウェア担当に加え、スキャナを搭載するハードウェアパートナーや、PMS(宿泊管理システム)パートナーなど、複数のステークホルダーが密に連携する必要があった。

2.改善したいこと

当初、製造担当パートナーからは 15 ヶ月の開発計画が提示されていた。 しかし、導入先ホテルの開業日が決まったことで、経営トップ間の合意により、開発期間をわずか 3 ヶ月に短縮するという決定が下された。 これは「間に合わないモンスター」 とも呼べる状況であり、従来の勘と経験に頼ったプロジェクト管理では不可能な、極めて高い課題であった。

3.改善策を導き出した経緯

この状況を打開するために問題の可視化が急務であった。 そこで、派生開発で実績のある XDDP(eXtreme Derivative Development Process)の考え方を採用。 具体的には、全ステークホルダーの要求と影響範囲を一枚の図で表現できる USDM(Universal Specification Describing Manner)とトレーサビリティマトリクス(TM)を、プロジェクトの中心に据えることを決定した。

4.改善策の内容

本プロジェクトの「解」として、USDM を「パートナー連携」のために活用した。 具体的には、要求内容(What)を縦軸に、関係するステークホルダー(Where)を横軸にした一覧表を作成。 これにより、「どの要求が、どのパートナーに、どのような影響を及ぼすか」が一目瞭然となった。この USDM には、営業の要求から設置環境の制約(照度など)まで、あらゆる情報を集約し、常に最新の状態に保った。

5.改善策の実現方法

USDM を全コミュニケーションの「錨(アンカー)」として活用した。 特に重要なパートナーとは週次で USDM をレビューし、影響範囲の明確化、修正箇所の最小化、機能と期間のトレードオフなどを迅速に議論した。 これにより、従来数週間かかっていた合意形成が大幅に短縮された。 さらに、USDM から各社の詳細設計やテスト計画へリンクを張り、システム全体のトレーサビリティを確保した。

6.改善による変化や効果

最大の効果は、3 ヶ月という超短納期での開発を成功させたことである。この手法の有効性は、プロジェクト終盤で発覚したファームウェアの不具合対応時に証明された。ハードウェア製造が開始された後だったが、USDM で影響範囲が正確に識別済みだったため、トレードオフ表を用いた議論を即座に行え、OTA(Over-The-Air)更新を前提に製造を継続するという最適な判断を、全関係者の合意のもと下すことができた。

7.改善活動の妥当性確認

XDDP、特に USDM を中核に据えた開発手法は、本プロジェクトの成功に不可欠であった。 複数の企業にまたがる複雑な依存関係と、厳しい納期という制約を乗り越えられたのは、この「一枚の図」によってチーム全体のビジョンと状況が常に共有されていたからに他ならない。本アプローチは、同様の課題を抱える IoT システムの派生開発において、極めて有効なモデルケースであると考える。

参考情報

[1] https://f2ff.jp/introduction/8609?event_id=etexpo-2023&fbclid=IwAR3F9JxPJqmueJfy9A-yczYa6gyKkIlBRHZdFmGX1Xtxe_wdJe01Jf4G9a8

1B3　社内サービス保守運用における作業時間短縮のためのテスト自動化と生成 AI 活用実践　飯塚陸斗（株式会社日立ソリューションズ）

＜タイトル＞
社内サービス保守運用における作業時間短縮のためのテスト自動化と生成 AI 活用実践
＜サブタイトル＞

＜発表者＞
氏名(ふりがな)：飯塚　陸斗（いいつか　りくと）
所属：　株式会社　日立ソリューションズ　業務革新統括本部 技術革新本部　生産技術部

＜共同執筆者＞
氏名(ふりがな)：
所属：

＜主張したい点＞
社内サービスの保守運用作業におけるバージョンアップ時のテスト、ユーザ向けマニュアルのメンテナンスを対象に作業時間短縮を目的とした改善活動を実施した。バージョンアップ時のテストでは、E2E テストツールによる自動化の結果として、テスト実施にかかる時間を短縮した。マニュアルメンテナンスについては生成 AI を用いることにより、マニュアル執筆にかかる時間を短縮した。

＜キーワード＞
E2E テスト、テスト自動化、生成 AI、MCP、サービス運用、メンテナンス

＜想定する聴衆＞
- テスト自動化や生成 AI を用いた業務改善に従事している人
- 定期的にバージョンアップを行うサービスの運用担当者

＜活動時期＞
2024 年 4 月～

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。
□ 着想の段階(アイデア・構想の発表)
■ 改善活動を実施したが、結果はまだ明確ではない段階
□ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞
1.背景
弊社生産技術部では、「高速開発基盤サービス」と呼ぶ社内サービスを構築・運用している。高速開発基盤サービスは、ソースコードリポジトリである GitLab や CI/CD 基盤である GitLab CI/CD など開発プロジェクトに共通で必要なツールを社内向けに提供している。また、これらのツールの社内利用を促進するために日本語による利用マニュアルや関連技術の資料も整備している。
高速開発基盤サービス運用における課題として、ツール群やマニュアルを提供するための作業に時間がかかり、関連技術のドキュメントを整備する時間を十分に確保できていないことが挙げられる。更に、今後運用対象のツールやマニュアルが増加するにつれて、ツール提供にかかる時間も増加し、課題が深刻化すると考えられる。そのためツールやマニュアルの提供における課題に対し、改善が必要であると判断した。

2.改善したいこと
本発表では、本サービス運用に関連する 2 つの作業についての改善活動を取り上げる。
① 高速開発基盤サービスのバージョンアップ(以降、VUP)作業におけるテストにおける課題
本サービスで提供している開発ツールの 1 つである GitLab は、弊部署が用意した Azure 上の環境にホストしたものを社内向けに提供している。そのため、定期的に GitLab インスタンスや Azure 上のインフラリソースを VUP する必要がある。
VUP 作業では、GitLab インスタンスおよびインフラリソースの VUP、および正常に動作することを確認するテスト（E2E テスト）を実施している。VUP 作業の際は 1 日（約 8 時間）のサービス停止を伴うが、そのうち E2E テスト作業は 1 時間を占めることが課題となっていた。
そこでテスト時間の短縮を目的として、作業手順の整備および E2E テストツールを用いた自動化に取り組んだ。
② マニュアルメンテナンスにおける課題
本サービスでは、弊社における開発シーンでのツールの利用手順を示したマニュアルを提供している。本マニュアルはツールの VUP による画面 UI の変更や新規機能追加に伴い、示された手順や画面のスクリーンショットを修正する必要がある。しかし、メンテナンスは時間がかかる上、既存の利用者に与える効果が少ない。そのため、他の運用作業に対してマニュアルのメンテナンス作業は優先度が低く、マニュアルは古い情報のまま放置されることがある。一方、マニュアルは新規利用者が最初に確認するドキュメントであり、サービスに対する満足度に影響する。そのため、マニュアルは常に提供しているツールのバージョンに合った情報を示すことが望ましい。
そこでドキュメントのメンテナンス作業の時間短縮を目的に、生成 AI 技術を使った作業の自動化に取り組んだ。

3.改善策を導き出した経緯
① テスト自動化と作業手順の整備
改善に当たり、VUP 作業担当者とディスカッションし、E2E テストに時間がかかる原因を特定した。(4-①章参照）特定した原因に対し、テスト自動化と作業手順の整備が改善策として有効であると考えた。
また昨今のソフトウェア数やリリース頻度の増加により、テスト自動化技術の需要が増加しており、社内においても普及する見込みが高いと考えられる。そのため、改善策としてテスト自動化を実践することで、社内のテスト自動化導入時に役立つノウハウを取得できると考えた。
② マニュアルメンテナンスの自動化
マニュアルメンテナンス作業を分析し、時間がかかる原因を特定した。(4-②章参照）特定した原因に対し、生成 AI の利用によりメンテナンス作業時間を短縮できると考えた。また弊社における生成 AI の利用促進として、本改善活動は類似業務に携わる担当者への有益なノウハウになると考えた。

4.改善策の内容
① テスト自動化と作業手順の整備
1. テスト自動化
E2E テストに 1 時間かかる原因として以下を特定した。

- 手作業で画面 UI を操作し、各機能を確認していること
- 同じテストを 2 回実施すること（GitLab インスタンス VUP 後・インフラリソース VUP 後）

この原因に対し、VUP 作業で実施しているテストを、E2E テストツールを用いて自動化した。これにより 1 回当たりのテスト時間を短縮できると考えた。

2. 作業手順の整備

上述した原因に加え、E2E テストに時間がかかる原因として以下を特定した。

- テスト実施が属人化している

この原因に対し、自動化したテストの実行手順を整備し、誰でも実施可能な状態にした。これにより、属人化が解消でき、テスト作業の分業により VUP 作業にかかる時間を短縮できると考えた。

② マニュアルメンテナンスの自動化

マニュアルのメンテナンス手順と作業にかかる時間は以下の通りである。

1. 【調査】 VUP に伴う変更箇所の抽出
2. 【整理】 現行マニュアルのうち、変更・追加・削除する内容を明確化(5 分)
3. 【執筆】 マニュアルの加筆・修正（91 分）

内訳（手順の操作説明を 1 ページに 4 つのスクリーンショットを用いて、計 5 ページ作成する場合）

- ブラウザ操作手順の確認・スクリーンショットの取得(手動: 56 分)
- マニュアル文面の記述(手動: 35 分)

4. 【公開】 メンテナンス後のマニュアルを公開

上記より、【執筆】作業において、文面の記述やスクリーンショット取得の作業に時間がかかっていた。よって文章作成を自動化する生成 AI や、Web ページの操作やスクリーンショット取得を自動化する Playwright MCP（Model Context Protocol）を用いてメンテナンス作業を改善することにより、作業時間を短縮できると考えた。

5.改善策の実現方法

① テスト自動化と作業手順の整備

以下の 3 つの作業を実施した。

1. テスト内容の明確化

作業担当者とディスカッションを実施し、VUP 作業で実施する 24 個のテストケースに対する詳細な実施手順を明確化した。

2. Playwright を用いたテスト自動

E2E テストツールの選定に当たり、複数のツールを比較した。比較の結果、テストの実装容易性、無料で商用利用可能、社外ナレッジの充実等の機能・非機能的な観点を満たしていた Playwright を採用した。

VUP 作業時に、予め作成した Playwright の自動テストを実行し、VUP 後の機能が正常に動作することを確認した。

② マニュアルメンテナンスの自動化

【執筆】の工程に対し、 以下の作業を実施することでマニュアルメンテナンスの自動化を実施した。

1. 作業者が生成 AI にインプットするプロンプトを作成
2. 生成 AI がブラウザ操作、スクリーンショット取得、マニュアルの記述を修正
3. 作業者が生成されたマニュアルの確認を実施

生成 AI（＋Playwright MCP）を用いてマニュアルの加筆・修正と操作画面のスクリーンショットを自動化した。

6.改善による変化や効果

① テスト自動化と作業手順の整備

【効果】

テスト自動化により、1 回に 30 分かかっていたテストが 11 分に短縮した（約 65%削減）。

また VUP 作業全体の実施時間について、属人化排除により、テスト作業と次の VUP 作業準備を平行して実施できるようになったため、1 時間（30 分×2 回)短縮した。(図 1)

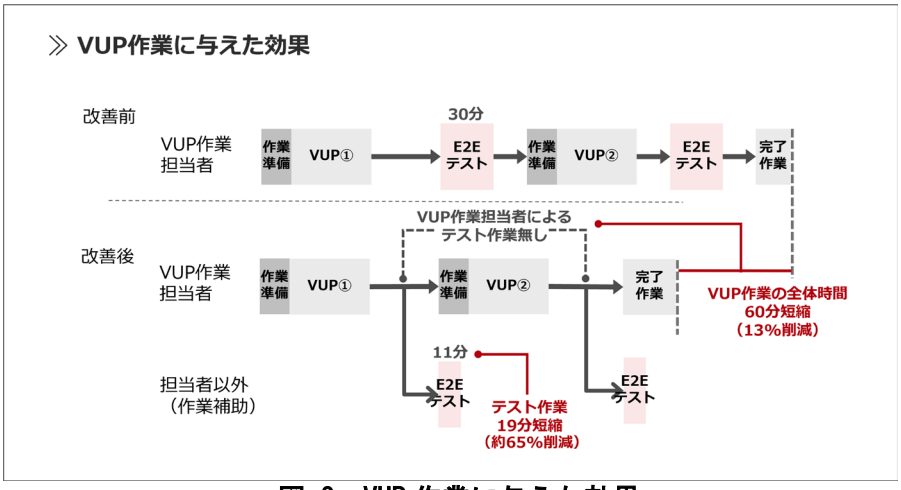


図 2　VUP 作業に与えた効果

今後継続して活用する場合、1 回の VUP につき、16 分の時間削減効果を得る。

- VUP 事前準備
  - 手順確認時のテスト　19 分×4 回（GitLab インスタンス/インフラリソースの VUP/切り戻し手順確認時）
  - テストコードのメンテンナンス -2 時間
- VUP 時に活用
  - テスト自動化と分業化　-1 時間

【変化】

VUP に伴うシステム停止時間が短縮した。

本改善活動を E2E テストの自動化事例として整備、公開することで社内のノウハウ拡充に貢献した。

② マニュアルメンテナンスの自動化

【効果】

ドキュメントメンテナンスにかかる作業時間が、96 分から 51 分へ短縮（約 45%削減）した。

内訳

- 高速開発基盤サービスのマニュアル PDF 7 ページ　修正対象総行数 41 行
  - 修正箇所・方針の計画　（改善前: 5 分 → 改善後: 12 分）
  - ブラウザ操作・手順確認　（改善前: 35 分 → 改善後: 7 分）
  - スクリーンショットの取得　（改善前: 21 分 → 改善後: 4 分）
  - マニュアルの記述　（改善前: 35 分 → 改善後: 7 分）
  - 出力の確認（改善前: 0 分 → 改善後: 21 分）

【変化】

- ドキュメントの更新サイクルの短縮でき、メンテナンス性が向上する
- マニュアル整備によるサービスに対するユーザ満足度の向上する
- 社内の生成 AI、MCP に関するノウハウ拡充に貢献する

7.改善活動の妥当性確認

①ではテスト時間を約 65%削減できた。しかし今回のテスト自動化に当たり、初回導入（環境作成・テスト作成）に 14.6 時間かかったのに対し、VUP1 回における時間の削減効果は 16 分のため、約 55 回実施する必要がある。これに対し、テストコードのメンテナンス作業の効率化やテストケースの増加により、1 回時間の削減効果が高まることで必要な実施回数が下がる可能性がある。加えて、改善②で用いた Playwright MCP をテストコードのメンテナンスで利用することにより、作業を効率化できる可能性がある。また本活動を通して得たノウハウを関連技術ドキュメントとして公開できた。

②については妥当性を確認できるまでには至っていない。

参考情報

[1] Microsoft, Playwright

[2] Microsoft, Playwright MCP server

1B4　ローコードツールを用いた PLM 業務アプリ開発　次世代 PLM の導入と業務プロセス改革　佐藤新（パナソニックコネクト株式会社）

＜タイトル＞

ローコードツールを用いた PLM 業務アプリ開発

＜サブタイトル＞

次世代 PLM の導入と業務プロセス改革

＜発表者＞

氏名(ふりがな)：佐藤新（さとう　あらた）
所属：　パナソニックコネクト株式会社

＜共同執筆者＞

氏名(ふりがな)：
所属：

＜主張したい点＞

ローコードツール導入による、業務アプリの迅速な開発と導入
内製化によるコスト削減
既存業務システムや PLM アプリケーションとの統合・連携
ユーザエクスペリエンスの向上

＜キーワード＞

PLM
ローコード

＜想定する聴衆＞

メーカ・製造業関係者
ソフトウェア開発関係者

＜活動時期＞

2023 年 4 月～

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
■ 改善活動を実施したが、結果はまだ明確ではない段階
□ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1.背景

従来の製造業では、システム連携が確立されていないことによる手作業の連携や、システムの乱立による情報のサイロ化が問題になっており、それらを改善する DX（デジタルトランスフォーメーション）の一環として、ローコード開発が導入されている。弊社においても、現状の PDM（製品データ管理）システムの老朽化対応や、業務プロセス標準化の実現に向けて、新規 PLM システムの構築を推進している。新規 PLM システムでのデータ一元管理や、個別業務要望のシステム化対応を実現するために、ローコードツールの導入を検討した

2.改善したいこと

・システムの乱立や、情報のサイロ化により、各システムや部門のなかで独立して管理している情報を一元管理する
・ユーザが直感的に操作できるインターフェースの実装により、効率的な業務の実現。生産性向上を図る
・ローコードプラットフォームは、ビジネスの変化に迅速に対応できる柔軟性と拡張性を持つため、新しい機能の追加・変更など、製品ライフサイクルの管理に柔軟に対応できる
・開発内製化により、外注コストの削減を図る。また、従来のスクラッチ開発と比べ、複雑なカスタム開発が不要となり、標準化されたコンポーネントを利用することで、開発コストを抑えることができる

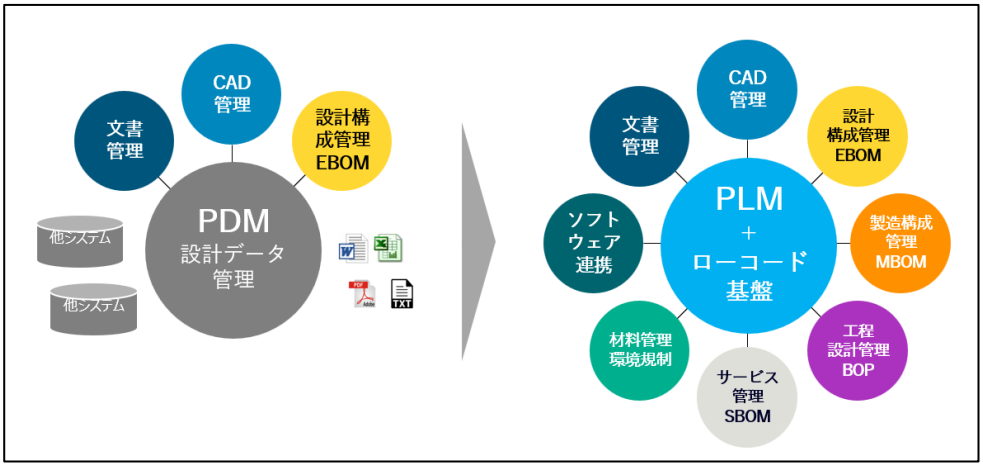
3.改善策を導き出した経緯

従来の PDM システムでは多数のアドオンや個別システムが乱立しており、それらを一元管理するための PLM システムの構築が急務となっていた。PLM システムの BestPractice 導入により、業務改革を推進していく中で、個別の業務に対応する機能がいくつか必要となった。そこで、それらを PLM システムとの親和性の高いローコードツールで外だしし、それらを内製化することで、コスト削減にも寄与することができると考えた

4.改善策の内容

PLM システムとローコードツール導入による業務改革実現に向けて、以前の PDM システムと多数の個別アドオンによる業務プロセスを標準化する必要があった。そのため、はじめにユーザにヒアリングを複数回おこなうことで、現状機能の仕様理解・要件定義を進めた。決定した機能要件に基づきローコードツールを用いた業務アプリを実装・開発することで、業務標準化の実現を目指した。

今回導入したローコードツールは、新規 PLM システムおよび、既存の業務システムとのコネクタが充実しており、データ連携・機能拡張が容易に行えるものとなっている。また、開発の内製化を図ることで、システム導入のコスト削減にも成功した



5.改善策の実現方法

PLM システム・ローコードツール導入による業務プロセス標準化に向けて、下記の活動を行なった

- ユーザの要望調査
  - 業務を行う上で必須となる機能・管理項目などについて、ユーザにヒアリングを実施した
- AS-IS のシステム仕様調査
  - ユーザと定期的に協議を重ね、現在のシステムの仕様・ロジックなどの理解を深めた
- TOBE（ローコード）仕様作成
  - ローコードアプリでの実装機能・ロジックの仕様・テーブル定義・UI 設計などの要件定義を行なった
- 実装・開発
  - 外部ベンダと協業し、機能の実装開発を進めた
- ユーザデモ
  - ローコードアプリの仕様・実装した機能が動作する様子のデモをユーザ向けに実施し、機能に対するフィードバックを受けた
- ユーザテスト
  - PLM システムとそれらに付随するローコードアプリを利用して、ユーザに実際の業務を再現いただき、ソリューションが現状業務へフィットするかどうかの見極め、使用感の確認を行なった

6.改善による変化や効果

スクラッチ開発によって、乱立していたアドオンをローコードアプリに集約することで、データ管理・アプリ運用が効率的になり、業務標準化の実現に貢献できた。また、従来のスクラッチ開発と比べ、複雑なカスタム開発が不要となり、標準化されたコンポーネントを利用することで、開発コストが抑えられた。

7.改善活動の妥当性確認

背景で述べた現状の PDM（製品データ管理）システムの老朽化対応に関して、ローコードアプリを導入することで、容易にバージョンアップ可能なシステム構成を実現した。また、現状業務を一つのシステムに集約・データの一元管理を実現することで、業務プロセス標準化に寄与することができた。開発内製化を推進することでローコードの知見がたまり、今後のさらなるローコードツール活用・ユーザ要望にへの対応にも期待できる

参考情報

1C1　アーキテクチャ進化の意思決定におけるメトリクスの活用　過去発表事例の振り返りと新たなメトリクス活用の可能性

牧隆史（テクマトリックス株式会社）

＜タイトル＞

アーキテクチャ進化の意思決定におけるメトリクスの活用

＜サブタイトル＞

過去発表事例の振り返りと新たなメトリクス活用の可能性

＜発表者＞

氏名(ふりがな)：牧　隆史（まき　たかし）
所属：　テクマトリックス株式会社

＜共同執筆者＞

氏名(ふりがな)：
所属：

＜主張したい点＞

参考文献に示す事例ではアーキテクチャの評価指標として当時知られていたいくつかのメトリクスを活用していた。近年、アーキテクチャメトリクスの認識の広がりに伴い、新しいメトリクスが提案されている。これらのメトリクスを新たに取り入れることで、リファクタリングの判断に用いるアーキテクチャ評価の精度向上が期待できる。

＜キーワード＞

アーキテクチャ、進化、リファクタリング、ソフトウェアプロダクトライン

＜想定する聴衆＞

アーキテクチャの進化およびリファクタリングに興味のある人

＜活動時期＞

2007-2010 年頃

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
□ 改善活動を実施したが、結果はまだ明確ではない段階
□ 改善活動の結果が明確になっている段階
■ その他(既発表の取り組みについて、近年提案されたメトリクスを用いて振り返りを行った)

＜発表内容＞

1.背景

プロダクトライン開発では一般に参照アーキテクチャと呼ばれる目標とするアーキテクチャを定め、これに基づいて各製品の実装を行う方法が行われる。しかしながら技術環境の変化の速い製品分野においては、製品開発と並行して参照アーキテクチャの見直しが必要になることがある。従来、こうした参照アーキテクチャの見直しは、ドメイン開発経験者の半ば直観的な判断によって行われることが少なくなかったが、その判断の妥当性を客観的に評価する手段が十分にあるとは言えなかった。参考情報に示した文献では、これに対する評価手法を提案し、組込み製品開発の実プロジェクトでの結果との比較を行った。その概要を下記 2～６に示す。本発表ではその後に開発されたメトリクス手法を用いることで、当時の提案手法を振り返り、アーキテクチャ評価の精度向上の可能性を探る。

2.改善したいこと

従来は属人的な判断によって行われていたアーキテクチャリファクタリングの意思決定を、客観的な指標に基づいて評価したい。

3.改善策を導き出した経緯

開発周期の短い複数製品の開発と並行してアーキテクチャ改善を行う必要があり、その判断基準および手法が求められていた。

4.改善策の内容

アーキテクチャ上の問題を参照アーキテクチャの問題と実装アーキテクチャの問題に分け、そのそれぞれの問題の大きさを評価することによって、意思決定の方向付けに役立てた。

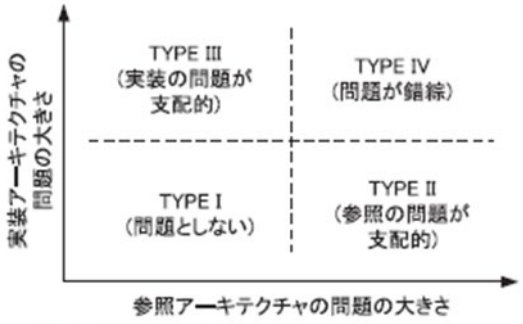


図 2　構造上の要因のポートフォリオ分析図

5.改善策の実現方法

開発過程で観測される「不吉な匂い」を起点とし、その要因を参照アーキテクチャと実装アーキテクチャの問題に分解した。それぞれの問題に対し、その問題を引き起こしている構造上の問題を明らかにし、これを定量化するための計測対象(メトリクス)を決定した。その上で、計測値を改善するためのリファクタリング項目を決定し、リファクタリングを実施前後でのメトリクスを比較し、改善の効果を確認した。

6.改善による変化や効果

実プロジェクトにおいては、本手法の提案前に専門家の経験に基づく判断によってリファクタリングが行われていたが、ポート

フォリオ分析が示唆する結果は概ね実プロジェクトでの結果と方向性が合っていることが確認できた。これにより、リファクタリングの意思決定を属人的な判断に頼ることなく、定量的なデータに基づいて進めることができるということの理解が深まったことが、プロジェクトでのその後のリファクタリングの進めやすさにつながったと言ってもよい。

7.改善活動の妥当性確認

手法提案当時の妥当性確認結果については参考情報の文献に示されているとおりである。本来は当時のデータを用いて、最新のメトリクス手法を活用して再評価するのが望ましいが、今回は当時のプロジェクトデータの入手が困難なため、文献で示したメトリクスと、その後提案されたアーキテクチャメトリクスの項目を比較することで、手法の洗練化や精度向上の可能性について議論する。

参考情報

[1] 牧隆史、岸知二：「プロダクトライン開発におけるアーキテクチャリファクタリングの意思決定法」 情報処理学会論文誌 Vol.55 No.2 1069-1078 (Feb. 2014)
[2] Niel Ford 他 島田浩二 訳：「ソフトウェアアーキテクチャメトリクス アーキテクチャ品質を改善する 10 のアドバイス」オライリージャパン

1C2 タグ情報を使ってリスクエリアを可視化し、効率的にバグを検出するテスト手法 福西章記（ソニーデジタルネットワークアプリケーションズ株式会社）

＜タイトル＞

タグ情報を使ってリスクエリアを可視化し、効率的にバグを検出するテスト手法

＜サブタイトル＞

＜発表者＞

氏名（ふりがな）：福西章記（ふくにしたかのり）

所属： ソニーデジタルネットワークアプリケーションズ（株）

＜共同執筆者＞

氏名（ふりがな）：

所属：

＜主張したい点＞

リスクの大小を可視化できる点

リスクの高い部分に対して効率的にテストできる点

＜キーワード＞

『タグ』

『タグマトリクス』

＜想定する聴衆＞

テスト業務に従事している人

＜活動時期＞

2017 年から現在に至るまで

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

☑ 着想の段階（アイデア・構想の発表）

☐ 改善活動を実施したが、結果はまだ明確ではない段階

☑ 改善活動の結果が明確になっている段階

☐ その他（ ）

＜発表内容＞
1.背景
ある PJ で「製品のミドルウェアモジュールテスト」のプロセス改善を行い、その成果確認を行っていたところ、想定外の結果となっていた箇所が有った。
この点を調査していく中で、「ミドルウェアモジュールテストのバグ検出率が非常に低い」という事が判明した。
このため、急遽、「ミドルウェアモジュールテストのバグ検出率を上げる事」が求められた。

2.改善したいこと
ミドルウェアモジュールテストのバグ検出率向上

3.改善策を導き出した経緯
1ヶ月以上かけて、約1000件のミドルウェアモジュールのバグレポート内容を地道に確認していったところ、機能や手順に傾向が有る事に気が付いた。
この傾向を上手く分類してスコア化できれば、リスクの高い手順や機能が可視化できるのではないか？と考えた。

4.改善策の内容
・テスト内容や機能仕様を連想できるワードをタグとして作成し、バグレポートに付与して集計する事で、リスクの偏在状況を可視化する。
　⇒リスクの高い箇所に対して、集中的にテスト設計を行い、テストケースを作成する。
・テストケースにもタグを付与して集計する事で、テスト密度の可視化も行う。
　⇒リスクの偏在状況に対して適切なテスト密度になるようにテストボリュームをコントロールし、効率的にバグを検出する。

5.改善策の実現方法
1.テスト内容や機能仕様やリスクを連想できるワードをタグとして作成する。
　・以下のフローでタグを作成する。
　1.1 項目洗い出しフェーズ
　　-以下方法で、ミドルウェアモジュールのテストで必要になりそうなキーワードをとにかくリストアップする
　　--テスターの経験による思いつき
　　--バグレポートを見て手順や機能を抽出
　1.2 絞り込みフェーズ
　　-文字の表記ブレや意味が同一のキーワードをマージする
　　※半数程度に絞り込む
　1.3 レビュー/正規化フェーズ
　　-以下の点に留意してテスト内容や機能仕様を連想できるワードにタグとして決める
--ミドルウェアモジュールに関連しないワードは除外する
　　--ミドルウェアモジュールテスト目的に合致する操作や機能は出現回数が少なくても追加する
　　--開発側にもレビューしてもらい、コード上、類似の操作や機能はワードをマージする
　　--開発者目線でも実装個所のイメージが湧くワードにする
　　-作成したタグの説明を記載する
--適用範囲(どのような場合に付与するかの条件)を書く

2.バグレポートの内容に合わせてタグを付与する。
　・条件に合うタグを付与する。(合致するものを付与するため、1つしか付与されない事も有れば5つ以上付与される事も有る)
　※現在は LLM ツールで大量のバグレポートに一度に付与している。(LLM ツールと比べると大幅に数は

少なくなるが、人が判断して付与する事でも実施可能。）

3.バグレポートに付与したタグを集計して、タグのマトリクスを作成する。
　・以下のフローで集計してマトリクスを作成する。
　　3.1 1件目のバグレポートを参照する。
　　3.2 バグレポート内の1つ目のタグを起点にして、2つ目以降のタグの出現回数をマトリクス上に1ずつ加算する。
　　3.3 バグレポート内の2つ目のタグを起点にして、3つ目以降のタグの出現回数をマトリクス上に1ずつ加算する。
　　※以降、バグレポート内の最後のタグになるまで繰り返す。
　　3.4 2件目のバグレポートを参照して、3.2～3.3を繰り返す。
　　※以降、最後のバグレポートになるまで繰り返す。

※作成したタグマトリクス(一部)のイメージ図

【凡例】
白：タグの出現カウント無し　（リスク低のエリア）
青：タグの出現カウント5件以下
緑：タグの出現カウント10件以下　（リスク中のエリア）
黄：タグの出現カウント15件以下
赤：タグの出現カウント16件以上　（リスク高のエリア）

4.タグマトリクスをベースにして、リスクの高いエリアに集中的にテスト設計をする。
　・以下のフローでテストケースの作成を行う。
　4.1 バグの狙いどころ(*)を検討する。
　　(*)バグレポートにおけるバグタイトルの粒度の情報。例）製品の設定をAに変更してから、機能Bを使用すると、機能Cが勝手に動作してしまう。
　　・以下のいずれかの方法で検討する。
　　4.1.1 自分で狙いどころを検討する。
　　4.1.1.1 タグマトリクス内のスコアが高いセルのタグをいくつかピックアップする。
　　4.1.1.2 選んだタグから狙いどころのバグを考える。(★この時、現在のPJの新規機能も混ぜた狙いどころにする)
　　4.1.2 過去の類似バグを見ながら狙いどころを検討する。

4.1.2.1 タグマトリクス内のスコアが高いセルのタグをいくつかピックアップする。
4.1.2.2 選んだタグを使って、過去のバグをフィルターして絞り込む。
4.1.2.3 現在の PJ に合う類似バグを選び、狙いどころのバグを考える。(★この時、現在の PJ の新規機能も混ぜた狙いどころにする)

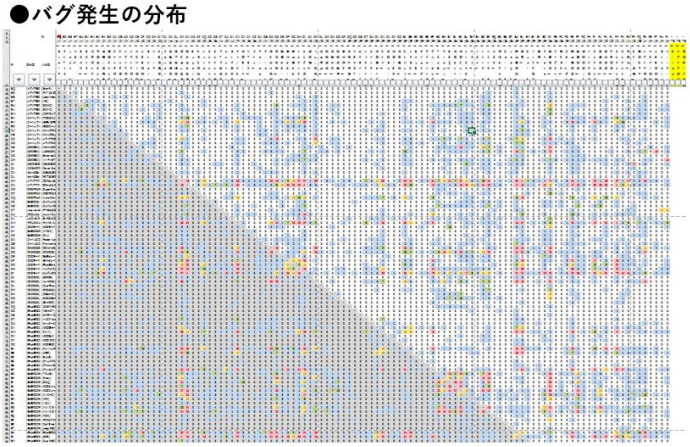
4.2 バグの狙いどころに沿ったテスト手順を記載する。
・狙いどころの内容は粒度が荒いものも有るため、「必ず実施する必要が有る条件や手順」や「任意の値を一回以上実施すれば良い条件や手順」はその旨を明記する。

4.3 作成したテストケースにタグを付与して、タグマトリクスを作成する。
・バグに付与した時と同様に条件に合うタグを付与する。

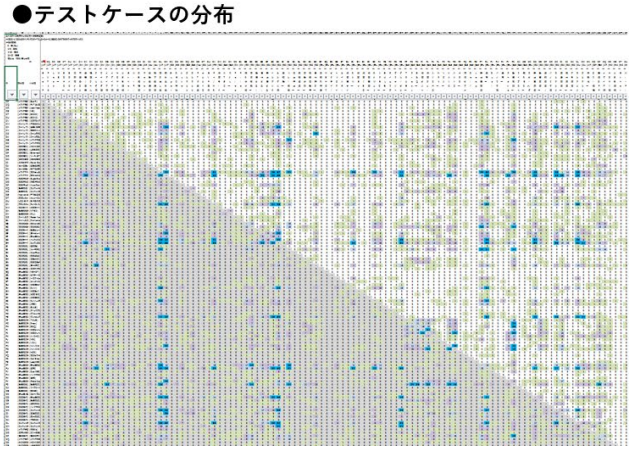
4.4 バグのタグマトリクスと、テストケースのタグマトリクスを照合して、テストの漏れが無い事と密度の適切さを確認する。
・「テスト漏れが無い＆テスト密度が十分である」になるまで、テストケースの作成を行う。

■「バグ発生数」／「テストケース実施数」＝『テスト密度』

●バグ発生の分布



●テストケースの分布



6.改善による変化や効果
従来のテスト手法と比べて、バグ検出率が、最大で約 6 倍、年間平均で約 3 倍、向上させる事ができた。

7.改善活動の妥当性確認
「傾向の分類」と「集計によるスコアの可視化」を行った上で、品質リスクの高い箇所に重点を置いてテスト設計しているため、バグ検出率が向上する事は想定の通りの結果だと考えている。
本仕組みは最初に「製品のミドルウェアモジュールテスト」に対して導入し、上述の成果を出す事ができて、その後、「他のテストも同様にバグ検出率を高くしてほしい」という依頼が来たため、本仕組みを横展開していった。
「電源制御モジュールテスト」、「ネットワークモジュールテスト」、「UI 画面全般テスト」に適用していき、各テストでも同様に 3 倍～6 倍程度の改善効果を実現する事ができている。

1C3　大規模・多層構造プロジェクトにおけるマネジメントプロセス改善 ― 統合管理ビューによる予兆検知とマネジメント品質向上の実践 ― 坂本公輝（INTLOOP Project Management 株式会社）

＜タイトル＞

大規模・多層構造プロジェクトにおけるマネジメントプロセス改善

＜サブタイトル＞

― 統合管理ビューによる予兆検知とマネジメント品質向上の実践 ―

＜発表者＞

氏名(ふりがな)：坂本 公輝（さかもと こうき）
所属： INTLOOP Project Management 株式会社
テクノロジーパートナー事業部

＜共同執筆者＞

氏名(ふりがな)：珍田 晋之介（ちんだ しんのすけ）
所属： 株式会社 NTT データグループ
コーポレート統括本部 IT マネジメント室 DX 推進部 システム開発担当

氏名(ふりがな)：伊丹 淳（いたみ じゅん）
所属： 株式会社 NTT データグループ
コーポレート統括本部 IT マネジメント室 DX 推進部 システム開発担当

＜主張したい点＞

大規模システム開発プロジェクトでは、要員数が 500 名を超えると、情報の分断や管理の複雑化により、問題の兆し（予兆）が見えづらくなる。本発表では、維持と開発が並走する複雑な環境下で、進捗・品質・課題などの情報を統合的に可視化する「統合管理ビュー」を構築し、予兆検知とマネジメント品質の向上を実現した取り組みを紹介する。類似プロジェクトへの展開や、プロセス改善の参考となることを期待する。

＜キーワード＞

大規模開発、予兆検知、統合管理、進捗管理、品質管理、故障管理

＜想定する聴衆＞

大規模プロジェクトのリーダー層、プロセス改善を担う関係者

＜活動時期＞

2024 年 4 月から開始、2025 年 10 月時点で継続中

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
□ 改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1.背景

本プロジェクトは、2024 年 4 月に会計系システムの更改を完了し、サービスを開始した維持・運用業務と、2026 年 4 月の次期開発に向けた新規開発が並走する大規模プロジェクトである。要員は維持・運用に約 100 名、次期開発に約 500 名、15 チーム体制で構成されている。
維持業務では、月次・四半期・年度末などの会計処理に伴い作業量が急増し、障害対応など即時対応が求められる。一方、次期開発では複数チームが連携するシナリオ試験が計画されており、1 チームの遅延が他チームに波及するクリティカルな構造となっている。
加えて、委託先はチームごとに異なる企業であり、開発拠点も東京・北海道・仙台・四国・博多・中国に分散しているため、中央集権的な要員管理が困難であった。こうした複雑な環境下で、PMO としてはプロジェクト全体の状況を俯瞰し、問題の兆し（予兆）を早期に捉えてアクションに繋げる仕組みの整備が急務であった。

2.改善したいこと

週次の進捗会議を通じてプロジェクト全体の状況把握を試みたが、以下のような課題が浮上した：

- ・各チームの報告は達成事項が中心で、遅延やリスクに関する情報が不足していた。
- ・繁忙期の作業量や要員の充足状況が見えづらく、計画の実現性が判断しづらかった。
- ・開発プラットフォームの違いにより、品質状況の比較が困難であった。

3.改善策を導き出した経緯

大規模プロジェクトでは、進捗・品質・課題・故障などの情報が管理領域ごとに分散し、報告者の主観に依存した情報収集では限界がある。そこで、報告者の負荷を軽減しつつ、客観的な定量情報に基づいて兆候を捉えるため、各管理領域の情報を統合的に集約・可視化する「統合管理ビュー」の構築に着手した。
このビューは、PMBOK でいう統合管理領域の考え方をベースに、複数の管理領域を横断的に分析することで、予兆の検知とマネジメントの質向上を目指すものである。

4.改善策の内容

（1）遅延状況の可視化
チーム×業務分類のマトリクスで、計画・実績を 3 段階評価し、進捗の停滞箇所を特定。前週からの推移も表示し、悪化傾向のある領域に優先的に対応できるようにした。
（2）計画の実現性の評価
報告日の前後 1 か月分の週次作業件数と遅延件数を表示し、作業集中やリカバリ可能性を判断。マイルストンやシナリオ試験の開始日など、期限に対する実現性を事前に評価可能とした。
（3）品質状況の比較
試験消化件数を母数とし、進捗率（試験消化／予定試験）と故障率（故障／試験消化）を算出。開発方式の違いを吸収し、チーム間で品質状況を横並びで比較できるようにした。

5.改善策の実現方法

各チームのメンバーが日々記録する作業・故障情報を、PMO が週次で集計し、Excel ベースのダッシュボードとして更新。トップ層・リーダー層・メンバー層それぞれの視座に応じたビューを作成し、ヒートマップやマトリクス形式で直感的に状況を把握できるようにした。

6.改善による変化や効果

リーダー層は複数の管理簿を横断する手間が省け、問題の早期発見と対処が可能に。
進捗会議では、報告内容が重要課題に絞られ、議論の質が向上。会議時間も短縮された。
トップ層は、全体の状況を俯瞰し、要員配置やリソース調整の判断精度が向上。

7.改善活動の妥当性確認

統合管理ビューの導入により、維持・開発間の要員調整が適切に行えるようになり、プロジェクト全体の遅延リスクが減少。予兆検知が定着し、マネジメントの基本動作（把握・判断・共有）が改善された。今後は AI や自動化ツールの活用によるさらなる高度化を目指し、他プロジェクトへの横展開も視野に入れている。

参考情報

—

1C4 ソフトウェア生産性向上の可視化プロセス —AI エディターの効果検証— 山崎裕司（株式会社ニデック）

＜タイトル＞
ソフトウェア生産性向上の可視化プロセス — AI エディターの効果検証 —
＜サブタイトル＞

＜発表者＞
氏名(ふりがな)：山崎 裕司（やまざき ゆうじ）
所属： 株式会社ニデック

＜共同執筆者＞
氏名(ふりがな)：
所属：

＜主張したい点＞
AI エディターの導入にあたり生産性への効果検証を行う必要がありました。生産性向上を可視化するために、ソフトウェア開発のプロジェクト管理プロセスを活用しました。プロジェクト管理プロセスを、生産性向上の可視化に活用できた事例を報告する。

＜キーワード＞
QMS, SOP, IEC 62304, プロジェクト管理, 見積もり, メトリクス, 可視化

＜想定する聴衆＞
SEPG, ソフトウェアエンジニア, プロジェクトマネージャー

＜活動時期＞
2024 年 8 月～2025 年 3 月

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。
☐ 着想の段階(アイデア・構想の発表)
☐ 改善活動を実施したが、結果はまだ明確ではない段階
☑ 改善活動の結果が明確になっている段階
☐ その他(　　)

＜発表内容＞

1. 背景

2024 年 10 月開催の SPI Japan 2024 にて、「AI 共創開発：新時代の生産性と人材育成に関するトライアル～生成 AI 活用によるソフトウェア開発プロセスと人材育成の最適化～」と題して、生成 AI 活用をコーディング工程に活用することで生産性向上と人材育成への期待を報告した[1]。概報では、生産性向上への期待を示したが、生産性の向上の程度を示すには至らなかった。この結果を受け、ソフトウェア開発の生産性を可視化するプロセスの確立と実践が必要となった。

2. 改善したいこと

既存のプロジェクト管理プロセスを活用した生産性可視化プロセスを確立し、AI エディターの導入効果を定量的に評価する。これにより、開発プロセスへの影響を最小限に抑えながら、経営層への投資判断の根拠となる定量的なデータを提供する。

3. 改善策を導き出した経緯

当社では、医療機器の開発を手掛けることから品質マネジメントシステム（QMS：Quality Management System）に基づき製品開発を行っている。ソフトウェア開発に適用する標準作業手順（SOP：Standard Operation Procedure）では、IEC 62304[2]へ準拠する各プロセスの一つとしてプロジェクト管理プロセスを定めている。しかし、IEC 62304 では実践的なプロジェクト管理手法を規定しているものではなく、医療ソフトウェアの開発における最低限の要求事項を規定している。したがって、計画プロセスは IEC 12207 や共通フレームをベースに組み立て、工数見積もりおよび日程見積もりについては、見積もり技術の参考書を用いて構築している[3]。計画プロセスの成果物として図 1 のように日程計画を算出している。プロジェクト実施中は、実績工数やタスク・アクティビティ進捗を監視するが、これはプロジェクトの健全性を確認し、必要に応じて対策を講じることを目的としている。これらのプロジェクトデータに着目し、プロジェクトの生産性を評価するため、プロジェクト管理プロセスと統合した生産性可視化プロセスとして利用することとした。なお、これらのプロセスはウォーターフォール型のプロセスを前提としているが、1 つのプロジェクトはスクラム開発を導入しており、この可視化プロセスで使用するデータとギャップのある部分を抽出して置き換えることとした。

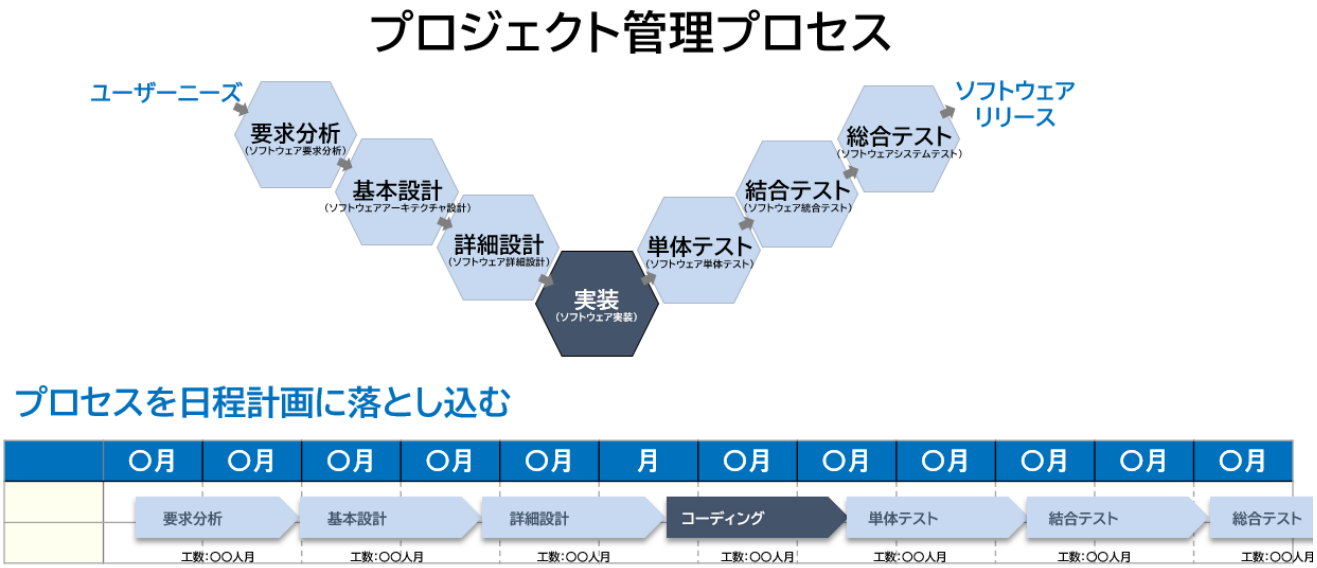


図 3　プロジェクト管理プロセスにおける各工程の日程計画の作成例

3.1 可視化プロセスの全体像

AI エディターの効果を可視化するプロセスは、以下の 5 つのステップで構成される。なお、ステップ 1 からステップ 4 までは通常のプロジェクト管理プロセスの活動である。

1. 事前準備
   - プロジェクトの特性分析
   - 初期プロジェクトメンバー結成
   - 開発メンバーのスキル評価

2. 見積もり実施
   - コード規模の見積もり
   - 工数見積もり
   - 日程見積もり

3. 開発実施
   - プロジェクト計画入力
   - 工数実績および進捗の記録

4. 効果測定
   - 日程変動の計算
   - 工数変動の計算

5. 結果分析
   - 生産性指標の判定

3.2 評価指標の選定

効果の可視化には、以下の指標を採用した。この視標は通常のプロジェクト管理プロセスの用いるメトリクスである。

1. 工数変動：Ev
   - Ev = (Ea - Ep) / Ep * 100
   - Ep：見積もり工数
   - Ea：実績工数
   - 工数変動は計画に対する実績工数の増減を表す

2. 日程変動：Sv
   - Sv = (DRa - DRp) / DRp * 100
   - DRp：見積もり期間
   - DRa：実績期間
   - 日程変動は計画に対する実績の遅れ/早まりを表す

3.3 生産性基準値の設定

AI エディター導入にあたって効果判定基準として日程変動の 5%以上の改善を基準として設定した。日程変動を設定した理由として、製品リリースの短縮化を組織目標として掲げているため日程変動を重要視した。

4.改善策の内容

生産性を評価するにあたり、対象プロジェクトを昨年のトライアルからさらに広げて測定することとした。対象拡大にあたっては、生成 AI によるプログラミングの効果をより正確に測定するため、以下の 3 つの条件を設定した。

1. 評価対象の拡大
   - 生成 AI によるプログラミングに不慣れなメンバーも含めて評価対象とした

2. 評価基準の統一
   - 当社の標準作業手順（SOP）に従ってプロジェクトの見積もりを実施（アジャイル開発を除く）
   - 見積もりには、コード規模、工数、日程を含む

3. アジャイル開発での生産性評価
   スクラムチームでは、見積もり技法が異なるため、ベロシティー推移による比較を行うこととした。
   - スプリントごとの完了ストーリーポイントの計測
   - ベロシティーの推移による生産性評価
   - AI エディター導入前後のベロシティー比較

4.1 可視化プロセスの実施手順

表 1 に可視化プロセスの流れを記載する。このプロセスはプロジェクト管理プロセスで実行させるタスクを含めている。

表 1　ウォーターフォール型開発とアジャイル型開発の実施手順比較

ステップ	ウォーターフォール型開発	アジャイル型開発
1. 事前準備	- プロジェクトの特性分析 - 初期プロジェクトメンバー結成 - 開発メンバーのスキル評価	- プロジェクトの特性分析 - 初期プロジェクトメンバー結成 - 開発メンバーのスキル評価
2. 見積もり実施	- コード規模の見積もり - 工数見積もり - 日程見積もり(*1)	- ストーリーポイントによる見積もり - スプリントバックログの作成 - チームベロシティーの設定
3. 開発実施	- プロジェクト計画入力 - 工数実績および進捗の記録	- スプリントごとの完了ストーリーポイントの計測 - バックログの実績記録
4. 効果測定	- 日程変動の計算	- スプリントごとのベロシティー計算
5. 結果分析	- プロジェクトごとの日程変動の評価 - 効果とプロジェクト特性との関係の評価	- 導入前後のベロシティーの評価 - 効果とプロジェクト特性との関係の評価

5.改善策の実現方法

5.1 使用する AI エディターの定義
本評価では、Anysphere 社が開発する Cursor[5]を AI エディターとして採用した。Cursor は以下の特徴を持つ。

1. 主な機能
   - コード補完・生成機能
   - コードレビュー支援
   - バグ修正提案
   - ドキュメント生成支援
   - プロジェクト固有の設定管理

- チーム間での設定共有

2. 選定理由
   - 高いコード生成精度
   - 日本語でのプロンプト入力に対応
   - セキュリティ要件を満たす
   - 既存の開発環境との親和性が高い

3. 使用上の制約と対策
   - セキュリティ要件
     - 機密情報の取り扱いルールの確立
   - 品質要件
     - 生成コードのレビュー基準の確立
     - テスト自動化との連携

5.2 データ収集と評価プロセス

これまで述べてきた通り、データ収集は既存のプロジェクト管理に関する SOP で従ったものとし、アジャイル開発においては、スクラムの管理指標であるベロシティーを用いた評価プロセスとした。従って、評価プロセスに関して現場でのタスクを追加することなく評価することが可能となった。なお、開発方法論による評価を柔軟に対応するためにウォーターフォール開発とアジャイル開発とで収集するデータと評価指標が異なる。

ウォーターフォール開発における見積もりプロセスを解説する。見積もりは表 2 にようにアプリケーションの機能単位で見積りを行う。見積もりは三点見積り法[2]を採用し、熟練者が規模を見積もり、その見積もった規模に対して表 3 のように機能の複雑性を楽観的/悲観的/最尤値を推定し予測値を算出する。

工数実績や進捗率は、デンソークリエイト社の TimeTracker[4]という工数管理システムを使って、工数を入力する。なお、TimeTracker の使用方法についても SOP について詳細にその使い方を定めている。表 4 にメトリクス定義表の一部を抜粋したものを示す。表 5 は、TimeTracker と帳票を連携して自動的に出力することができるため、SOP で定めた手順で日程変動および工数変動を取得でき、プロジェクト完了後に取得できる。一方でスクラム開発での生産性に使用するデータは SOP で定めた管理方法がないため、プロジェクトメンバーからデータを提出させる運用とした。

表 2　機能別のコーディング・単体テストの工数見積もり（サンプル）

No.	機能	規模 (LOC)	複雑度 (表 1)	生産性 (LOC/人月)	見積工数(人月)
1	例：アプリケーション	7833	高	2433	3.22
2	例：送液管理	1517	中	3000	0.51
3	例：電源制御	2300	低	4000	0.58
4	例：センサ監視	900	低	4000	0.23
5	例：UI 入力	900	中	3000	0.30
6	例：UI 出力	12033	高	2433	4.95
7	例：警報・ログ管理	2783	中	3000	0.93
8	例：通信	3783	低	4000	0.95
9	例：システム制御・OS	1517	中	3000	0.51
10	例：データ管理	7906	中	3000	2.64
	合計	41473	-	2805	14.78

表 3　コーディング・単体テストにおける複雑度別生産性（サンプル）

複雑度	LOC に換算した生産性（LOC/人月）			
	楽観的見積（A）	悲観的見積（B）	最もありそうな見積（M）	予想値＝(A+4*M+B)/6
高	3000	1800	2500	2433
中	3500	2500	3000	3000
低	4500	3500	4000	4000

表 4　メトリクス定義（抜粋）

メトリクス	単位	必要な測定	測定時期	定義
Sv = 日程変動	%	DRp = 見積期間（日数） DRa = 実績期間（日数）	プロジェクト進捗会議（月 1 回）	Sv = (DRa - DRp) / DRp*100
Ev = 工数変動	%	Ep = 見積工数（人時） Ea = 実績工数（人時）	プロジェクト進捗会議（月 1 回）	Ev = (Ea - Ep) / Ep *100

表 5　日程変動・工数変動入力シート（サンプル）

フェーズ Phase	見積工数（人時） Ep	実績工数（人時） Ea	計画開始日	計画終了日	実績開始日	実績終了日	見積期間（日数） DRp	実績期間（日数） DRa	工数変動（%） Ev	日程変動（%） Sv
PJ 開始	20	10	2017/04/01	2017/04/30	2017/04/01	2017/04/27	20	19	-50.0	-5.0
PJ 計画	40	40	2017/05/01	2017/06/30	2017/04/28	2017/07/02	45	46	0.0	2.2
要求分析	240	300	2017/07/01	2017/08/31	2017/07/03	2017/09/15	44	55	25.0	25.0
アーキテクチャ設計	360	420	2017/09/01	2017/10/30	2017/09/16	2017/11/15	42	43	16.7	2.4
詳細設計	520	600	2017/11/01	2017/12/10	2017/11/16	2017/12/22	28	27	15.4	-3.6
コーディング	360	480	2017/12/11	2018/01/30	2017/12/23	2018/02/20	37	42	33.3	13.5
単体テスト	440	480	2017/12/21	2018/02/25	2018/01/07	2018/03/02	47	40	9.1	-14.9
結合テスト	520	680	2017/11/15	2018/03/31	2017/12/01		98	0	30.8	-100.0
総合テスト	360	320	2017/10/01	2018/04/30	2017/10/15		151	0	-11.1	-100.0
PJ 終結	15	15	2018/05/01	2018/05/20			14	0	0.0	-100.0
合計	2875	3345	2017/04/01	2018/05/20	2017/04/01	2018/03/02	295	240	16.3	-18.6
合計（実績進捗率 100%のフェーズのみ）	60	50	2017/04/01	2017/06/30	2017/04/01	2017/07/02	65	65	-16.7	0.0

5.3 アジャイル開発における評価方法

アジャイル開発（スクラム）における評価方法は以下の通りである。

1. スプリントバーンダウンチャート
   - 日次のタスク完了状況を可視化
   - AI エディター使用によるタスク完了速度の変化を測定
   - チーム全体の生産性トレンドを把握

2. ベロシティの測定

- スプリントごとの完了ストーリーポイントを計測
- AI エディター導入前後のベロシティを比較
- チームの生産性向上を定量的に評価

3. ベロシティ向上率の計算
   - ベロシティ向上率 =（導入後ベロシティ － 導入前ベロシティ）/ 導入前ベロシティ × 100

6. 改善による変化や効果

図 2 に AI エディター導入による生産性向上効果の測定結果を示す。60%のプロジェクトで効果が確認され、効果があったプロジェクトにおいては、20%（中央値）の生産性向上が確認された。本評価ではウォーターフォールとアジャイル開発のプロジェクトを含むがアジャイル開発は中央値の 20%程度の成果となっている。ただしアジャイル開発のプロジェクト数は少ないため、開発方法論による効果検証には至っていない。

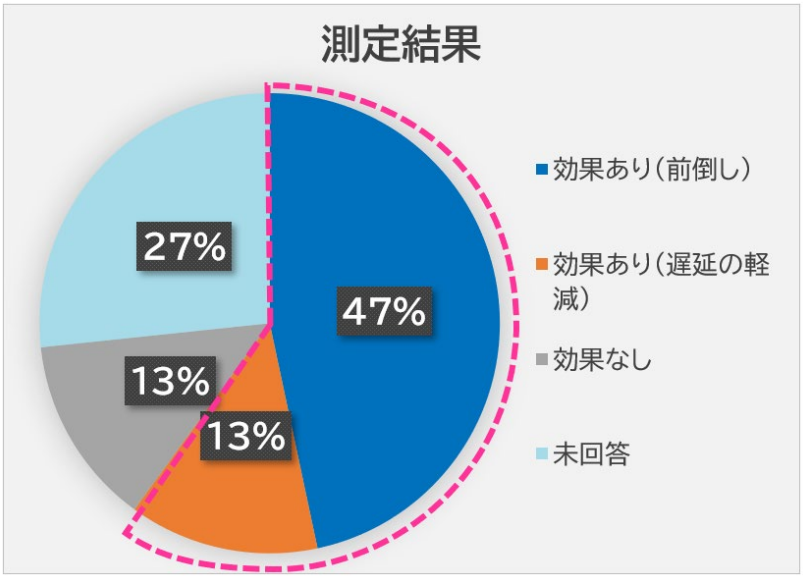


図 4　測定結果

また、効果が無かったプロジェクトを分析し、以下の特徴を持つ場合に、効果が出ないことが示された。
- 初心者
- コーディング以外の計画外作業の多いプロジェクト
- プロジェクト管理者
- レガシーコードの保守
- 独自のコーディング規約（対応により遅れが発生）

このことより、生成 AI を用いたプログラミングには、利用にあたり基準を満たす必要があると判断し、判定基準として設定した（図 3）。図 3 には、効果が出ないプロジェクトの他に効果を期待できるプロジェクトも追加して推奨例として加えた。例えば、宣言的 UI は AI エディターとの相性が良いことが、使用者へのインタビューで分かったため推奨としている。

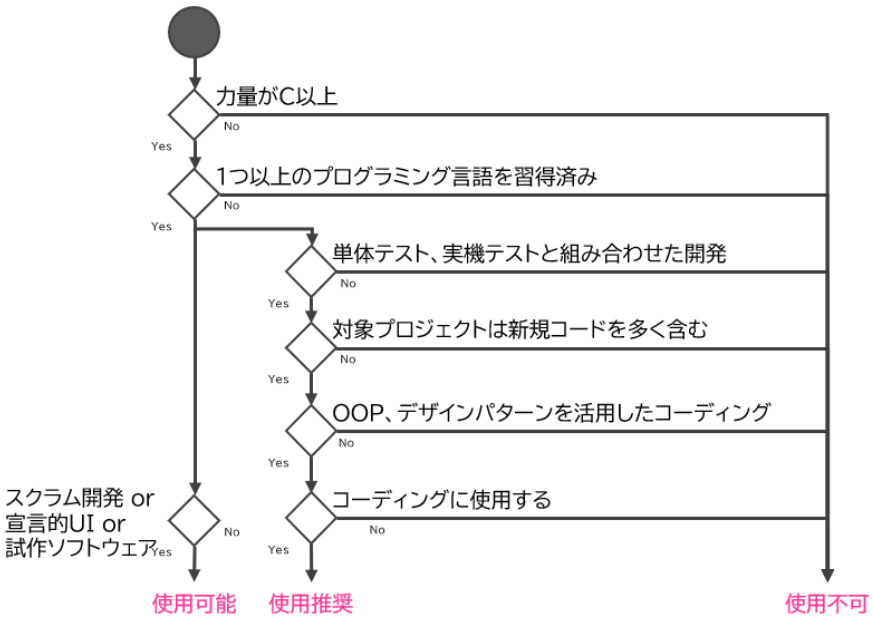


図 5　判定基準

※図 3 の力量 C：助手としてコーディングができる能力を有する

以上により、SOP で定めたプロジェクト管理プロセスを活用して、生産性向上の可視化プロセスとして確立し、その実践を通して効果を示すことができた。

当社では、本稿の成果を是としトライアルとしての試用を完了し開発プロセスに定着するツールとして正式導入を決定した。また、プロジェクト管理プロセスが、開発プロジェクトの管理のみではなく生産性可視化プロセスの一部として、経営視点に応えるデータとして用いることができた。

7. 改善活動の妥当性確認

QMS や SOP は、製品の品質を確立するためのものであるが、SOP が今回生産性評価を示せるという結果を導き出せた結果は、ソフトウェアプロセス改善に取り組む立場としても大変喜ばしいことである。

一方でスクラム開発においては、標準作業でカバーできない領域となっており、今後の検討課題である。

本稿では、プロジェクトの立上げからリリースに対しての生産性の評価はできていないため、断片的な評価にとどまる。本来は製品開発全体における QCD 視点で評価すべきものであるが、生産性はその期間が長くなればなるほど様々な要因が影響するため効果測定が難しくなるというジレンマがある。特に医療機器開発においてはその開発ライフサイクルが長期化する傾向がある。これらは今後の検討事項として継続して取り組んでいく必要がある。

参考情報

[1] 山崎裕司（株式会社ニデック），「AI 共創開発：新時代の生産性と人材育成に関するトライアル～生成 AI 活用によるソフトウェア開発プロセスと人材育成の最適化～」，SPI Japan 2024
[2] IEC 62304, "Medical device software - Software life cycle processes", International Electrotechnical Commission, 2006
[3] スティーブ・マコネル著、石井信明訳『ソフトウェア見積り』日経 BP ソフトプレス、2007 年、ISBN978-4-89100-467-1
[4] デンソークリエイト株式会社, "TimeTracker - 工数管理システム", https://www.denso-create.co.jp/products/timetracker/, 2024
[5] Anysphere Inc., "Cursor - AI-first code editor", https://cursor.sh/, 2024

2A1 生成 AI を活用したテストコード自動生成　テストの未来を変える!?　テストコード自動生成への挑戦!　森下直人（パナソニック　アドバンストテクノロジー株式会社）

＜タイトル＞

生成 AI を活用したテストコード自動生成

＜サブタイトル＞

テストの未来を変える!?　テストコード自動生成への挑戦!

＜発表者＞

氏名(ふりがな)：森下　直人（もりした　なおと）
所属：　パナソニック　アドバンストテクノロジー(株)　経営管理センター　業務推進課

＜共同執筆者＞

氏名(ふりがな)：　中前　直義（なかまえ　なおよし）
所属：　パナソニック　アドバンストテクノロジー(株)　経営管理センター　業務推進課

＜主張したい点＞

弊社業務推進課では、社内開発現場への CI ツール導入やサポートを通じて自動化を推進している。
その一環で単体テストの自動化を実現した現場があるが、手作業によるテストコード整備が課題であった。
そこで生成 AI を活用した単体テストコードの自動生成にトライした結果、効率化に繋がることを確認した。
試行錯誤はあったが、生成 AI の特性を理解して利用する必要があることがわかった。

＜キーワード＞

生成 AI、ChatGPT、GitHub　Copilot、CI、Jenkins、自動化、単体テスト、テストコード、テスト

＜想定する聴衆＞

SEPG、マネージャ、プロジェクトリーダ、テスト担当者

＜活動時期＞

自動化推進は 2018/4～継続。　生成 AI 活用は 2024/10～継続。

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
■ 改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
■ その他(生成 AI 活用は拡充中　　)

＜発表内容＞

1. 背景

2017 年度に弊社で連続して発生した品質問題をきっかけとして、経営層より品質向上の指示を受け、業務推進課（当時は品質管理課）にて静的解析ツールの選定と導入、サポートから始まり、解析の自動化、構造解析ツールの導入、リバースエンジニアリングの自動化、単体/結合テストの自動化、CI ツール Jenkins によるツールの自動化のサポートを順次実施し、それらを通じて、品質向上と効率化を推進してきた[図 1]。

2024 年度からは弊社戦略企画室と連携して自動化を希望するプロジェクトを募集し、サポートを行い、弊社内の自動化事例報告会にて結果を共有・発信することで自動化の横展開を図っている。

なお、弊社では外部からの委託によるソフトウェア開発やシステム開発が主業務であるためプロジェクトの事業分野は様々であり、顧客からプロセスやツールを指定されることが多い。

このため、弊社としての推奨ツールはあるものの、部門やプロジェクトによって採用されるツールは様々であり､CI によるツールチェーンの構成や自動化のスコープもプロジェクトによって異なっている[図 2]。

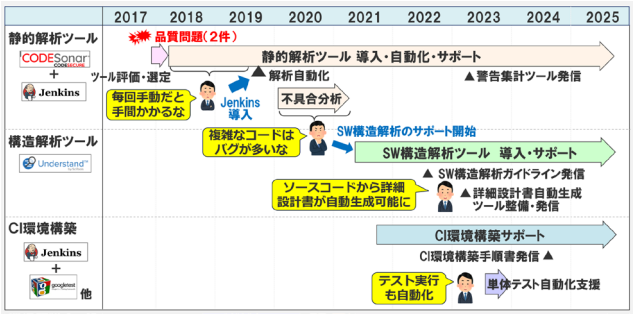


図 1.　これまでの取り組み

工程 / プロジェクト	トレーサビリティ管理	ビルド	静的解析	単体テスト	結合テスト
プロジェクトA		■ →	■ →	■	
プロジェクトC	■ →	■ →	■ →	■ →	■
プロジェクトD			■ →	■ →	■
プロジェクトE		■ →	■ →	■	
プロジェクトF		■ →	■ →	■	
プロジェクトG	■ →	■ →	■		
プロジェクトH	■ →	■ →	■		

図 2.　各プロジェクト(PJ)の CI ツールチェーン

2. 改善したいこと

単体テストの自動化例として、プロジェクト A では 4100 件の単体テスト項目をソースコードの更新ごとに任意タイミングや深夜時間を利用し、200 回/年以上自動実行している。単体テスト 1 回あたりの実行時間は９分であり、インクリメンタル開発における迅速なテスト実行と不具合の修正を実現し、大幅に効率化されている［図 3]。

しかしながら、単体テストコードは手動で整備する必要があり、テストコード整備自体の自動化は難しく、テスト自動化の横展開が進まない一因となり得るため改善したいと考えた[図 4]。

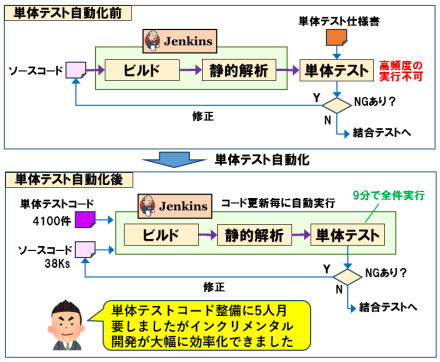


図 3.　単体テスト自動化事例

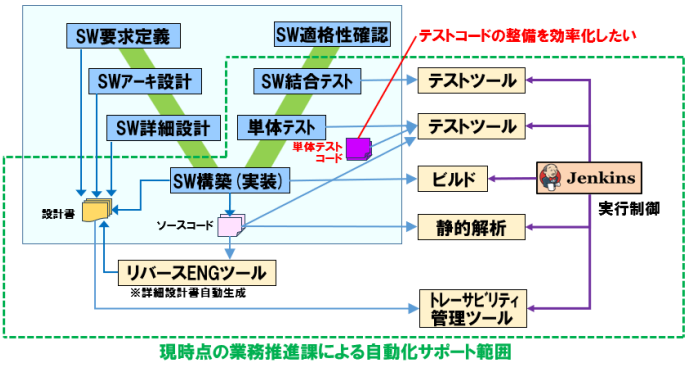


図 4.　改善したいこと

3. 改善策を導き出した経緯

2023 年度より、PanasonicHD 全体で ChatGPT ベースの PX-AI と呼ばれる生成 AI ツールが利用可能で、社内情報も扱えるようになっている。

委託業務での生成 AI の利用にあたっては顧客の許可を得るルールがあるが、許可を得たプロジェクトでは、ソースコード生成や設計検討などへの活用が始まっており、テストコード生成にも生成 AI を適用で

きないかと考えた。
生成 AI によるテストコードの自動生成を実現すると、生成したテストコードはその後のリファクタリング、保守、機能拡張に活用できるため、プロジェクトのライフサイクルを通じた効率化効果は大きいと考えている［図 5］。
なお、ソースコードからテストコードを整備すると詳細設計の検証にはならないが、詳細設計とソースコードの整合性は
レビューにて確認することとしている。

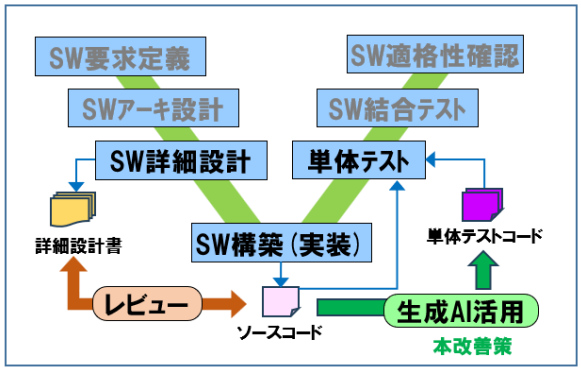


図 5.　生成 AI 活用によるテストコード生成

4. 改善策の内容
2024 年度下期に、プロジェクト B より「単体テストを自動化したい。単体テストコードを効率的に整備したい。」との相談を受け、「生成 AI を活用しませんか。Try させて欲しい」と提案したところ、「お願いする。顧客要件である C1 カバレッジ 100%を達成したい。生成 AI の活用ノウハウが欲しい」との要求を受け、実施することとした。
こうして、プロジェクトから一部のソースコード提供を受け、生成 AI 活用ノウハウ作りを開始した。
生成 AI ツールとしては、まず、PanasonicG 内で誰でも利用できる利便性を考慮し、PX-AI（ChatGPT ベース）を採用した。単体テストフレームワークは GoogleTest である。

5. 改善策の実現方法
当初の 1 ヵ月は ChatGPT（PX-AI）を用いた単体テストコードの生成に取り組んだが、プロンプトへの入力文字数に制限があり、大規模なソースコードは分割してコピー＆ペーストする必要があった。また、ビルドを通すために手動で修正を加える箇所が多く、それでも C1 カバレッジ 100%にはならず、手動でテストコードを追加する必要があったため、テストコード生成の用途には合わないと判断した。
そこで、プログラム支援に適した生成 AI を調査したところ、社内で導入が始まった GitHub Copilot を利用することにした。
VSCode 上でソースコードから直接テストコードを生成でき、補完機能も充実しており、操作性が良く、ビルドを通すための手修正箇所も少なくなった。
テストコード生成のプロンプトは、最初の関数では 5 回に分けて指示する必要があったが、同じ指示を繰り返すことでコンテンツ保持機能により、1 回のプロンプトで生成できるようになり、効率も向上した。
また、生成されたテストコードは一部手修正が必要であったが、カバレッジ計測ツール（gcov）を用いて全て C1 カバレッジ 100%であることを確認した［図 6］。
また、テストコードの整備工数は生成 AI へのプロンプトを試行錯誤で最適化したり、手修正に対する人間の習熟度の向上により、効率化できることがわかった。
　また、最初から用途に適した生成 AI を選択することの重要性を実感した［図 7］。

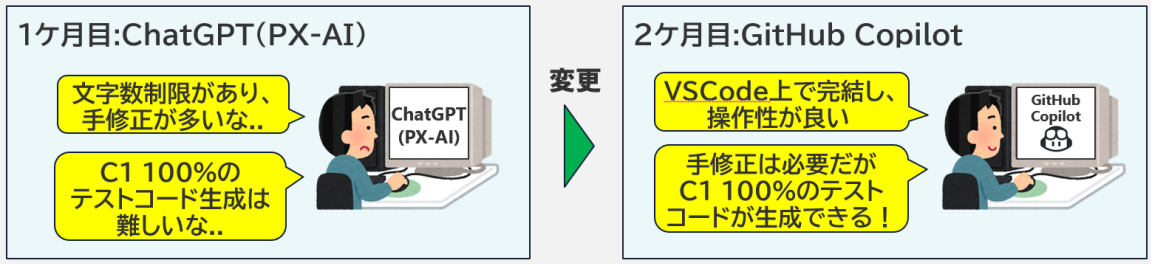


図 6.　改善策の実現

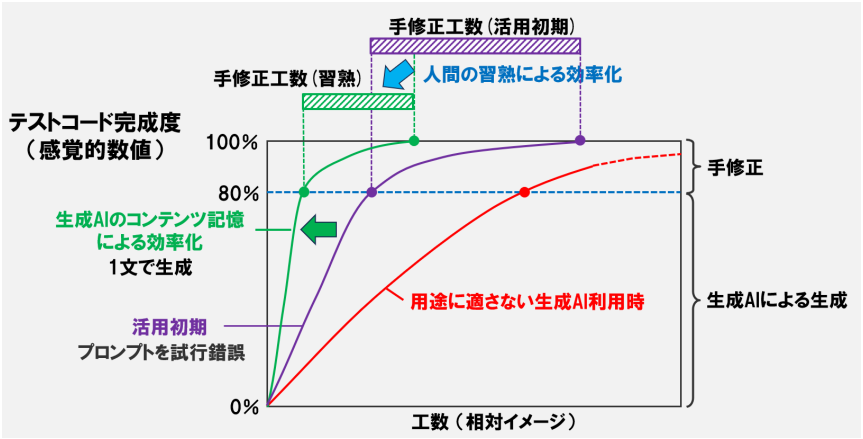


図 7.　生成 AI 活用によるテストコード完成度と工数

6. 改善による変化や効果

生成 AI によるテストコード生成フローとノウハウをまとめ、現場に説明したところ、期待通りとの評価を頂いた。

また、本事例の概要とノウハウは弊社内の自動化事例報告会を通じて社内で共有した［図 8］。

この時点では、手修正に 1 関数で数時間を要することもあったため、効率化策として、GitHubCopilot のインストラクションファイルと Agent モードを利用することとした。

実際にプロジェクト B が単体テストを開始するのは 2025 年度第２四半期の予定であったため、効率化策を含めて 2025/7 月よりサポートを開始した。また、横展開としてテスト自動化を支援したプロジェクト A にも声をかけ、生成 AI による単体テストコード生成のサポートを開始した。

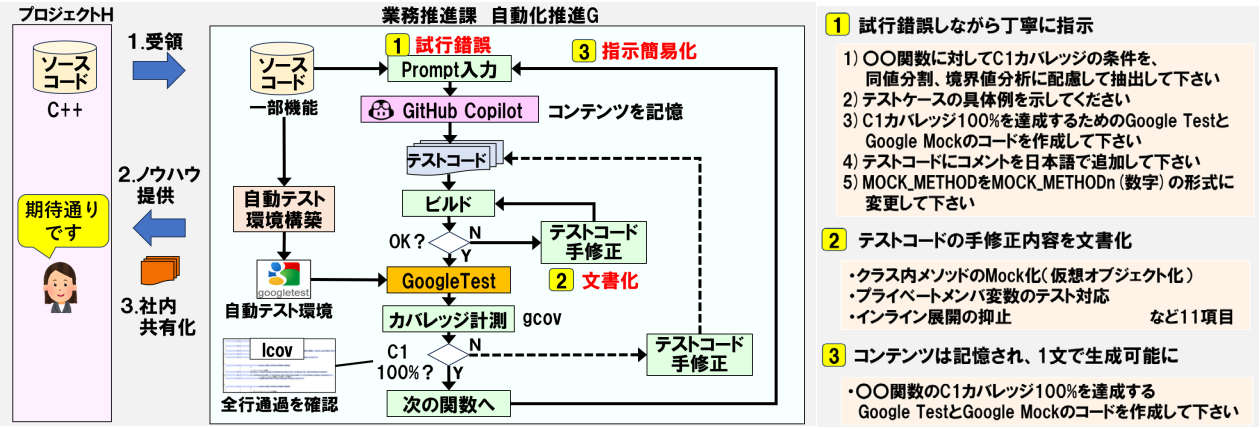


図 8.　生成 AI によるテストコード生成フロー

7. 改善活動の妥当性確認

2 件のプロジェクトについて定期的な情報共有などを通じてサポートした結果、半数の関数で生成 AI のみで C1 カバレッジ 100%の単体テストコードを自動生成できることが確認された。

また、途中経過ではあるが、テストコード整備工数は手修正を含めても人手による整備に対して大幅に削

減できており、生産性が飛躍的に向上することを確認した［図 9］。
プログラミング支援に適しているとされる GitHub　Copilot であってもテストコード生成の完全な自動化には至らなかったが、引き続き、手修正内容をインストラクションファイルの設定などを通じて効率化を図る予定である。
ただ、生成 AI のみでの完全な自動生成は追求せず、生成されたコードのレビューと修正に掛ける工数とのバランスをうまく取る方が良いと感じている。
今後は、生成 AI による複雑なソースコードの自動リファクタリングや、AI エージェントによるツール間の自律的な実行制御などに取り組み、そのノウハウの蓄積と横展開を通じて、更なる効率化、品質向上を追求していきたい。

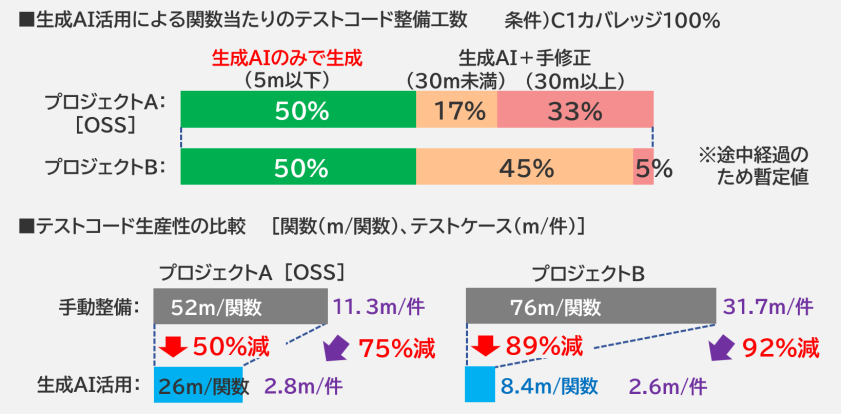


図 9.　GitHub Copilot による単体テストコード生成結果（途中経過）

参考情報

[1]　AI アシスタントサービス「PX-AI」に OpenAI（米国）の最新モデル「GPT-4」を導入　～AI 機能強化により、事業の競争力強化へ～ | 経営・財務 | 企業・経営 | トピックス | Panasonic Newsroom Japan：パナソニック ニュースルーム ジャパン
[2]　【徹底解説】GithubCopilot とは？ / メリット・デメリットは？ #生成 AI - Qiita

2A2 ソフトウェア開発における DX～AI 活用へ DX 活動を基盤にした AI（LLM）導入事例 寺村幹夫（株式会社デンソー）

＜タイトル＞

ソフトウェア開発における DX～AI 活用へ

＜サブタイトル＞

DX 活動を基盤にした AI（LLM）導入事例

＜発表者＞

氏名(ふりがな)：寺村幹夫（てらむらみきお）
所属： 株式会社デンソー ソフトウェア技術３部

＜共同執筆者＞

氏名(ふりがな)： 村端亨太（むらはしこうた）
所属： 株式会社デンソークリエイト 生産革新部

＜主張したい点＞

技術の進展が目覚ましく、AI、（特に LLM：大規模言語モデル）をソフトウェア開発に取り入れることは、品質、コスト、納期（QCD）の向上の観点から必須の要素となっています。しかし、現時点では LLM の回答に揺らぎがでることが難点として捉えられることが多いです。揺らぎのない回答を求めるのであれば、LLM を無理に使用せず、規則性を持ったルールベースの自動化を検討する方が良いと考えます。LLM の活用はむしろ、その揺らぎを利用する方法が良いと考えます。 このようなプロセス改善手段の選択肢が増えるほど、見極めが難しくなるため、体系的に開発・設計情報を整備する DX 活動を基盤とした LLM 活用を推奨します。この際、プロセス改善推進者（SEPG、PMO）はデジタル的な改善に向け、アナログ的な活動が必要となります。

＜キーワード＞

DX、 改善、 変革、 標準開発プロセス、 デジタル化、 自動化、 ＡＩ、 LLM（大規模言語モデル）

＜想定する聴衆＞

ソフトウェア開発者 特にプロセス改善推進者となる SEPG、PMO の方

＜活動時期＞

2024/5~2025/5

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
■ 改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1.背景

弊社は、経済産業省と東京証券取引所および独立行政法人情報処理推進機構が共同で進める「デジタルトランスフォーメーション銘柄(DX 銘柄)2025」に、初めて選定されました。
2023 年に JASPIC で “ソフトウェア開発における DX 活動の道しるべ” というテーマで発表させていただきましたが、今回はその続編として発表させて頂きます。前回発表した 23 年当時は、全社レベルで仕事の進め方を変革するための DX 活動が強力に推進され始めたところでした。それから 2 年が経ち、活動の定着と浸透のために効果計測のインフラも整い、活動が軌道に乗ってきました。それと並行して、さらなる競争力向上を目指し、新しい技術である AI 活用（LLM）をソフトウェア開発に取り入れることが次の課題となりました。

2.改善したいこと

ソフトウェア開発の改善活動は単なる一時的な取り組みではなく、継続的に進化し続けるプロセスとして確立する必要があると私は考えます。新しい技術である LLM のソフトウェア開発へ積極的に活用することで、革新の速度と質を向上させると共に、一過性の留まるのではなく、これからも加速していく LLM の技術的進歩に、同期したプロセス改善の基盤を構築したい。特に LLM 導入時の躓きでネガティブな先入観を与えてしまうと、改善活動のスタートに支障を来たすこともあり、LLM の導入活動の立ち上げ時の躓きをプロセス改善推進者として回避したい。

3.改善策を導き出した経緯

弊社は、ChatGPT などの LLM の導入によって、ソフトウェア開発に関する膨大な知見を効率的に抽出し、品質・コスト・納期（QCD）の向上を図ろうとしました。しかし、導入初期には回答にばらつきがあり、ソフトウェア開発者自身が精度向上の具体的な手法を模索している状態でした。模索して解決策が見つかったとしても、大規模言語モデル自体の技術進歩が早いため、知見の抽出に固執するのではなく、LLM を他の思考プロセスへの応用も検討し始めた。
検討の過程で、これまで進めてきた DX 活動においてデジタライゼーションを推進する中で、人が考えるべきタスクを絞り込めるのではないかと考えました。 また、LLM の回答のばらつきを多様性として活かす方針を取り、AI とのコミュニケーションをヒトと同じように扱い、必ずしも高い回答精度を求めないことにしました。さらに、LLM とのインターフェースを強化するためにはプロンプトエンジニアリング能力の向上が鍵であると考え、本格的にプロンプトの開発を実施する前にコミュニケーションスキルのリスキリングが必要であると感じた。

4.改善策の内容

・コミュニケーションスキルのリスキリニング （現業務の言語化も含む）
・LLM を使いたい業務とのマッチングプロセス
・DX ツールへ LLM を起動させるための定型プロンプトの準備
・DX ツールへの LLM 実行、自動化、手順ガイド 実装

5.改善策の実現方法

LLM 導入活動として、LLM 使いこなしスキル、DX ツール実装及びプロンプトコンテンツ開発、
実開発時の運用プロセスの整備 の 3 STEP 分け、確実にステップアップしていった。

・STEP1：LLM 使いこなしスキル習得

2024 年度は、効果を急がず、投資期間と割り切って計画。活動メンバの募集条件として、ミスを許容し、失敗を好み、自分の意思を持つことを条件としました。Fail-Fast の原則に基づき、「検討」よりも「検証」を重視することを合意し、活動を開始しました。DX が進み、開発に関する情報が体系化されている場合、LLM 導入が容易であり、DX と AI 活用は相互に補完する関係になるのではないかと仮定し DX が進んだ環境下で導入検討を開始。また、LLM 導入には柔軟なツールの使いこなし能力が求められ、言語で伝える技術は LLM 活用には不可欠であるため、一般的な伝え方テクニックや問いかけ方法の参考として FAQ の仕組みなども学ぶこととした。

・STEP2：プロンプトコンテンツ開発　及び、実開発への実践

開発業務の言語化する際に、整理用のプロセスモデルとテンプレートを準備しました。そのテンプレート（図 1 、図 2 参照）を使用し業務工程を Input/Output/考えるタスクに分解し、考える箇所を LLM の支援候補箇所として特定しました。LLM ができることとして、生成、検証、フィードバックループの実施タイプがあり、6 つのタスク（分類、抽出、校正、要約、形式変換、演繹・演算）を遂行できるとし、また結果を生成するための情報源として一般知識情報と専門知識情報に分かれていることを理解させました。これらを踏まえ、考える箇所と LLM でできそうなところを整合し、整合が取れそうな箇所に対してプロンプト開発を行うこととした。

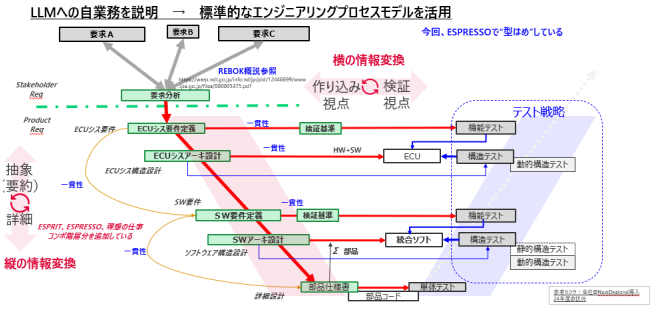


図 1 ：開発業務整理用のプロセスモデル

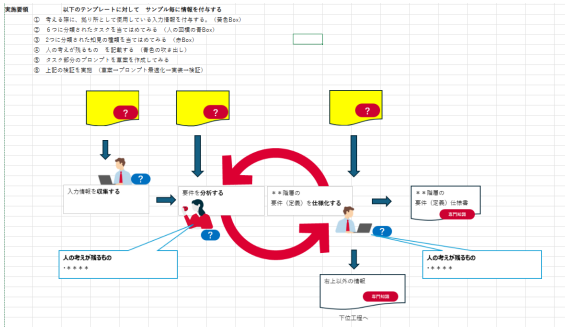


図 2 ：開発業務整理用のテンプレート

・STEP3：DX ツール実装と運用プロセスの整備

プロンプト開発では、よく使用するプロンプトを特定し、社内外の事例を収集して提供し、真似てみて、メンバー間で共有し、さらに試すといったことを繰り返し、LLM のコツを掴み、アドバイザ・レビュアの一員レベルに達するまでプロンプト開発しツールで起動できるよう実装。 また、 ルールベースの検証や手順のガイドポップアップ といったことをユーザが任意に設定できるツール機能も組み込み、自動化と LLM 活用とを適切に分別しながらプロセス改善していく基盤を構築しました。

＜分別の例＞

タスク：成果物間で一貫性あるトレーサビリティを確保する

→ トレーサビリティのぬけもれ検証は・・・。 ルールベースとして検証可能で自動化で対応

→ 一貫性の検証は・・・。 LLM で一貫性か否かの判断を支援

6.改善による変化や効果

・AI の導入活動として約 1 年間通して、大きく迷走することなく、活動初期のネガティブな印象を与えずに活動を継続、遂行することができた。

・LLM の有効活用するために必要な環境のポイントを把握することができた。（図 3 参照）

・活動メンバの LLM の理解度や応用力の向上だけでなく、ソフトウェア開発プロセスの理解度の向上及び、D X ツールの理解度の向上も同時に底上げすることができた。 （アンケートデータからの判断）

・活動メンバからもLLM活用したトレーニングを企画しはじめ、ソフトウェア開発の持続的な改善を進める気運が生まれた。
・LLM導入に対して、プロセス改善推進者（SEPG、PMO）の課題・心構えも提示できた。

導入のための環境

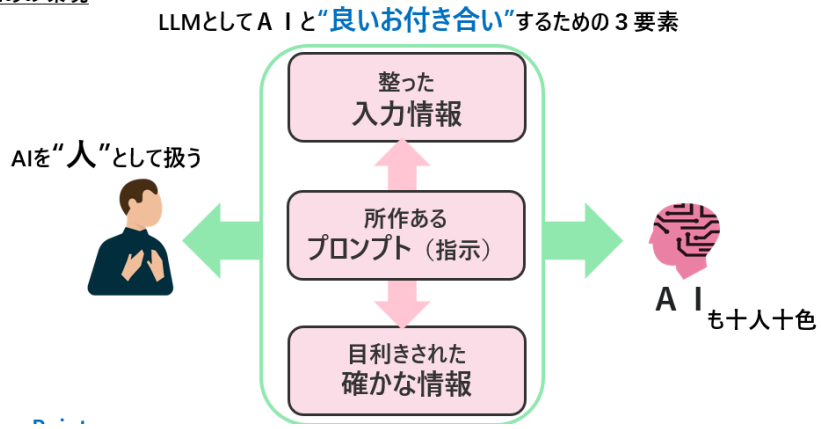


図３　LLMの有効な結果を得るための環境

7.改善活動の妥当性確認

以下、これまでのDX活動を基盤として活かし、その延長線上としてソフトウェア開発へのAI（LLM）導入の活動を遂行することができた。この成果から、持続的にプロセス改善が進み、開発力が向上していく方向性が定まった。

・DX基盤の整備状況が、LLM導入負荷と相関関係があり、本格的にLLM導入活動を行う前に、設計情報の体系化するなど、"急がば回れ"的な地道な準備が功を奏した。（図４参照）
・AIの技術進歩も見据え、進歩の内容に応じてソフトウェア開発の自動化、DX、AIといった改善アイテムの適切な選定することによって改善サイクルが回っていく素地が整った。
・ソフトウェア開発のプロセス改善において、ソフトウェア開発者側の道具を使いこなすスキルの比重が高まっていくため、本質的な開発プロセスを見失わずに、柔軟なモノの考え方の習得していく土壌が形成できた。
・経験、スキル差、練度が低い領域において工数削減の効果を確認

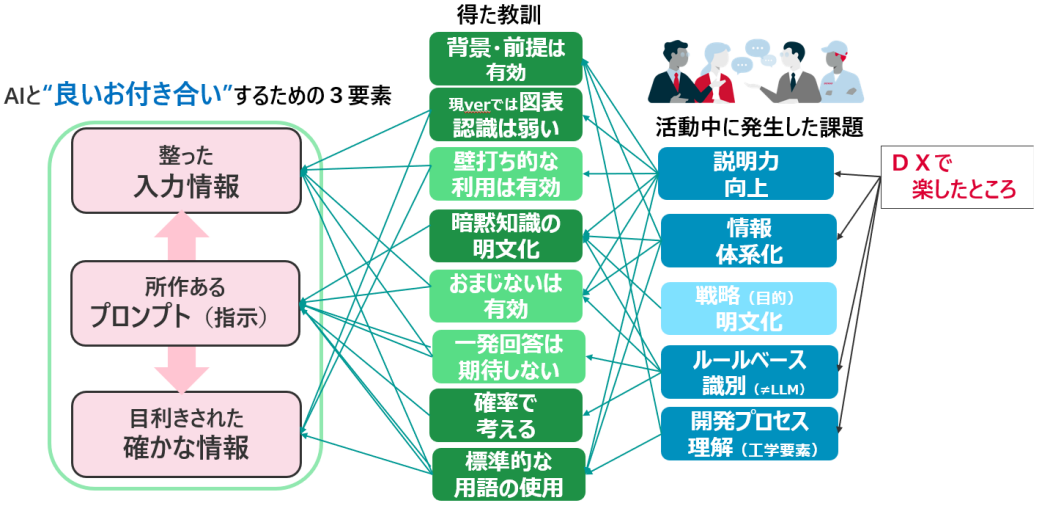


図４　LLM導入時のDX基盤との関係

参考情報

[1] 生成 AI 導入の教科書 小澤健佑 著
[2] 良い FAQ の書き方 樋口恵一郎 著
[3] ChatGPT などの初心者用ガイド https://prompt.quel.jp/basic.php
[4] SPI JAPAN 2023 日立研究所 Invitedpresentation2023 小川英人
[5] SPI JAPAN 2023 ソフトウェア開発における DX 活動の道しるべ 寺村幹夫
[6] 2019Sqip 研究会 研究５報告 組み込みシステムの設計者向け要件定義ガイドラインの提案と評価
[7] 2024Sqip 研究会 研究５報告 ＲＡＧによる既知障害情報の活用方法に関する考察
[8] Automotive SPICE Process Assessment Model V4.

2A3　ソフトウェアの価値を高頻度に創造していくためのスクラムチームにおける AI 活用事例　～週 3 ～ 4 回の商用リリースという高速開発の舞台裏～　小堀一雄（ジャイアント・アジャイル株式会社）

＜タイトル＞

ソフトウェアの価値を高頻度に創造していくためのスクラムチームにおける AI 活用事例

＜サブタイトル＞

～週3～4回の商用リリースという高速開発の舞台裏～

＜発表者＞

氏名(ふりがな):小堀一雄(こぼり　かずお)
所属:　ジャイアント・アジャイル株式会社

＜共同執筆者＞

氏名(ふりがな):　西　慎一郎(にし　しんいちろう)
所属:　レバレジーズ株式会社

＜主張したい点＞

ソフトウェア開発の高頻度なリリースの実現を目指して、従来のスクラムチームの開発に様々な AI をつないで活用したところ、要件調整の時間が 60%削減、スプリントレビュー時間が 70%削減できた。この効果のポイントとして、人間はスクラムチームの仕様検討やリリースの判断など集合知を活用した仕事に集中し、AI には単純作業を任せるというようにそれぞれの役割を明確化して、人と AI をつなぐことが重要である。

＜キーワード＞

AI，生成 AI，スクラム，アジャイル，生産性向上，ソフトウェア開発，高頻度リリース

＜想定する聴衆＞

ソフトウェアの生産性向上、リリース高速化を実現したいと考えるスクラムチーム、プロジェクトマネージャ、エンジニア

＜活動時期＞

2024 年 4 月～2024 年 10 月

＜活動状況＞:発表内容に複数の事例が含まれる場合は複数選択可能です。

☐ 着想の段階(アイデア・構想の発表)
☐ 改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
☐ その他(　　)

＜発表内容＞

1.背景

現在、スクラムチームの生産性改善を支援するサービスである Agile Effect を開発・提供している。このサービスは、スクラムチームのバックログの状況を分析し、業務プロセスを可視化・効率化することでアジャイルに生産性改善ができる特徴を持っている[1]。市場では類似の可視化ツールが競合として複数存在する環境であったため、高頻度にユーザーが求める価値を提供し続けることで市場競争力を強化し続ける必要があった。

2.改善したいこと

背景で説明したような市場で競争力を持つまでの高頻度リリースとはどの程度の頻度が必要かという点を考えた際、一般的なスクラムの考え方では1～4週間でスプリントを回すことがスクラムガイド[2]に定義されている。そこで、このプロダクトでは 1 週間未満のサイクルでの価値創造、つまり週3～4回といったリリース頻度を実現できるような改善を目指した。

週3～4回といった高頻度なリリースによる価値提供を実現する上で大きな課題となっていたのは、リリース前の確認に時間がかかっていることであった。リリース前の確認はプロダクト品質およびプロセス品質の両方をチェックする必要があり、それはひいてはリリースまでの開発プロセス及びスクラムイベント全ての状況の積み重ねを確認することであるため、すべての開発プロセス・スクラムイベントにおいて改善を実施して生産性を向上したいと考えた。しかも、通常の働き方の開発にとどまらず、近年生産性を著しく向上させると期待されている生成 AI を含む各種ツールを可能な限り活用する改善に挑戦した。

3.改善策を導き出した経緯

まず、現在利用可能な AI ツールの特性を分析し、どの開発プロセスやスクラムイベントに適用できるかを検討した。特に注目したのは、ChatGPT Pro や Cursor などの生成 AI が要件整理やコード生成に優れていること、Notta や Miro AI が議事録作成や情報整理に有効であること、Agile Effect がスクラムの可視化に特化していることであった。

次に、各開発プロセス（要件定義、設計、開発・単体テスト、結合テスト）とスクラムイベント（スプリントプランニング、デイリースクラム、バックログリファインメント、スプリントレビュー、レトロスペクティブ）において、最も時間がかかっている作業を特定した。その結果、要件整理の手作業、モブワークでの開発速度低下、テスト結果の文書化、各イベントでの進捗確認や議事録作成などが主要なボトルネックであることが判明した。

これらの分析を基に、人間が集合知を活用すべき創造的な作業（仕様検討、リリース判断など）と、AI に任せることができる定型的な作業（文書作成、情報整理、進捗確認など）を明確に分離し、それぞれに最適なツールを組み合わせる改善策を設計した。

4.改善策の内容

この課題に対して、PBI（プロダクトバックログアイテム）ごとにスクラムによるアジャイルソフトウェア開発を行う中で、表1及び表 2 に示すように AI 機能を含む各種ツールを活用し、生産性向上の効果を狙う改善策を実施した。

改善策は、開発プロセスの各段階（要件定義、設計、開発・単体テスト、結合テスト）とスクラムイベント（スプリントプランニング、デイリースクラム、バックログリファインメント、スプリントレビュー、レトロスペクティブ）の両方に AI ツールを組み込むことで、包括的な生産性向上を図るものである。
特に重要な改善として、スプリントレビューをフェーズゲートとして活用し、PO が直接承認するフローを確立した。これにより、リリース判断基準を明確化し、リードタイムの短縮を実現している。また、各プロセスにおいて人間と AI の役割を明確に分離し、人間は集合知を活用した仕様検討やリリース判断に集中し、AI には単純作業を任せるという基本方針を徹底した。

表 6 PBI 単位の開発におけるプロセス毎の生産性向上に対する改善策

開発プロセス	**利用したツール**	**生産性を向上させるための想定効果**
要件定義	ChatGPT Pro+Github Figma	• 従来の手作業による要件整理プロセスを、AI を活用した自動抽出・構造化により大幅に効率化する • 要求定義・要件定義の質が大幅に向上し、後工程での仕様変更や手戻りを減らす • 視覚的な情報（Figma）から一貫性のある要件文書を自動生成し、解釈の相違を防止する • 要件からタスクへの変換プロセスを自動化し、開発チームがすぐに作業開始できる状態を実現
設計	Cursor + Miro	• コードレビューの時間、リードタイムを削減する • 仕様書の作成にかかる時間を減らす • 設計品質を保ちながら、開発速度も向上させる • モブワークと組み合わせることで、集合知やレビュー工数削減を狙いつつ、モブの弱点である開発スピードの向上を実現させる
開発・単体テスト	Cursor	• モブプログラミング × AI によって、集合知の形成・開発スピードの向上と、単純作業に近いコーディングの実装時間を削減する • 複雑な実装においても、AI を活用したリサーチャーがいることで効率的に進める • プログラムコードができたタイミングで仕様書も自動的に生成ができるため、仕様書の更新業務を不要にする • AI によるコード品質チェックにより、ソースコードの保守性を高める

結合テスト	Notta	• 関係者全員がテスト状況を簡単に把握できるようになり誰でも統一された基準で品質を確認できるようにする • PO（プロダクトオーナー）が参加できなかった場合は、Notta が出力した概要を読み、録画を倍速で見て怪しいところだけチェックすることで、リリースまでのリードタイムを短くする • E2E テストの削減により、テスト実行時間を大幅短縮しつつ品質は向上させる • テスト結果の自動文書化により、品質改善のための知見が組織資産として蓄積する

表 7 スクラムイベント毎の生産性向上に対する改善策

イベント	**利用したツール**	**生産性を向上させるための想定効果**
スプリントプランニング	Agile Effect, Github, Miro, Cursor	• 確実なスプリントゴール達成のためのキャパシティープランニングのために、ツールを活用する • スプリントプランニング 2 部でモブワーク＋AI 活用で詳細仕様までのある程度実施することで、対応方針や、明確な受け入れ条件を全員が認識する
デイリースクラム	Agile Effect, Google Form, Google Spread Sheets, GAS	• デイリースクラムを議論の場として扱う • 進捗の確認は、Agile Effect ですぐに完了させる • 開始 5 分前に、チェックイン代わりに”セレブレーションセレモニー”を設け、チームビルディングを実施する
バックログリファインメント	Github, Chat GPT, Google Spread Sheets	• 毎週必ず、全ステークホルダーで実施する • タスクを切る際は、バックログとテンプレートをそれぞれ3つに分けて使い分ける • PO はできる限り発言を自粛する • 優先度が決まらない場合は、スプレッドシートで作成した優先度計算アルゴリズムを用いて、優先度を確定させる
スプリントレビュー	Google Meet, Notta , Miro	• Google Meet で録画しながら、極力オフラインにて全ステークホルダーで実施する • Notta が Google Meet に自動で入室するようにし、論点整理や NA をリアルタイムに文字起こし＆要約してもらうことで、全員がスプリントレビューに集中する

		• 条件分岐が複雑なところや探索的テストに関しては、Miro 上にやったことを付箋＋動画で貼り付け、スプリントレビュー前に全員が倍速で効率的に内容を確認する • スプリントレビュー内で PO が承認まで完了させることでリリース可能なインクリメントにする • フェーズゲートとして本セレモニーを活用することになるので、開発者はしっかりと準備をした上で臨むようにする
レトロスペクティブ	Agile Effect, Miro＋Miro AI	• Agile Effect で可視化されたスプリントの実績データをもとに定量的な振り返りを実施する • Miro でさまざまなテンプレートを用いて振り返りを実施する • 議論した内容は、Miro AI で最終的に情報を整理する

5.改善策の実現方法

先述した改善策を実現する具体的な方法と工夫点を表 3 及び表 4 に示す。

まず、開発プロセスにおいては具体的なツールの特性を活かした工夫を行っている点が挙げられる。要件定義では、GPT で Figma の画像を読み込ませる際のプロンプト工夫により、タスク管理ツールへの直接反映を可能にした。設計・開発では、通常のモブワークに「リサーチャー」役割を追加し、AI を活用した調査やテストコード作成を並行実行することで、ナビゲーター・ドライバーが他の作業を継続できる体制を構築した。結合テストでは、Notta を活用したリアルタイム動画解析により、議事録とネクストアクションの自動テキスト化を実現している。

次に、スクラムイベントにおいては、各イベントの目的に応じたツール活用により、本質的な議論時間の確保を図った。特にスプリントレビューでは、事前の動画共有とリアルタイム文字起こしを組み合わせることで、効率的な内容確認と集中的な議論を両立させている。

表 8　PBI 単位の開発におけるプロセス毎の実現の工夫点

開発プロセス	**実現方法や工夫点**
要件定義	• GPT で Figma の画像を読み込ませる際にプロンプトを工夫することで、そのままタスク管理ツール側にも反映可能にする • 開発に着手できるようにする • （登壇発表ではテンプレートの公開とアウトプット、運用方法を共有する予定）
設計	• 実装だけではなく設計からモブワークのスタイルで進める

	• 基本的には通常のモブワークと同様に進めるが、AI 支援を行う役割「リサーチャー」を設け、ナビゲーター・ドライバーをサポートする • 例えば設計を進める中で必要な調査などが出てきた場合は、リサーチャーが AI を活用した調査・テストコードの作成を別ファイルに記載し支援する。また AI により現時点での仕様が要件を満たしているかもリアルタイムでチェックする • その間、ナビゲーター・ドライバーは他で進められる作業を並行して進めることが可能にする • 集合知を高めるだけではなく、眺めているだけのメンバーが生まれないよう、全員が協力し合う効果的なモブを行う
開発・単体テスト	• 設計プロセスと同様、全てのタスクを Cursor + モブ+AI を組み合わせて行う • Cursor は既存のコードからのコーディングに長けているので、既存のコードが参考に出来る場合では基本 Cursor にコーディングしてもらう • 単体テストのコーディングやフロントエンドの UI 実装などは特に有効なので上記プロセスを活用する
結合テスト	• AI を活用して、コンポーネントテストやリグレッションテストを効率的に充実させることにより、E2E テストを大幅に減らす • 簡単な E2E テストは GPT のオペレーターを活用してチェックする • スプリントレビューで E2E テストを実施する • Notta でリアルタイムに動画解析して議事録やネクストアクションを自動でテキストに起こす

表 9 スクラムイベント毎の実現の工夫点

イベント	実現方法や工夫点
スプリントプランニング	• 確実なスプリントゴール達成のためのキャパシティープランニングのために、ツールを活用する • スプリントプランニング 2 部でモブワーク＋AI 活用で詳細仕様までのある程度実施することで、対応方針や、明確な受け入れ条件を全員が認識する
デイリースクラム	• デイリースクラムを議論の場として扱う • 進捗の確認は、Agile Effect ですぐに完了させる • 開始 5 分前に、チェックイン代わりに”セレブレーションセレモニー”を設け、チームビルディングを実施する

バックログリファインメント	• 毎週必ず、全ステークホルダーで実施する • タスクを切る際は、バックログとテンプレートをそれぞれ3つに分けて使い分ける • PO はできる限り発言を自粛する • 優先度が決まらない場合は、スプレッドシートで作成した優先度計算アルゴリズムを用いて、優先度を確定させる
スプリントレビュー	• Google Meet で録画しながら、極力オフラインにて全ステークホルダーで実施する • Notta が Google Meet に自動で入室するようにし、論点整理や NA をリアルタイムに文字起こし＆要約してもらうことで、全員がスプリントレビューに集中する • 条件分岐が複雑なところや探索的テストに関しては、Miro 上にやったことを付箋＋動画で貼り付け、スプリントレビュー前に全員が倍速で効率的に内容を確認する • スプリントレビュー内で PO が承認まで完了させることでリリース可能なインクリメントにする • フェーズゲートとして本セレモニーを活用することになるので、開発者はしっかりと準備をした上で臨むようにする
レトロスペクティブ	• Agile Effect で可視化されたスプリントの実績データをもとに定量的な振り返りを実施する • Miro でさまざまなテンプレートを用いて振り返りを実施する • 議論した内容は、Miro AI で最終的に情報を整理する

6.改善による変化や効果

先述した改善による開発プロセス毎の変化や効果を表 5 に、スクラムイベント毎の変化や効果を表 6 にそれぞれ示す。

まず、開発プロセスにおいては、要求定義・要件定義工程で約 60%の削減、手戻りがほとんどなくなるという大幅な改善を実現した。設計プロセスでは、モブワークによる集合知やレビュー工数削減の利点を享受しつつ、開発スピードも維持・向上させることができた。開発・単体テストでは、AI による単純作業の自動化により実装時間が削減され、同時に仕様書の自動生成により更新業務が不要になった。

次に、スクラムイベントでは、各イベントの効率化と質的向上を両立している。デイリースクラムでは進捗確認時間を 2 分以内に短縮し、小さな障壁に関するコミュニケーション遅れも解消された。スプリントレビューでは 70%以上の時間削減を実現し、本質的な議論と称賛の場へと変化した。レトロスペクティブでは、データドリブンなプロセス改善が可能になり、雑談の場になる問題も解消された。これらの改善により、リリース判断基準の明確化とリードタイム短縮が実現し、高頻度リリースの基盤が構築された。

表 10 PBI 単位の開発におけるプロセス毎の改善による変化や効果

開発プロセス	改善による変化や効果
要件定義	• 要求定義、要件定義の工程が約 60%削減された • 要求定義、要件定義の質が大幅に上がり、手戻りがほとんどなくなった
設計	• モブを行うことによるメリット(集合知やレビュー工数削減など)を享受しつつ、モブの弱点である、開発スピードも維持/向上させることができた • コードレビューの時間、リードタイムは大幅に削減された • 仕様書の作成にかかる時間が殆どなくなった
開発・単体テスト	• モブ×AI によって、集合知の形成・開発スピードの向上と、単純作業に近いコーディングの実装時間が削減した • 複雑な実装においても、AI を活用したリサーチャーがいることで効率的に進めることができた • 実装物ができたタイミングで仕様書も自動的に生成ができるため、仕様書の更新業務は不要になった
結合テスト	• 関係者全員がテスト状況を簡単に把握できるようになり誰でも統一された基準で品質を確認できるようになった • PO が参加できなかった場合は、Notta が出力した概要を読み、録画を倍速で見て怪しいところだけチェックすることで、リリースまでのリードタイムが大幅に短くなった

表 11 スクラムイベント毎の改善による変化や効果

イベント	改善による変化や効果
スプリントプランニング	• 手戻りがほぼゼロになった • 集合知の獲得として、属人化を防ぐことができているため、誰がどのタスクでもやれる状態でフロー効率が向上した • 見積もり精度が向上し、ベロシティーも大幅に安定した
デイリースクラム	• 進捗確認に使う時間は 2 分以内になった

	• Agile Effect を見ることで、議論のきっかけが生まれるため、日々進行プロセスの改善が迅速に進んだ • 小さな障壁に関するコミュニケーション遅れが発生しなくなった
バックログリファインメント	• 優先度の決定に迷わなくなった • 全員が次やるタスクが明確で、タスクの説明もテンプレ化されているため、内容理解も早くなった
スプリントレビュー	• リリースの判断基準がスプリントレビューの承認だけになるので、リードタイムが短くなった • 改善点や意見に関しては、自動で AI が議事録を取ってくれるため、ヒアリングに集中できるようになった • スプリントレビューの時間が 70%以上削減され、本質的な議論が実施され、称賛の場になった
レトロスペクティブ	• 振り返りの内容が出てこない、雑談の場になってしまうなどの問題は発生しなくなった • データ・ドリブンなプロセス改善が進められるようになった

7.改善活動の妥当性確認

Agile Effect というスクラム開発の可視化・業務効率化ツールを活用し、例えば以下のようなデータドリブンなプロセス改善の妥当性確認を実施している。

- スプリントの実績データを可視化し、定量的な振り返りを実施
- デイリースクラムにて進捗を迅速に確認し、議論の時間を確保
- スプリントプランニングにてキャパシティプランニングに活用し、見積もり精度を向上

これらのデータ分析により、見積もり精度の向上、ベロシティーの安定化、デイリースクラムでの進捗確認の時間を 2 分以内に抑制、などの効果が上がっていることを定量的に確認した。また、妥当性の定性的な確認として、レトロスペクティブにおいて Agile Effect が示すデータを基にプロセス改善を実施することで、「振り返りのネタが出てこずに雑談の場になってしまう」という問題が解消した。

参考情報

[1] レバレジーズ株式会社, Agile Effect の特徴 (2024), https://lp.agile-effect.com/#facility
[2] Ken Schwaber & Jeff Sutherland, The Scrum Guide (2020), https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

2A4　生成 AI を活用したレビュー効率の向上と有効性の検証　西本真由（株式会社日立製作所）

＜タイトル＞
生成 AI を活用したレビュー効率の向上と有効性の検証
＜サブタイトル＞

＜発表者＞
氏名（ふりがな）：西本　真由　（にしもと　まゆ）
所属：　株式会社　日立製作所

＜共同執筆者＞
氏名（ふりがな）：古田　莉央（ふるた　りお）
所属：株式会社　日立製作所

＜主張したい点＞
ソフトウェア開発の品質向上を目的とした取り組みの一つとして、レビューチェックリストの適用がある。そのレビューチェックリストを生成 AI で作成し、実プロジェクトへ適用することでレビューチェックリストの有効性を検証した。
本手法を用いることでレビューチェックリストの作成および、メンテナンス負荷を軽減することができた。
さらに、生成 AI による開発成果物レビューに関する検証結果を報告する。

＜キーワード＞
生成 AI、ソフトウェア品質、アジャイル開発、バグ報告、レビュー、レビューチェックリスト

＜想定する聴衆＞
ソフトウェア開発者、プロジェクトマネージャー、品質保証担当者

＜活動時期＞
2024 年 8 月～（継続中）

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。
□ 着想の段階（アイデア・構想の発表）
■　改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
□ その他（　　）

＜発表内容＞
1.背景
ソフトウェア品質を向上させるための取り組みの一つとして、レビューチェックリストを用いたレビューの実施が挙げられる。その有効性については加藤らや野中が提示している [1, 2]。アジャイル開発においてもウォーターフォール開発と同様、同じバグを繰り返さないための取り組みの必要性は認識されている。しかしながら、レビューチェックリストの定期的な見直しが省略されることが多く、タイムリーかつ効率的なフィードバックが求められる。それは、成果を短期間で繰り返しリリースするという開発特性上、アジャイル開発ではスピードが重視されるため、レビューチェックリストのメンテナンスにかかる時間が課題となり、最新状態のレビューチェックリストを維持することが難しい。また、レビュー自体にも時間や工数に関する問題に起因して、本来レビューで検出すべき不良を見逃すという問題がある。さらに、各種レビューでのチェックリストの作成およびレビューは担当者のスキルに依存するという問題がある。

2.改善したいこと
下記 2 点の課題を解決し、生成 AI を用いて先行研究の手法をより効率的に適用することで、レビューおよびレビューチェックリスト作成を支援する。
課題 1：バグ分析およびレビューチェックリストへのフィードバックは時間がかかる傾向にあり、メンテナンスが追い付いていないこと。
課題 2：レビューチェックリスト作成時には暗黙知やプロジェクト独自の観点が含まれ、レビューチェックリストの質に対する課題（スキル依存）があること。
課題３：レビューすべき成果物とレビューチェックリストの項目数が大量にあり、すべてを網羅することが困難である。レビューチェックリストの項目内容が設計書のどの部分に記載されているか、具体的な記載箇所の特定や妥当性評価に工数が膨大にかかってしまう。

3.改善策を導き出した経緯
昨今、生成 AI を利用した業務効率化が多くの企業で取り組まれており、日立製作所内でも同様の取り組みが進められている。今回はレビューというプロセスに着目して、業務効率化に取り組んだ。生成 AI の使用方法については、ユースケースごとにガイドラインが作成されている。ユースケースの例としては、文書要約や計画書作成補助などがあるがその中の一つとしてレビュー観点を作成する手法が紹介されている。ユースケースの全体像を図 1 に示す。検査観点やレビューチェックリストなど、成果物を評価するための補助資料を生成 AI で作成するアプローチとしては、2 種類ある。機能概要の一覧および、社内のナレッジ（標準レビュー観点表）からレビュー観点を生成するパターン（図 1 の手法 1）と、過去のバグ一覧をインプットにレビュー観点が記載されたレビューチェックリストを作成するパターン（図 1 の手法 2）である。本報告では、後者の手法を検証する。

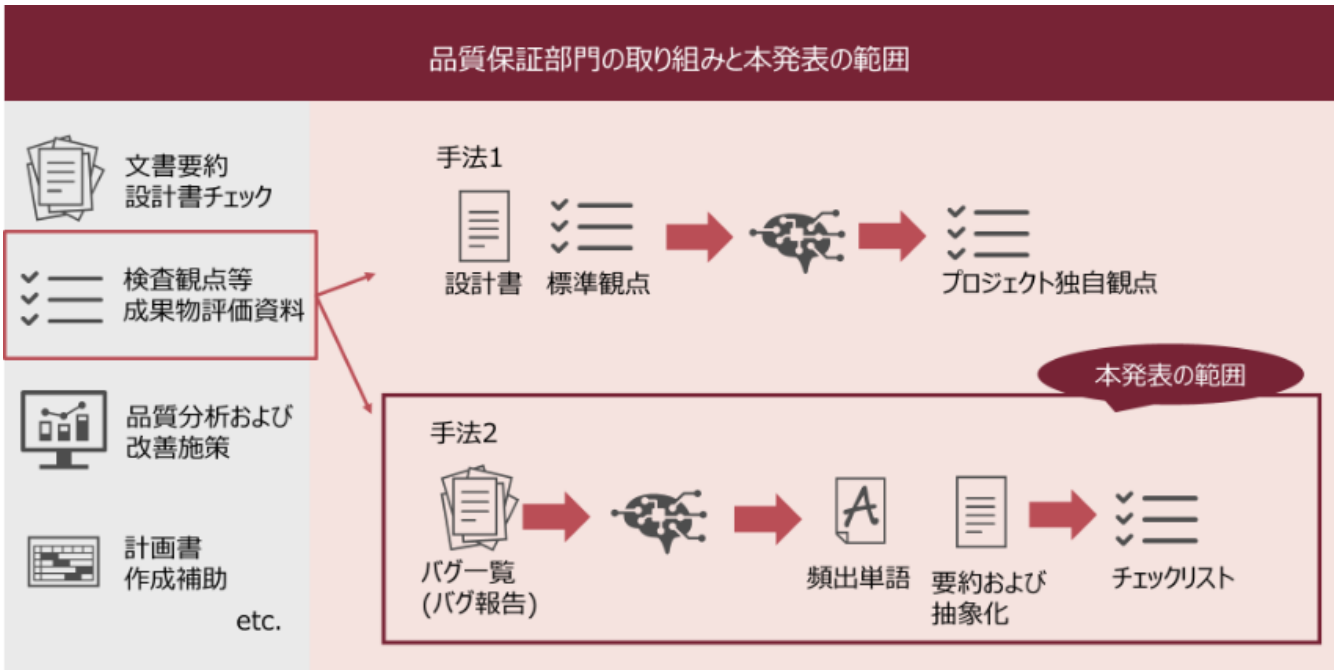


図 6　品質保証部門の取り組みと本論文の範囲

4. 改善策の内容

自然言語処理を用いて過去のバグを要約および抽象化し、その結果をもとにレビューチェックリストを作成する取り組みが研究されている [3]。生成 AI を用いて先行研究の手法を自動化することで、レビューチェックリスト作成を支援する。生成 AI による支援を行うことで、作業効率化（課題 1 の解決）および、レビューチェックリスト検討および作成時における暗黙知などによるレビューチェックリストの質問題（課題 2 の解決）の効果を検証する。さらに、追加の取り組みとして生成 AI にレビューそのもの（担当者によるセルフチェック、以下セルフチェックと呼称）を実施させ、その効果を検証する（課題 3 の解決）。

5. 改善策の実現方法
改善策の全体を下記図に示す。詳細は後述 5.1 節、5.2 節に記載する。

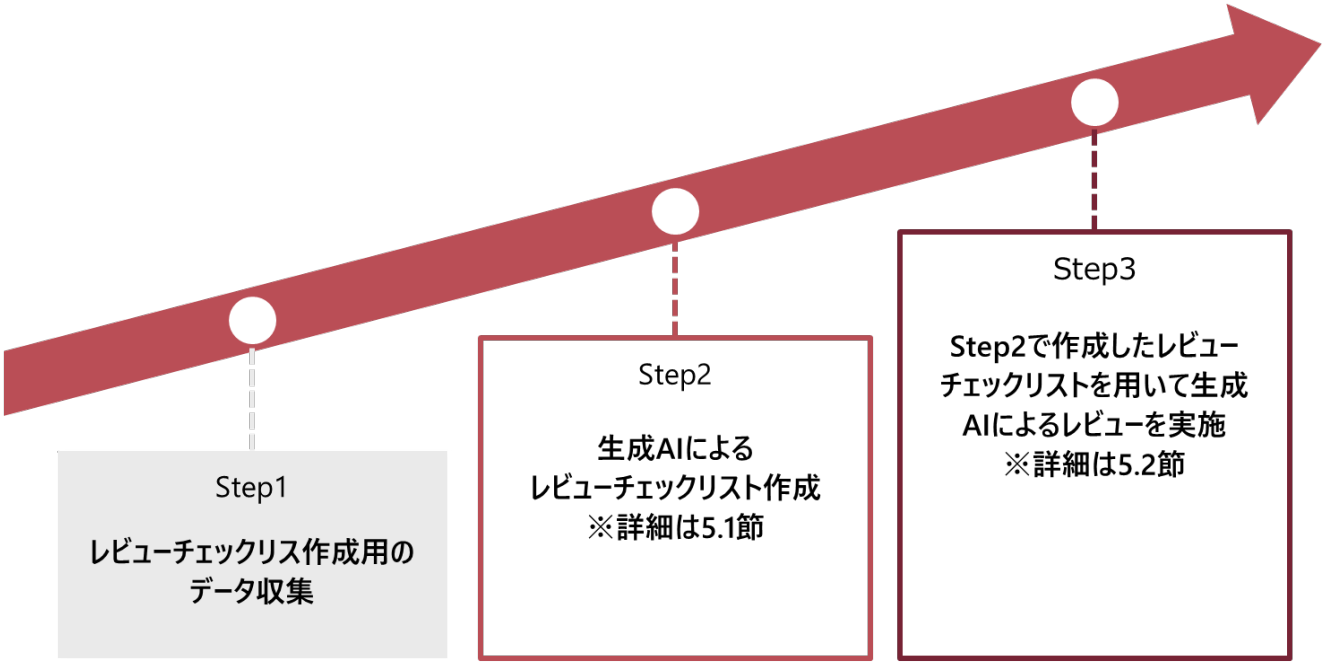


図 7　解決策の全体像

5.1. 生成 AI によるレビューチェックリスト作成

5.1.1. 先行研究と本論文での手法・工夫点

渡邊ら [3]は効率的なレビューチェックリスト作成方法として、過去のバグ報告書に含まれる単語に注目し、単語の出現頻度からバグの抽象化を実施した。バグを抽象化したものをチェックリストとして、同一事象を未然に防ぐ研究を実施している。今回の取り組みでは渡邊らの手法を参考にレビューチェックリスト作成の自動化を試みた。先行研究と手順が異なる部分がいくつか存在するため、以下に詳細を記載する。

a) 表 1 先行研究の手順の項番 3 にて上位の単語を抜き出しているが、本論文ではすべての単語を使用する。

b) バグのグルーピングは未実施である。

上記の先行研究からの変更は少数しかないが重要なバグを見逃す恐れがあること、また、アジャイル開発で規模が大きくない案件のため、生成 AI のインプットとなる指摘件数も少ないため最終的なアウトプットとして出力されたものを手作業で確認しても大きな工数はかからないと考えたためである。ただし、指摘が膨大な場合は先行研究の手順のとおり上位の単語を選ぶ方が効率的と考えられる。また、本取り組みでは先行研究の手順を 3 分割し、それぞれプロンプトを作成した。詳細を以下の表に示す。

表 12　プロンプトの 3 分割に関する詳細

#	先行研究の手順	本論文での手順
1	すべてのバグ報告から単語を抽出する。	プロ…
2	単語の出現頻度を計算し、出現頻度が高い順に並び替える。出現頻度別の単語リストを作成する。	【変更点1】上位の単語ではなく、すべての単語を抜き出し
3	作成した単語リストの上位にある単語を含むバグ報告をすべてのバグ報告から抽出する。	プロンプトb)単語リストにある単語を含むバグをバグ一覧から抽出する。
4	バグを要約する。	【変更点2】グルーピングはしない
5	バグを抽象化する。	プロンプトc)バグを抽象化する。
6	抽象化したバグをグルーピングする。	削除。

プロンプトエンジニアリングに関しては以下の工夫を行った。

ア）Markdown 形式で記載する

イ）前提条件や制約条件を記載する

ウ）解析手順を詳細に記載する

エ）手順に応じた役割を与える

また、バグ一覧を Markdown 形式に変換する際、指摘件数が多くなると手動変換が困難なため、エクセルマクロで自動化を行うことで、データ整形にかかる時間を削減した。

5.1.2. 適用プロジェクトの概要

手法を試行したプロジェクトはアジャイル開発の A プロジェクトである。

(1) 案件特徴

A プロジェクトの案件特徴は、以下に示す。

ア）スプリント期間は 2 週間である。

イ）A プロジェクトでは半年ごとに既存機能の改修・新規機能実装を実施している。

ウ）今回作成したレビューチェックリストを適用する開発（以下、今回開発と呼称）において開発期間は半年である。

エ）今回開発の開発規模は 14.5kS である。

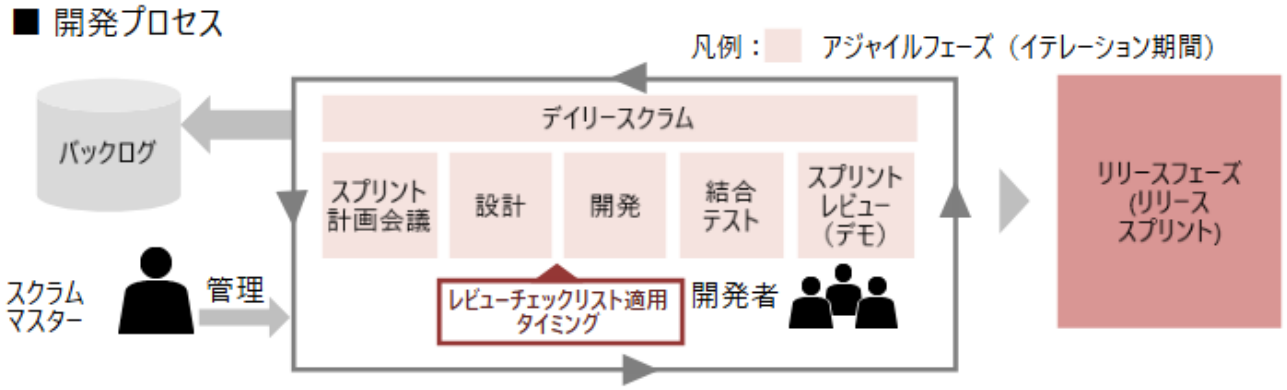


図 8　開発プロセス

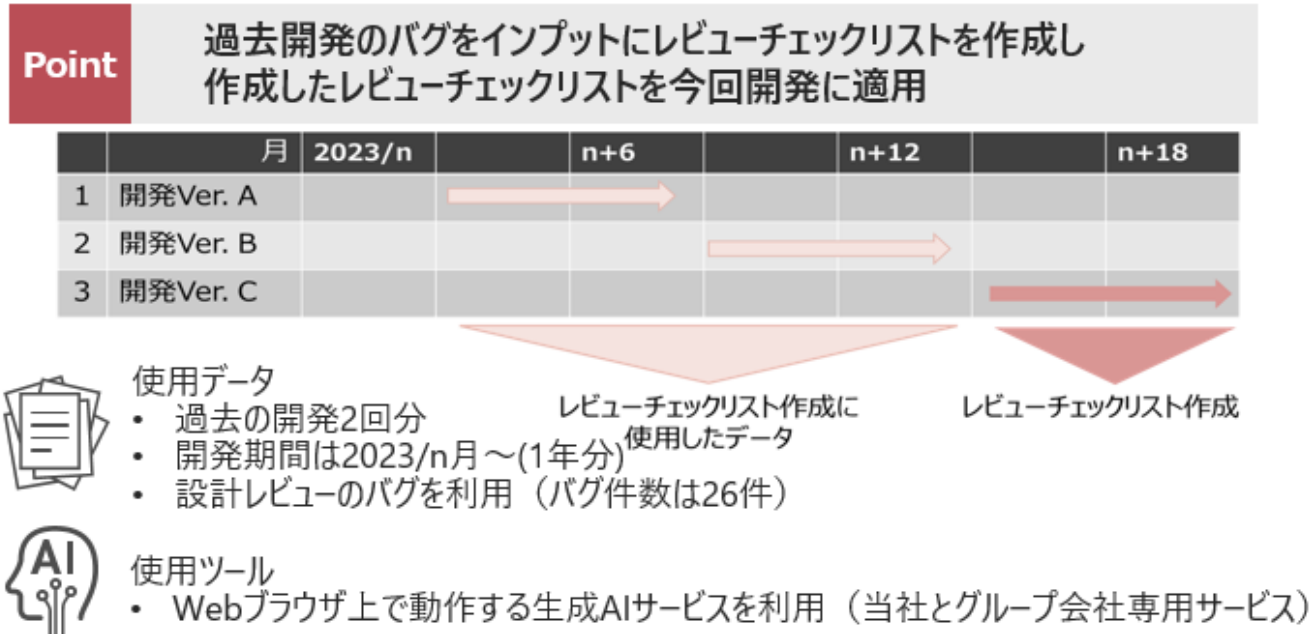


図 9　開発期間全体像と使用データ

(2) 使用データ
今回使用したデータは過去の開発２回分（開発期間は 2 回の開発を合計して 1 年分）で、設計レビューのバグを利用した。インプットとしたバグ件数は 26 件分である。使用ツール（前提条件）は Web ブラウザー上で動作する生成 AI サービス（当社およびグループ会社専用サービス）を利用した。生成 AI のバージョンは、チャットでの応答回数制限があるが新しいバージョンとチャットでの応答回数制限がない古いバージョンの 2 種類が利用でき、本論文では処理が日本国内で完了する古いバージョンを主に使用した。

5.2. 生成 AI によるセルフチェックの実施

5.2.1. 手法
生成 AI によって作成したレビューチェックリストのチェック項目文をプロンプトに記載し、生成 AI にプロンプトの内容が設計書に含まれているかを確認させる。

5.2.2. 適用プロジェクトの概要
(1)案件特徴
適用案件は 5.1.2(1)と同様のプロジェクトとする。

(2)使用データ
レビュー対象物は設計書とする。
生成 AI へのインプットには 5.1 節にて作成したレビューチェックリストと社内の汎用ナレッジを用いる。

6.改善による変化や効果

6.1. 生成 AI によるレビューチェックリスト作成

6.1.1 効果検証内容
本報告では以下の検証を実施する。

a) 作業時間削減の効果に対する検証
b) レビューチェックリストの作成精度に対する検証

作業時間削減の効果に対する評価については、手作業で実施する際と比べて生成 AI を用いる場合、約半分の時間で作業が完了すると仮定した。レビューチェックリスト作成時のレビューチェックリストの質（スキル依存）に関する課題についての評価については、2 つの観点で評価する。1 つ目は、手作業で作成した場合を基準に、生成 AI に作成させた場合のレビューチェックリストの再現度 を評価する。2 つ目は、本手法で作成したレビューチェックリストを使用した場合での指摘件数およびバグ件数を計測し、手動で作成したレビューチェックリストを適用した際の指摘件数やバグ件数との比較を行う。指摘件数は、バグの疑いがありバグ一覧に起票された件数を示し、バグ件数は指摘件数のうちバグであると確定した件数を示す。前回開発の実績をもとに、今回開発での指摘件数、バグ件数についての仮説を立てた。以下の表に示す。開発中のプロジェクトに適用するため、前回開発に比べ、対象頁数が約 3 割となっている。そのため、指摘件数および、バグ件数も前回同様の品質であると仮定し、前回開発の 3 割を目標とする。

表 13　前回開発の実績をもとに本検証での指摘件数およびバグ件数の仮説

#	対象頁数	指摘件数	バグ件数
今回開発	27	5	1
前回開発※2	90	16	3

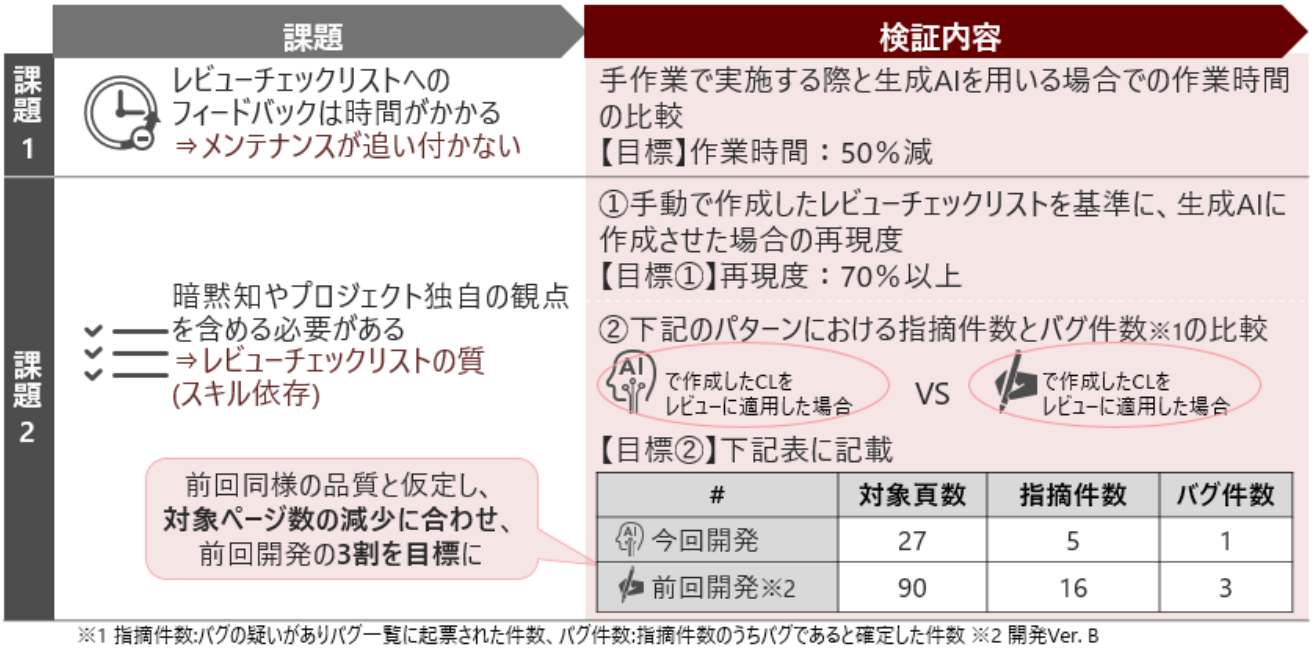


図 10　課題に対応する検証内容の詳細

6.1.2 検証結果

(1) 作業時間の削減効果に対する検証結果

作業時間の削減効果に関して、生成 AI を使用しない場合に比べて約 50%の作業時間を削減することができた。

結果の詳細は以下の表に示す。

表 14　レビューチェックリストの作成における所要時間

#	比較条件	レビューチェックリスト作成	評価
1	手作業(先行研究の手順を手作業)	2h35min	−
2	生成AI利用	1h00min	○

[凡例(評価欄)]　◎目標の倍以上、○目標どおり(作業時間半減)、×目標を下回る(仮説で定義した数値の半分以下)、−対象外

(2) レビューチェックリストの品質に関する検証結果

生成 AI に作成させたレビューチェックリストは手作業で作成する場合と比べても品質に大きな違いがないことが確認された。生成 AI を利用してレビューチェックリストを作成した場合の精度に関して、詳細を以下の表に示す。本取り組みでは再現率を、手動で作成した全レビューチェックリストに対して生成 AI が再現した割合として定義した。手動で作成したレビューチェックリスト 28 件のうち、再現できた件数は 25 件で、再現度としては 89%であった。また、インプットに扱ったバグ件数は 26 件で作成したレビューチェックリストは 21 件で件数が減少したが、これは要約および、抽象化する際に複数のバグを 1 つにまとめたためである。

表 15　生成 AI により作成したレビューチェックリストの精度

項目	再現率	評価
再現率（手動で作成したチェックリスト全観点に対してAIが再現した割合）	89%	○

[凡例(評価欄)]　◎目標を上回る(90%以上)、○目標(70%)どおり、×目標を下回る(50%以下)、−対象外

生成 AI によって作成したレビューチェックリストを用いて実際にレビューした結果を以下に示す。表 3 行目の実績は生成 AI を用いてレビューチェックリストを作成したのちにそのレビューチェックリストを用いて設計書のレビューを実施した結果で、指摘件数 4 件、バグ件数 1 件を摘出した。前回開発の実績値をもとに、仮説を立てたが、検証結果は指摘件数が 1 件少ないという結果が得られた。指摘やバグの件数は、前回開発の実績と比較しても割合においては前回開発相当の件数であるといえる。

表 16　生成 AI により作成されたレビューチェックリストを用いたレビュー結果

項目	対象頁数	指摘件数※1	バグ件数※1	バグ密度※2	評価
過去開発の実績	90	16	3	3.3	−
仮説	27	5	1	3.7	−
実績	27	4	1	3.7	○

※1指摘件数は、バグの疑いがありバグ一覧に起票された件数を示し、バグ件数は指摘件数のうちバグであると確定した件数を示す
※2 100頁あたりの不良件数
[凡例(評価欄)]　◎目標の倍以上、○目標どおり、×目標を下回る(仮説で定義した数値の半分以下)、−対象外

6.2. 生成 AI によるセルフチェックの実施

6.2.1 効果検証内容

本報告では以下の検証を実施する。

a) 作業時間削減の効果に対する検証
b) 生成 AI によるレビュー品質の検証

作業時間削減の効果に対する評価については、手作業で実施する際と比べて生成 AI を用いる場合、約半分の時間で作業が完了すると仮定した。
生成 AI によるレビュー品質の評価は、人の手でセルフチェックした際に摘出されている不良が実際に生成 AI でセルフチェックした場合に摘出できているかを検証する。正答率 70%以上を目標値とした。

6.2.2 検証結果

(1)作業時間の削減効果に対する検証結果
作業時間の削減効果に関して、生成 AI を使用しない場合に比べて約 50%の作業時間を削減することができた。
結果の詳細は以下の表に示す。

表 17　設計書セルフチェックにおける所要時間

#	比較条件	レビュー時間	評価
1	手作業	2h15min	−
2	生成AI利用	1h02min	○

[凡例(評価欄)]　◎目標の倍以上、○目標どおり(作業時間半減)、×目標を下回る(仮説で定義した数値の半分以下)、−対象外

(2)生成 AI によるレビュー品質の検証結果
生成 AI でのセルフチェックは手作業で実施する場合と比べても品質に大きな違いがないことが確認された。生成 AI を利用してセルフチェックを実施した場合の再現度に関して、詳細を以下の表に示す。

表 18　設計書セルフチェックにおける生成 AI の正答率

項目	対象CL件数	正答数	正答率	評価※
仮説	21	15	70%	−
実績	21	16	76%	○

※ [凡例(評価欄)]　◎目標の倍以上、○目標どおり、×目標を下回る(仮説で定義した数値の半分以下)、−対象外

本取り組みでは正答率を、全レビューチェックリスト(21 件)に対して生成 AI が正しく回答した割合として定義した。レビューチェックリスト 21 件のうち、正答数は 16 件で、正答率としては 76%であった。
本手法のメリットとして、以下が挙げられるが、正答率が目標値をわずかに上回るレベルのため効果が薄いことが問題である。
・体裁レベルの指摘のみならず処理漏れなど設計内容に関わる不良を摘出できた
・設計書に該当のチェックリスト項目の記載がある場合、記載箇所は生成 AI に出力させることで
セルフチェックの効率向上が期待できる
そこで、正答率が低い原因を以下であると仮定し、プロンプトの改善を試みた。

a) インプットするチェックリストの記載に曖昧表現がある場合、正答率が下がる
b) 設計書に記載されているか未記載かの 2 択方式でプロンプトを作成したため、要求仕様ではないことが理由による設計対象外がすべて記載漏れの不良として出力されてしまう

プロンプト改善後の結果の詳細を以下の表に示す。

表 19　プロンプト改善後における生成 AI の正答率

項目	対象CL件数	正答数	正答率	評価※
改良前プロンプト	21	16	76%	−
改良後プロンプト	21	19	90%	○

※[凡例(評価欄)]　◎目標の倍以上、○目標どおり、×目標を下回る(仮説で定義した数値の半分以下)、−対象外

プロンプト改善後の正答率は正答率が 90%と正答率の改善がみられた。

7. 改善活動の妥当性確認

7.1. 生成 AI によるレビューチェックリスト作成

6.1 章の検証結果より、本手法が有効であることが分かった。
インプットに扱ったバグ件数は 26 件、作成したレビューチェックリストは 21 件で、約 1 時間程度、割合にして 50%程度、作業時間の削減ができ、目標（作業時間の半減）を達成した。 また、生成 AI により作成したレビューチェックリストの再現度（手動で作成したチェックリスト全観点に対して AI が再現した割合）は、目標 70%に対して、結果 89%であり、生成 AI に作成させたレビューチェックリストは手作業で作成する場合と比べても品質に大きな違いがないことが検証結果から明らかになった。

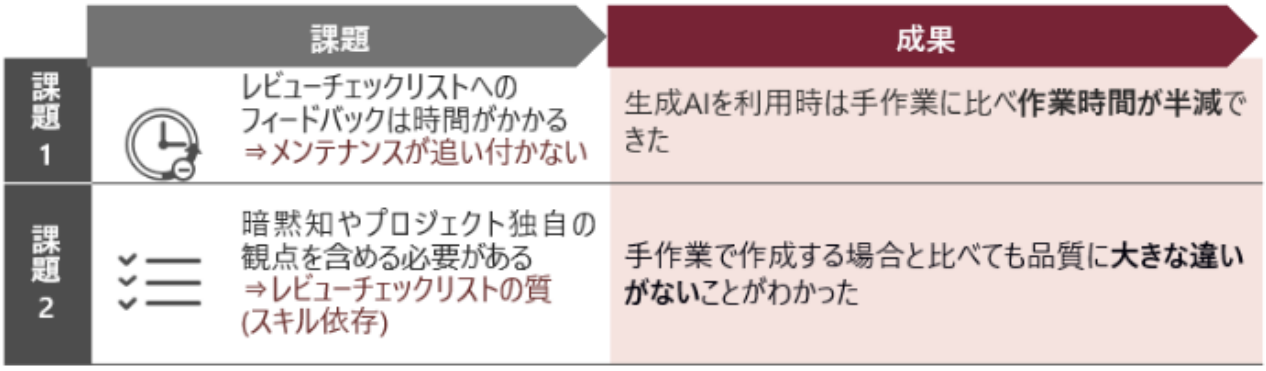


図 11　課題と対応する成果

さらに、他のプロジェクトに本取り組みを横展開し、異なるプロジェクトで適用タイミングが違う場合でも同等の効果を得られた。詳細を以下の図および表に示す。

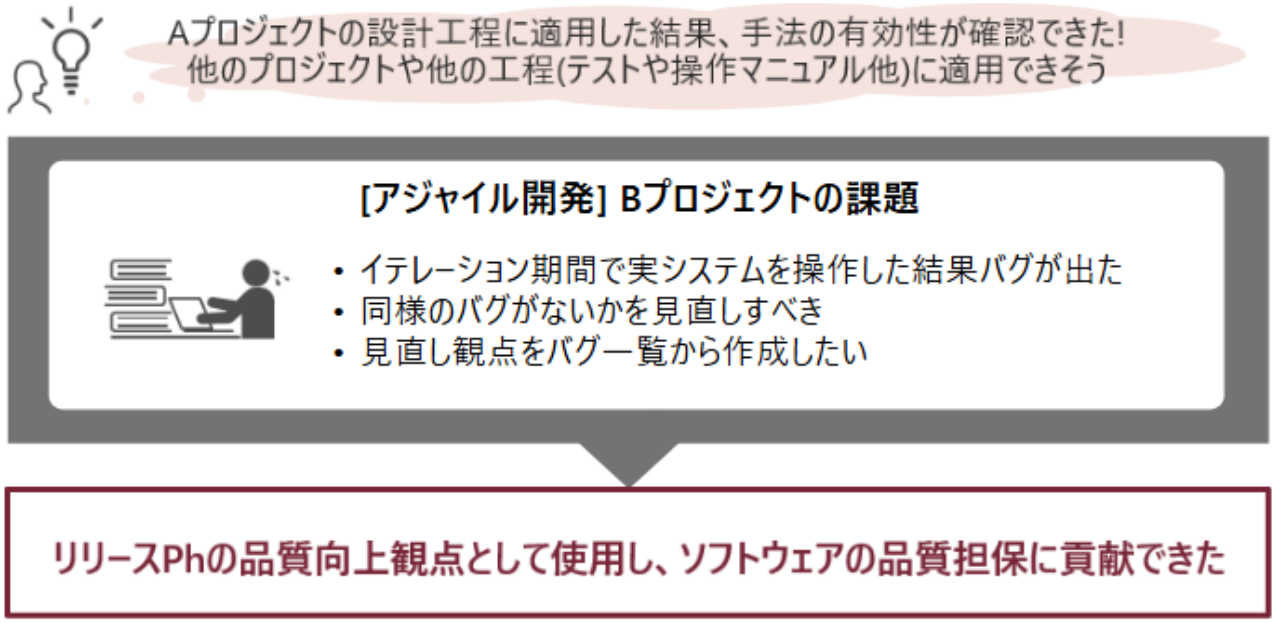


図 12　B プロジェクトでの本手法の適用

表 20　A プロジェクトと B プロジェクトでの適用条件の差異

項目	Aプロジェクト	Bプロジェクト
使用データ	過去開発2回分のバグ	アジャイルPhでのバグ
適用タイミング	今回開発(設計)	リリースPh
インプットデータ量(バグ件数)	26件	17件

7.2. 生成 AI によるセルフチェックの実施

6.2 章の検証結果より、本手法が有効であることが分かった。
手作業に比べ生成 AI を利用した場合約 1 時間、割合にして 50%程度、作業時間の削減ができ、目標（作業時間の半減）を達成した。 また、生成 AI による設計書のセルフチェックでは、目標 70%に対して、正答率 90%であり、摘出した不良の件数が、手作業と同等の水準であることがわかった。今後の課題としては、インプットデータが大きいプロジェクトやウォーターフォール開発への適用、有効性の検証の実施や、より広く活用するために AI エージェント化を検討する。

参考情報

[1] 加藤秀一, 堀江憲一, 小川克彦, 木村重良 (1995). HI 設計チェックリストとその有用性の評価. 情報処理
学会論文誌, 36(1), 61-69.
[2] 野中誠 (2004). 設計・ソースコードを対象とした個人レビュー手法の比較実験. 情報処理学会研究報告
ソフトウェア工学 (SE), 2004(118 (2004-SE-146)), 25-32.
[3] 渡邊正隆, 森崎修司, 松本健一 (2010). バグ報告の単語出現頻度に着目したチェックリスト作成の試行. 研究報告ソフトウェア工学 (SE), 2010(30), 1-7.

2B1 抽象プロセスの活用でアジャイル開発に適用しやすいＱＭＳを構築する ISO9001 認証を目指して 井芹久美子（三菱電機株式会社）

＜タイトル＞

抽象プロセスの活用でアジャイル開発に適用しやすいＱＭＳを構築する

＜サブタイトル＞

ISO9001 認証を目指して

＜発表者＞

氏名(ふりがな)：井芹　久美子（いせり　くみこ）
所属：　三菱電機株式会社

＜共同執筆者＞

氏名(ふりがな)：　細谷 泰夫（ほそたに　やすお）
所属：　三菱電機株式会社

氏名(ふりがな)：　湯川 純（ゆかわ　じゅん）
所属：　三菱電機株式会社

＜主張したい点＞

- 目的:アジャイル開発、特にスクラムを適用したソフトウェア開発の推進にあたり、品質保証上の懸念を払拭するため、ISO 9001 に適合した品質マネジメントシステム(QMS)を構築すること。特定の開発プロセスに強く依存せず、アジャイル開発の柔軟性を損なわない品質保証の枠組みが求められている。
- 手段: ISO 9001 の要求事項と、スクラムなどの具体的な開発プロセスとの間に、「抽象プロセス」という中間層を設ける QMS アーキテクチャを考案・構築した。この抽象プロセスにより QMS のうち安定した部分と、変化しうるソフトウェア開発プロセスで定義する部分とを分離できる。
- 成果:ソフトウェア開発プロセスの選択や追加に柔軟に対応できる、拡張性の高い QMS のモデルケースを提示できた。また、この QMS にて ISO 9001 認証を取得した。認証が取得できたことで、ISO 9001 と整合した QMS がアジャイル開発へ適用可能であることが実証でき、社内外のアジャイル導入に対する不安軽減に貢献できると考えている。

＜キーワード＞

アジャイル、QMS、ISO9001

＜想定する聴衆＞

- スピード感と品質を両立する開発に適用しやすい QMS 構築に関心がある方
- アジャイル開発（特にスクラム）を推進している方
- ISO9001 認証に関心がある方

＜活動時期＞

2023 年 4 月頃～現在

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
□改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞
1.背景
当社は、「イノベーティブカンパニー」への変革を目指している。その中核を担う組織として、DX イノベーションセンター(DIC)がある。デジタル基盤の整備、開発手法・品質保証に関わるルールの構築、社外パートナーとの共創等を担う組織である。顧客の要望を取り入れて柔軟に俊敏に進める必要があるデジタルサービスの開発を想定し、DIC ではアジャイル開発手法、特にスクラムを全面的に採用する方針が打ち出された。
この方針に伴い、品質保証の仕組みを構築し、アジャイル開発に対する社内外の懸念を払拭するために、国際的な品質マネジメント規格である ISO 9001 の認証を取得することが必要となった。
当社には既に、ソフトウェア開発を前提として QMS を構築している組織がある。しかし、既存の QMS は特定のソフトウェア開発プロセス（ウォーターフォール）と密結合しており、顧客の要望や社会情勢の変化への迅速な追従が難しいという課題があった。DIC の QMS 構築においては、アジャイル開発の特性を活かしつつ、ISO 9001 の要求を満たす、新しいアプローチが必要とされた。

2.改善したいこと
本活動における主要な課題は、アジャイル開発（スクラム）を実践する DIC において、以下を考慮した QMS を構築・運用し、ISO 9001 認証を取得することであった。
1. ISO 9001 適合性。ISO 9001 の要求事項を網羅し、第三者による審査に合格する QMS を構築する必要がある。
2. アジャイル開発との整合性。従来と同様の QMS では、アジャイル開発やスクラムの基本的な考え方（反復的な開発、変化への対応、自己管理の重視など）の実現に差し障る恐れがある。
3. 開発プロセスへの非依存性。ウォーターフォールなどの特定の開発プロセスの詳細に依存しない QMS の構築が求められている。ウォーターフォール以外の開発プロセスを採用したいチームや、プロダクトやプロセスに対して積極的に仮説検証を行いたいチームにとって、十分な柔軟性を確保する必要がある。
4. 社内理解と信頼の獲得。アジャイル開発そのものや、アジャイル開発における品質保証の進め方が理解されていない。これを明らかにし、経営層やアジャイル開発を経験したことがない関連部門の理解と信頼を得る必要がある。

3.改善策を導き出した経緯
当初、既存の IoT 推進組織で運用されていた ISO 9001 認証済みの QMS を適用、あるいは修正・拡張して構築することを考えた。しかし、この QMS はウォーターフォールを前提とした進め方や成果物に強く依存しており、前述した課題を解消することは難しいと判断した。
この経験から、アジャイル開発に活用できる新たな QMS の構築にあたり、以下の要件を定義した。
- ISO 9001 の要求事項を満たすこと
- アジャイル開発（スクラム）を主要な開発手法として支援できること
- 将来的に他の開発手法（例：カンバン、ウォーターフォールの一部利用）も選択・追加できる柔軟性・拡張性を持つこと

これらの要件を満たすためのアプローチとして、ソフトウェアアーキテクチャにおける抽象化や関心事の分離の考え方を QMS の構造設計に応用することを着想した。すなわち、普遍的な ISO 9001 の要求事項と、変化しうる具体的な開発プロセスとの間に、開発プロセスに依存しない「抽象プロセス」レイヤーを設けるというアイデアに至った。

4.改善策の内容
上記の経緯に基づき考案・構築した改善策は、「抽象プロセス」を持つ QMS アーキテクチャである。その構造は以下の通りである。
- レイヤー1: ISO 9001 要求事項
- レイヤー2: 抽象プロセス
- レイヤー3: 特定の開発プロセスのガイドライン等

- レイヤー4: チームごとのプロセス実装

このアーキテクチャの核心はレイヤー2 の抽象プロセスにある。抽象プロセスを設けることにより、以下の利点が生まれる。

- 特定の開発プロセスからの独立。QMS の中核（レイヤー2）が特定の開発プロセスに依存しないため、スクラム以外を導入する場合でも、レイヤー2 を大きく更新せずに、レイヤー3 のガイドラインを追加・修正することで対応可能となる。
- スクラムのプラクティスとの整合。レイヤー3 のガイドラインで、スクラムのイベントや成果物を記載することができる。

抽象プロセスを介することで、ISO の要求とアジャイルプラクティスとの対応関係が明確になり、かつ特定の実装詳細に QMS が縛られることを防いでいる。この QMS アーキテクチャ自体が、本活動における主要な改善策である。

5.改善策の実現方法

抽象プロセスを持つ QMS の構築と ISO 9001 認証取得は、以下のステップで進めた。なお、実際にはシーケンシャルに進めたわけではなく、それぞれの文書の整合性をとりながら徐々に全体を進化させていくアプローチをとった。

1. QMS の全体像の定義：抽象プロセスにより ISO 9001 と具体的な開発プロセスを結びつける全体像を描いた。
    - 抽象プロセスの定義: ISO 9001 の各要求事項を分析し、開発ライフサイクル全体をカバーしつつも開発プロセスに依存しない品質保証活動（抽象プロセス）を定義した。
2. アジャイル開発ガイドラインの作成: 定義された抽象プロセスを、DIC で実践するスクラム開発において具体的にどのように実現するかを記述した「DIC アジャイル開発ガイドライン」を作成した。
    - このガイドラインの内容は、アジャイル開発を前提としてスピード感と品質の両立を狙った品質保証の仕組みに基づいている。具体的には、以下を前提として各スクラムチームのプロダクトオーナーに出荷権限を委譲することとした。

- 「完成の定義（Done の定義）」をベースラインに基づいて作成し品質保証部長のレビューで合意する
- Done の定義に組み込むテストについて、確立された方法により設計方針を策定する
- スクラムチームに、プロダクトオーナーに先んじて Done の定義に基づき判定する、品質技術者（QE）が参加する

3. 認証審査への対応: QMS 文書体系を整備し DIC メンバーに公開、QMS 運用を開始した。 また、QMS の運用状況を確認するための内部監査、及び、QMS の構築・運用状況をマネジメントに報告するマネジメントレビューを実施。2025 年 4 月より第三者認証機関による審査を受け、2025 年 7 月に ISO 9001 認証を取得した。

6.改善による変化や効果

本活動により、柔軟性を持つ QMS フレームワークを確立できた。これにより、将来的な開発プロセスの変化や追加、組織の成長に柔軟に対応できる基盤が構築された。更に、この QMS は、スピード感と品質を両立した品質保証の仕組みに基づいており、当組織のアジャイル開発に適している。

7.改善活動の妥当性確認

ISO 9001 認証の取得を達成し、ソフトウェア開発において主としてアジャイルを実践している組織に対する ISO 9001 認証の可能性を提示できた。

また、今回構築した品質保証の仕組みを社内のスクラムチームに適用した結果、チームの自律性向上等の良い効果が見られた。

参考情報

[1] IR DAY 2024 三菱電機の経営戦略
https://www.mitsubishielectric.co.jp/news/2024/pdf/0529-a1.pdf
[2] 新卒採用・インターンシップ　三菱電機を「知る」　製作所・研究所　DX イノベーションセンター
https://www.mitsubishielectric.co.jp/saiyo/graduates/philosophy/place/dic/index.html
[3] IR DAY 2025 三菱電機の経営戦略
https://www.mitsubishielectric.co.jp/ja/pr/2025/pdf/0528-1.pdf

2B2　現場から始めるアジャイルの文化醸成　～製造部門での挑戦～　静かな職場にアジャイルの芽を育てた 3 つのアクション

村瀬勇磨（株式会社 SHIFT）

＜タイトル＞

現場から始めるアジャイルの文化醸成　～製造部門での挑戦～

＜サブタイトル＞

静かな職場にアジャイルの芽を育てた 3 つのアクション

＜発表者＞

氏名(ふりがな)：村瀬　勇磨（むらせ　ゆうま）
所属：株式会社 SHIFT　製造ソリューションサービス部

＜共同執筆者＞

氏名(ふりがな)：
所属：

＜主張したい点＞

製造部門においてアジャイルを単なる手法ではなく文化として根付かせるため、地域特性や組織の現実に向き合い、上司との信頼構築、楽しく学べる仕掛け、現場密着の活動を通じて、勉強会発足や資格挑戦などの行動変化を促進した。結果として、受け身だった組織に対話と挑戦の空気が芽吹きはじめている。

＜キーワード＞

アジャイル文化、製造業、組織変革、スクラム、関西文化、上司との信頼構築、学習型組織、勉強会、定量評価、ワークショップ、心理的安全性、若手育成

＜想定する聴衆＞

製造業の現場責任者、アジャイル推進担当者、文化醸成に悩むすべてのマネージャー、リーダー

＜活動時期＞

2024 年 9 月～2025 年 6 月（現在も継続中）

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

- □ 着想の段階(アイデア・構想の発表)
- ☑ 改善活動を実施したが、結果はまだ明確ではない段階
- ☑ 改善活動の結果が明確になっている段階
- □ その他(　　)

＜発表内容＞

1.背景

弊社では事業部制を取り入れており、技術に特化したソリューション部、インダストリーごとに分かれた事業部側がある。私はもともとソリューション部内のアジャイル専門組織に在籍していたが、2024 年 9 月から SHIFT 全体へのアジャイル浸透というミッションから、アジャイル専門組織から製造部門へ異動した。製造部門では、東日本、西日本とグループが分かれており、私は東日本グループの所属となった。しかし、異動先の製造部門内（約 400 名）では、変化への恐れ、無関心、自分事にならない空気があったため、この空気を変えるべく、文化醸成に着手した。

2.改善したいこと

- 「他人事」とされる空気
- リアクションのない会議
- 学び合いのない風土
- 若手の挑戦機会の不足

これらを「挑戦・学び・価値提供」が自然に行われる文化へ転換することが目的。

3.改善策を導き出した経緯

現場観察から課題抽出し、上司・メンバーとの対話、関西文化圏への理解などから、単なるアジャイル啓蒙ではなく、目的を意識した文化・関係性へのアプローチが必要だと判断。

4.改善策の内容

- **アジャイル人材倍増計画**：楽しくアジャイルを学べるプラットフォームとして、ワークショップや勉強会（ひよこクラブ、SM になろう 等）を立案、若手の育成、リーダーの育成、技術研鑽に着手
- **上司との信頼構築**：既存の組織 KPI に合わせた成果の可視化、対話を通じた活動の正当化
- **関西文化への対応**：直接対話や 1on1、カードゲームなどを通じた信頼関係構築と接触回数の増加

5.改善策の実現方法

- 各アクションを小さく始め、成功体験を周囲に可視化
- 若手を中心にしつつ、リーダー陣、ベテラン、外部有識者も巻き込む形で自走化促進
- 上司には組織課題の文脈で価値を訴求
- 関西圏の現場には定期訪問し、オンラインと対面のハイブリッドで接点を維持

6.改善による変化や効果

- 組織会議でのリアクション率 30%向上
- メンバー主体での勉強会 2 件新規発足
- 社内外の資格取得への挑戦者が増加（月間取得が平均 1 件→6 件へ向上）

7.改善活動の妥当性確認

- 若手、上司、関西支部とステークホルダー全体に意識変化が見られた
- 「アジャイル＝楽しい」という空気の醸成の兆しあり
- 部署問わず、社内横展開も開始しつつあり、SHIFT 全体への文化浸透に弾み

参考情報

[1]　Regional Scrum Gathering Tokyo 2023 登壇紹介動画（YouTube）
https://www.youtube.com/watch?v=9fMs2jcLjns
本動画は、Regional Scrum Gathering Tokyo 2023 にて「アジャイルな仕事場の創出事例」として登壇した際にも紹介した、過去私が実施した SHIFT 社内での文化醸成活動の一環です。若手の主体性を引き出す工夫や、楽しさを通じた学びの場づくりの実例として活用されています。

2B3 ウォーターフォール開発にアジャイルの風を！～アジャイルプラクティスの実践でチーム力を引き出す～ 中野安美（Agility Design 株式会社）

＜タイトル＞

ウォーターフォール開発にアジャイルの風を！

＜サブタイトル＞

～アジャイルプラクティスの実践でチーム力を引き出す～

＜発表者＞

氏名(ふりがな)： 中野 安美(なかの やすみ)
所属： Ａｇｉｌｉｔｙ Ｄｅｓｉｇｎ株式会社
※発表者と共同執筆者の２名で発表する

＜共同執筆者＞

氏名(ふりがな)： 吉田 圭吾(よしだ けいご)
所属： ニッセイ情報テクノロジー株式会社

＜主張したい点＞

ウォーターフォール開発にスクラムやアジャイルのプラクティスを取り入れることで、チームメンバーのタスク状況を可視化し、カイゼン活動などを通じて自立的な行動を促進し、チーム力の向上を図りました。また、エンゲージメント調査でも、導入したチームの方が良い結果が得られました。

＜キーワード＞

ウォーターフォール開発
スクラム
アジャイル

＜想定する聴衆＞

ウォーターフォール開発を行っているが、プロジェクト運営に課題をもっている方、自立的なチームを目指したい方

＜活動時期＞

2023 年 12 月～現在

＜活動状況＞ ：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
□ 改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階

＜発表内容＞

1.背景

長年にわたりウォーターフォール型の開発プロセスを採用しているが、要件定義から設計、実装、テスト、リリースまでを段階的に進めるこの手法は、金融・保険業界における高い品質要求や厳格なスケジュール管理に適しており、同社の業務においても安定した成果を支えてきた。

しかし近年では、窓販系保険契約管理システムを中心とした保険開発において、設計・実装をリードする中核層の人材が不足しており開発体制の内製化が進められていた。そのため、若手社員の開発スキル向上と自立的に業務を遂行できる体制づくりが求められた。

この取り組みが始まった時期にコロナ禍が重なり、働き方はリモート主体へと急速に移行した。しかし、育成方針は従来のフロア中心の運用と大きく変わらなかったため、日々のコミュニケーションやチームの一体感が十分に形成されず、進捗報告が形式的にとどまるなど、メンバー間の相互理解や協働意識の醸成に課題が生じた。

こうした背景を踏まえ、弊社ではウォーターフォール開発の枠組みを維持しつつも、アジャイル開発、特にスクラムのエッセンスを取り入れることで、チームのコミュニケーションの活性化と自立性の向上を図っている。これにより、社員一人ひとりが柔軟性と自律性を持って変化に対応し、保険開発人材としての成長を促進する環境づくりを進めている。

2.改善したいこと

・開発チームのコミュニケーション、チームの一体感醸成
・チーム内の作業状況の透明性
・属人化と、特定の人に作業が集中する状況
・目の前の作業に没頭しすぎて、チームとしての課題に感じていること等の改善が進まない状態
・チームの自立性と協働体制
・内製化の実施による若手社員の業務知識向上

3.改善策を導き出した経緯

昨年、部署内でアジャイル研修を実施しており、その中でウォーターフォール開発における課題に対しても、アジャイルの考え方やプラクティスを部分的に取り入れることで改善の可能性があると感じた。そこで、ウォーターフォール開発の枠組みを維持しながらも、効果が出やすいアジャイルプラクティスを選定し、導入を進めることとなった。

導入にあたっては、アジャイルコーチの伴走支援を受けながら、スクラムイベントや改善活動を実践し、徐々にチームが自立的に取り組めるような体制を整えていった。

4.改善策の内容

スクラムフレームワークの一部を導入し、以下の状態を目指した。

1. 進捗管理運営の改善とチームワーク促進を目指し、朝会をスクラムのプラクティスを取り入れた運営へ見直しする

2. 朝会運営の見直し
   1週間サイクルのリズムを作り、毎週改善活動を繰り返すことで、仕事のプロセス改善とメンバー間の助け合い促進を通じて、1週間の達成確度を高める。

項目	目指すこと
チームの自立化促進	・チームが自立的に最適な仕事の進め方を考え、起きている状況をもとに適応させながらゴール達成を目指す ・個人ワーク中心から、チームがソロ/ペア/モブなどの作業形態や効果的な進め方を選択できるようになる
仕事の見える化	・miroを活用してタスク/課題などチーム状況の透明性を高める ・タスクを分解して書く、常に最新の状況に更新することを定着する
カイゼン活動	・現状は個人視点の内容が多い状態だが、”チーム”のカイゼンに目を向け、日々の仕事の進め方やコミュニケーション方法などの改善を行なうことで、チーム全体の成長を促進する
チームワーク向上	・個人ワークからチームワークへ意識を変え、チームで協力しながら仕事を進める状態とする ・なんでも話し合える状態を作ることで、問題の早期検知やメンバーの協働を引き出す
ファシリテーション力向上	・チームメンバーの意見を引き出し、チームとしての合意形成を支援する中で、ファシリテーションのスキルを高める

5.改善策の実現方法

従来から毎日 30 分間朝会を実施していたが、1 週間のトータル時間は変えずに、スクラムのイベントを活用して、1 週間の計画→日々の状況確認→ふりかえりのサイクルをまわるようにした。

■ 1週間の運営
- 曜日によって実施内容を見直し、計画⇒日々の状況確認⇒ふりかえりのサイクルをまわす。
- 祝日がある週は、適宜チームで調整する。

[現在]	月曜日	火曜日	水曜日	木曜日	金曜日	合計時間
実施イベント	朝会	朝会	朝会	朝会	朝会	
実施内容	タスク状況確認、課題の共有					
所要時間	30分	30分	30分	30分	30分	150分

[見直し]	月曜日	火曜日	水曜日	木曜日	金曜日	合計時間
実施イベント	計画	デイリースクラム	デイリースクラム	デイリースクラム	ふりかえり	
実施内容	・1週間のチームゴールを設定 ・1週間のタスク確認	・前日の状況確認 ・課題の共有 ・今日やる事確認	・前日の状況確認 ・課題の共有 ・今日やる事確認	・前日の状況確認 ・課題の共有 ・今日やる事確認	・ゴール達成状況の確認 ・ふりかえりと改善	
所要時間	45分	15分	15分	15分	60分	150分

改善を進めるにあたって、最初のころにはいくつかの課題が顕在化していた。ファシリテーターのやり方が分からず、ファシリテーター一人が話し続ける状態になったり、指示がないと誰も発言せず沈黙が続くといった場面が多く見られた。

このような状況の背景には、ファシリテーターの経験者がいなかったことがあり、進行方法については手探りで模索しながら進める必要があった。特に、金曜日に実施する 1 週間のふりかえりでは、どのような点を課題として取り上げるか、深掘りの仕方や Try（次に試すこと）の出し方に苦労した。

沈黙が続く場面では、反応の少なさが問題だと感じ、自分がファシリテーターでない時でも積極的に反応するように働きかけた。その結果、反応の少なさ自体を課題として取り上げ、Try を設定することで徐々に改善が進んだ。特に効果的だった Try としては、「ミュート解除」や「月曜・金曜のアイスブレイクの実施」が挙げられ、これによりチーム内の発言が増え、コミュニケーションが活性化した。

6.改善による変化や効果

・チームの自立化促進

リーダーからタスクを指示される状態から、自らがサインアップしてタスクをとったり、他のメンバーが何をやっているか、また遅

れているものを助ける動きに変化した。

・仕事の見える化

1 週間の作業タスク状況をカンバンで可視化し、課題やチームのルールを掲示して、チームの今の状態がわかりやすくなった。

・カイゼン活動

毎週ふりかえりを行うことで、チーム内のコミュニケーション、仕事の進め方の改善を促進できた

・チームワーク向上

当初は指名されないと話しはじめない状態であり、沈黙の時間もあったが、ファシリテートをメンバーが交代で担当することで、自分が担当するもの以外の状況の理解が進み、チームメンバーのコミュニーションが活発化した。また、「チームのありたい姿」をディスカッションすることで、目指すチームの状態を全員で決めて、その状態になるような行動に繋がっている。

・若手社員の開発スキル向上と自立化

今回の取り組みにより、チームの自立性と協働体制が着実に整備されつつある。メンバーは自発的にタスクを選択し、互いにフォローし合う姿勢が定着し始めており、役割分担も柔軟に対応できるようになった。これにより、個々の業務に対する責任感や主体性が高まり、チーム全体の生産性と連携力が向上している。

品質面では当初課題もあったが、継続的な育成と知識共有により、メンバーのスキルが向上。現在では業務理解度や対応力も高まり、安定した内製化体制の構築が進んでいる。今後は、スクラムのエッセンスを取り入れた開発手法を通じて、さらなるチームの自立性とコミュニケーションの活性化を図っていく予定である。

・業務改善への意識向上

チームが活性化したことで、メンバーが意見を言いやすくなり、既存の設計ドキュメントや開発プロセスにとらわれることなく、より良い記載方法や業務プロセスの改善に向けた提案が徐々に生まれるようになってきた。これにより、従来のやり方に固執せず、柔軟かつ前向きに改善活動を進められる環境が整いつつある。

7.改善活動の妥当性確認

改善活動の成果として、若手メンバーが率先して改善提案を行うようになり、これがベテラン社員にも良い影響を与え、チーム内のコミュニケーションが活性化した。これにより、チームの一体感が醸成され、心理的安全性の高い環境が整いつつある。

また、改善活動の成果が認められたことで、所属長の主導のもと、活動の裾野を広げる取り組みが進んでいる。現在では、ブロック単位でワークショップを通じた改善活動が始まっており、組織全体への展開が加速している。さらに、スキルマップの見える化を進めることで、従来は人事制度上オープンにできていなかったスキル情報を共有し、育成の促進にもつなげている。

前段で挙げた以下の課題に対しても改善が見られた。

・開発チームのコミュニケーション、一体感醸成：若手の積極的な提案により、チーム内の対話が徐々に活発化。

・チーム内の作業状況の透明性：miro ボードの活用により、業務内容やチームの状況の可視化が進行。

・属人化と特定の人への作業集中：スキルの共有と育成によって、業務の分散と平準化が進みつつある。

・目の前の作業に没頭しすぎて課題が改善されない状態：改善活動の定着により、チーム全体で課題を捉え、継続的な改善が可能な体制が整ってきている。

・チームの自立性と協働体制：リーダーからタスクを指示される状態から、自らが主体的に行動し、助け合える動きも見られるようになった。

・内製化の実施による若手社員の業務知識向上：継続的な育成と知識共有により、業務理解度や対応力も高まり、安定した内製化体制の構築が進んでいる。

来年度に向けては、組織目標を全員が理解し、自分事として捉えることで、共通認識を持った活動の推進を目指している。これにより、チームや個人が主体的に改善に取り組む文化を醸成し、組織全体の活性化と持続的な成長につなげていくことに取り組む予定である。

参考情報

※なし

2B4　TDD ベースの高品質アジャイルにおける品質管理プロセス構築の取り組み　掛川悠（株式会社 NTT データ）

＜タイトル＞

TDD ベースの高品質アジャイルにおける品質管理プロセス構築の取り組み

＜サブタイトル＞

＜発表者＞

氏名(ふりがな)：掛川悠（かけがわゆう）
所属：　株式会社 NTT データ

＜共同執筆者＞

氏名(ふりがな)：
所属：

＜主張したい点＞

ミッションクリティカルシステムの高品質、アジャイルの柔軟性/アジリティ、テスト駆動開発（以降、TDD）の高生産性を高次元で両立するために、TDD をベースとした品質重視のアジャイル開発に最適化した品質管理プロセスを構築した。具体的には、試験設計およびすり抜けバグを重点監視する一方製造バグは管理対象外とするメリハリのある品質管理方針を策定し、ゲート型、モニタリング型を併用した多角的な定量分析手法を具備した品質管理プロセスを構築した。結果として、アジャイルの柔軟性/アジリティ、TDD の高生産性を損なわずに、試験設計に関する故障を早期に摘出し、既存ミッションクリティカルシステムのウォーターフォール開発以上の商用 AP 品質を実現した。

＜キーワード＞

ミッションクリティカル、品質保証、品質管理、品質評価、高品質、アジャイル、スクラム、スプリント、テスト駆動、TDD

＜想定する聴衆＞

- ソフトウェア品質保証の担当者
- 高品質が求められるアジャイル開発、テスト駆動開発従事者
- アジャイル開発の品質管理プロセスを模索している方

＜活動時期＞

2022 年～2023 年

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
□ 改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1．背景

ビジネス環境の変化に伴い、従来ウォーターフォール（以降、WF）型の開発が主流であった大規模ミッションクリティカルシステムにおいてもアジャイル開発へのシフトが進んでいる。筆者の所属プロジェクト（以降、当 PJ）では、ミッションクリティカルシステムならではの高品質を維持しつつ、柔軟性、アジリティ、そして生産性を高次元で両立させる開発としてテスト駆動開発（以降、TDD）をベースとしたアジャイル開発（以降、TDD 高品質アジャイル）を行っており、その品質管理プロセスを構築する必要があった。

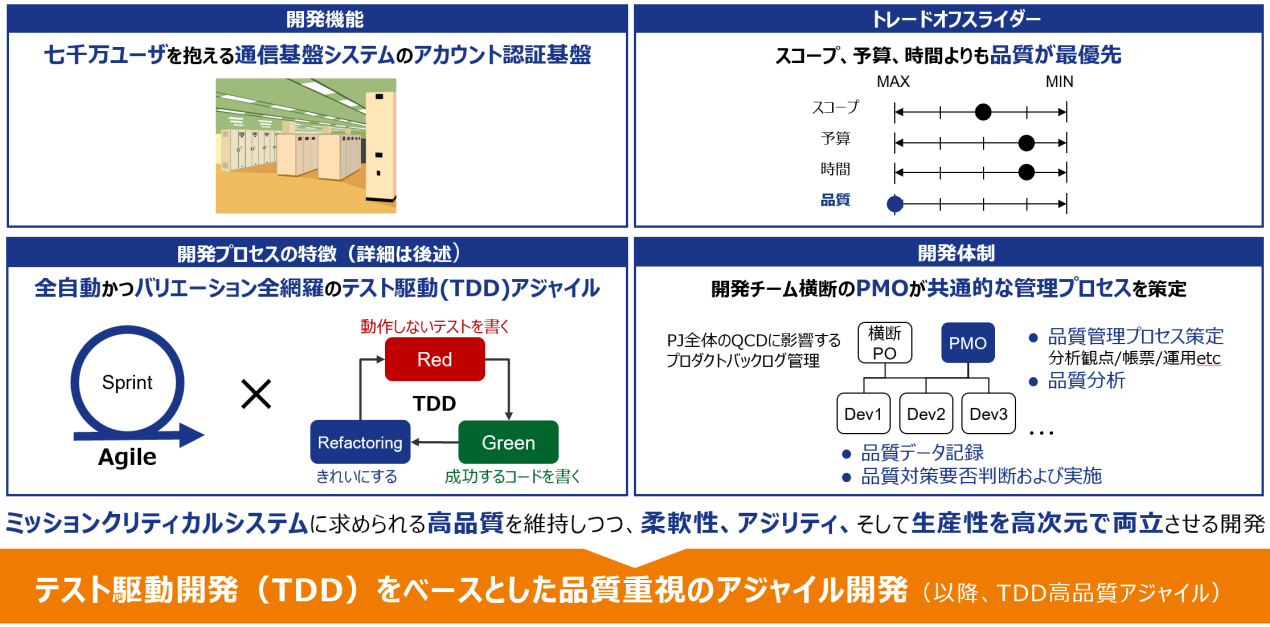


図 13 プロジェクト概要

2．改善したいこと

TDD 高品質アジャイルはミッションクリティカルシステムの高品質、アジャイルの柔軟性/アジリティ、TDD の高生産性を高次元で両立することが求められ、この観点から見ると先行研究（※）はいずれも個別事例に留まっている。従って、それぞれに求められる品質管理要件や先行事例を矛盾なく統合して、TDD 高品質アジャイルに最適化した実践的な品質管理プロセスを構築する必要があった。具体的には以下の 3 要件を満たす品質管理プロセスの構築を目指す。

① ミッションクリティカルシステムに求められる高品質を実現可能な品質管理
当 PJ の WF 開発の AP バグ密度目標値 0.03 件/Kstep クリア

② アジャイルの柔軟性/アジリティを阻害しない品質管理

③ TDD の高生産性を阻害せず、テスト駆動開発に最適な品質管理

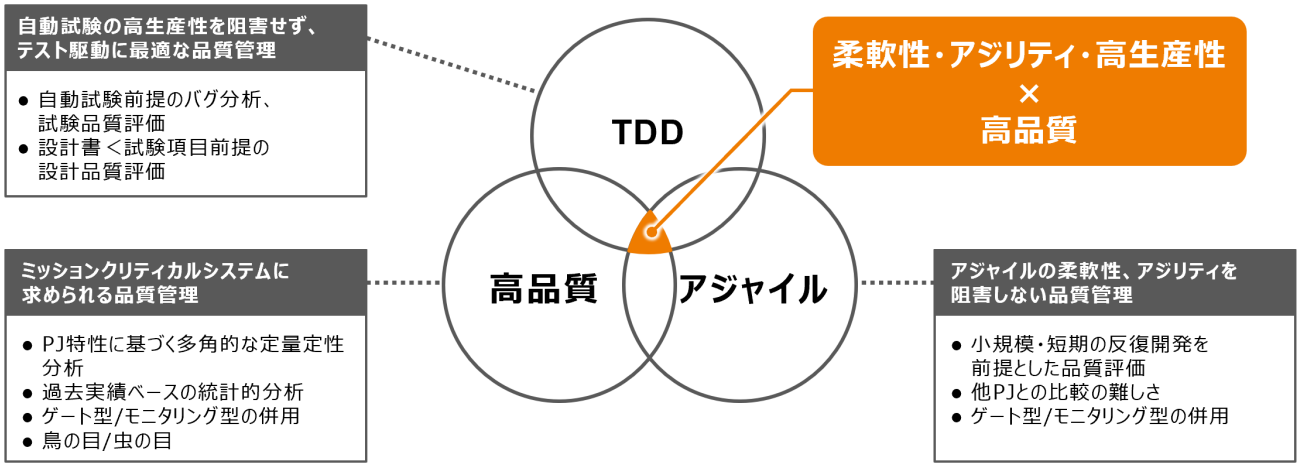


図 14 TDD 高品質アジャイルの品質課題

※関連する先行研究

関連する先行研究として、開発プロセス面では、山本ら(2023)が国内外のアジャイル開発事例、論文やインターネットを網羅的に調査した上で、デグレードを防止し効率的に品質担保するためにはテスト自動化および TDD が有効であり、特別な理由がない限りアジャイル開発でも実践すべきと結論づけている。この具体例として、清水ら(2022)はマトリクス型のテスト設計手法を用いた E2E テスト自動化によりアジャイル開発の品質及び生産性向上を両立している。また、アジャイルの品質管理については、綾野ら(2024)がストーリーポイント（以降、SP）を用いた時系列評価による品質評価の有効性を示している。一方で SP はチーム毎の固有値であることからチーム間の比較はできない。この課題に対して、中本ら(2022)は SP の基準をチーム間で統一するテーラリング手法を提案している。また、町田(2021)は従来のプロダクト品質やプロセス品質に代わる手法として、アジャイル開発チームの成熟度や健全性を表すリソース品質に着眼した品質管理手法を提案している。

3. 改善策を導き出した経緯

以下の 3 ステップで改善策を導いた。

① TDD 高品質アジャイルの開発プロセスの特徴をおさえる

② TDD 高品質アジャイルの柔軟性、アジリティ、高生産性を阻害せず、同時に高品質を実現するための品質管理プロセスをゼロベースで検討

③ 大規模ミッションクリティカルの WF で培った品質管理ノウハウやアジャイルの品質管理に関する先行事例を極力活用した上で、合わない部分はテーラリングする

ステップ①は②③の前提にあたるため本章で整理結果を述べておく。開発プロセスの大枠としては、基本設計（以降、BD）でサービスの外部設計及びマイクロサービスのアーキテクチャ設計を行い、製造 UTSprint でマイクロサービス設計～単体テストを実施し、最後に STSprint でマイクロサービス間結合試験を行う。製造 UTSprint では試験シナリオ毎に試験の前提条件および入出力パラメータ値（表示画面/リクエストの要求・応答項目値 etc）を整理したマトリクス（以降、サービスマトリクス）を作成し、ここから自動生成したテストコードにより、マイクロサービスの IF を全網羅した自動テストを行う。システムテスト（以降、ST）では、BD で作成したシステムテスト項目から自動生成されるテストコードを用いて、システムの外部 IF を全網羅した自動テストを行う。

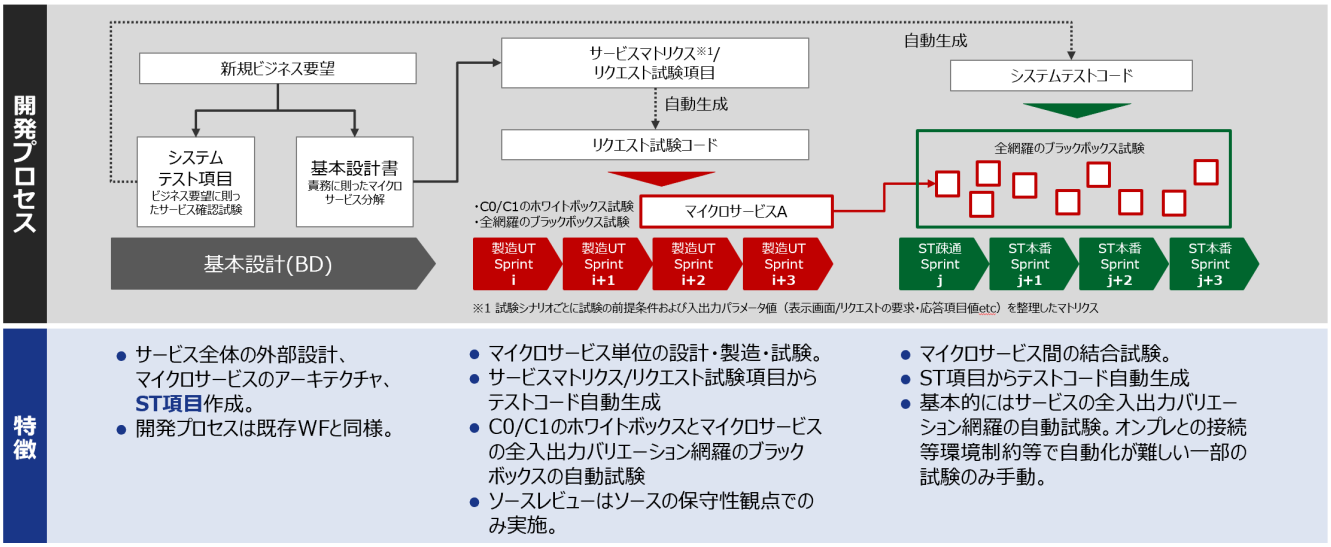


図 15 TDD 高品質アジャイルの開発プロセス

4. 改善策の内容

TDD 高品質アジャイルの開発プロセスの特徴を踏まえて、品質管理プロセスの基本方針を以下の通り定めた。本方針に基づいて品質管理プロセスを構築し、実案件を通じて妥当性を検証する。

【方針 1】

非すり抜け製造バグは品質分析対象外とする。また、自動生成のテストコードも分析対象外とする。

理由）TDD の高効率性を活かすには自動試験でバグ票を分析するのは非現実的である。一方、品質観点でも自動試験で全入出力バリエーションが網羅性される前提なら製造バグの品質分析の意味合いは薄い（全入出力バリエーション網羅の自動試験ならデグレリスクはほぼゼロで、また、テストを全てパスすれば製造品質は担保されるため）。

【方針 2】

BD および製造 UTSprint の試験系成果物含む設計成果物を品質評価対象とし、既存 WF の設計工程と同等の定量定性分析を行う。

理由）方針 1 の前提を保証するには試験系成果物含む設計品質が既存 WF 以上に重要になるため

【方針 3】

すり抜け率をリアルタイム監視するとともに、すり抜けバグは傾向分析に加えて 1 件ずつすり抜け原因を深堀することで、前提の有効性を監視する。

理由）すり抜けバグは方針 1 の前提の崩れを示唆するため、迅速な検知と原因追及が必要となる

5. 改善策の実現方法

ゲート型とモニタリング型の品質評価を併用した。ゲート型の品質評価としては、BD、各製造 UTSprint、各 STSprint 終了時に偏り分析やすり抜け原因分析等の詳細な定量定性分析を行う。モニタリング型の品質評価としては、チーム/製造 UTSprint 毎の累積エラー・バグ密度及びすり抜け率を中心に効率優先の俯瞰的な定量分析を行う。

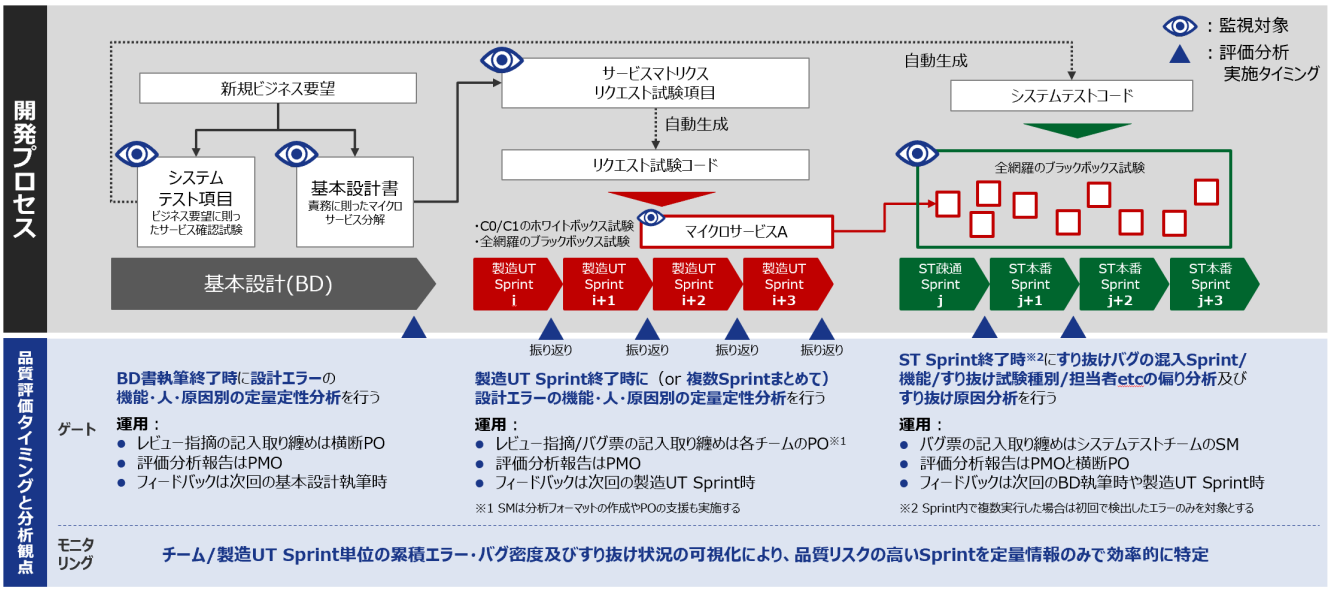


図 16 品質評価タイミングと分析観点

評価タイミングごとの分析内容と結果は以下の通り。

5.1 ゲート型：BD

BD 書および ST 項目のレビュー指摘に対して、頁数ベースのエラー密度[件/頁]、レビュー密度[分/頁]で定量定性分析（担当者・機能・原因別等）を実施した。結果としては、設計書に加えて、ST 項目等の試験系成果物のエラーも摘出（全エラー中 25%強）しており想定通りの結果となった。

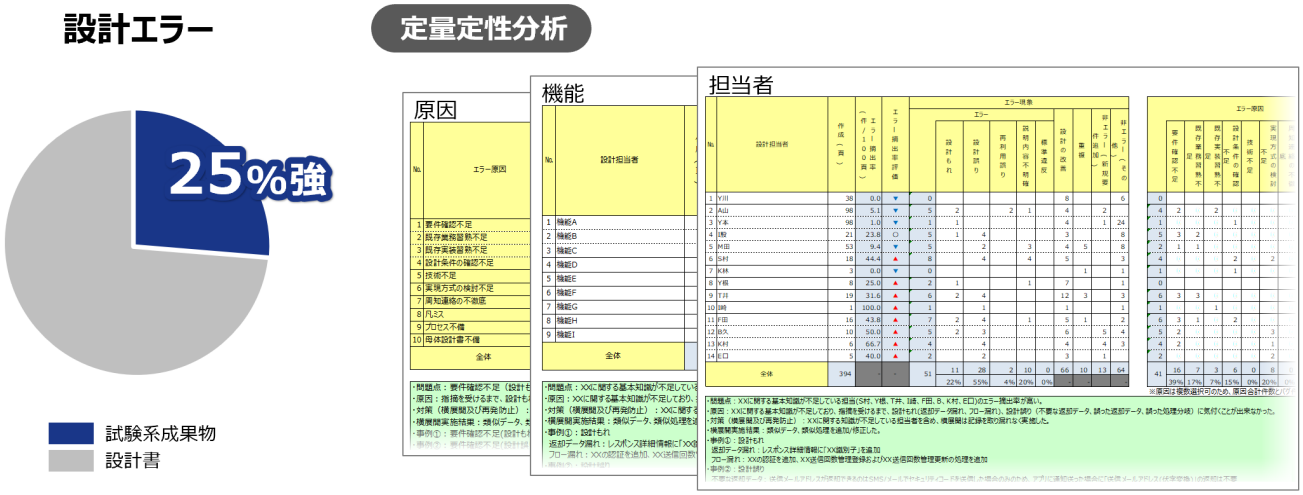


図 17 BD の分析例

5.2 ゲート型：製造 UT Sprint

まず、アジャイル開発における品質の定量評価の全量把握、指標値の策定方針について述べる。前者については、SP を用いた時系列評価による品質評価の有効性を示した綾野ら(2024)の事例、SP の基準をチーム間で統一した中本ら(2022)の事例を踏まえて、SP の基準をチーム間で統一した上で SP で全量把握を行うこととした。また、後者については、Sprint ごとの不具合状況推移に着目した品質管理手法を提案している射場ら(2023)の事例を参考に、製造 UTSprint 内の指標値は各 Sprint の実績推移ベースで定める方針とした。

続いて、製造 UT Sprint の分析例（Sprint ごと、Sprint 横断）とその結果について述べる。

① 製造 UT Sprint の分析例：Sprint ごと

サービスマトリクス/リクエスト試験項目のレビュー指摘に対して SP ベースのエラー密度で担当者・機能・原因別の定量定性分析を行った。結果として、大半を占める試験系成果物のエラーに対しても定量定性分析は実態に即した偏りを示しており分析の有効性を確認できた。

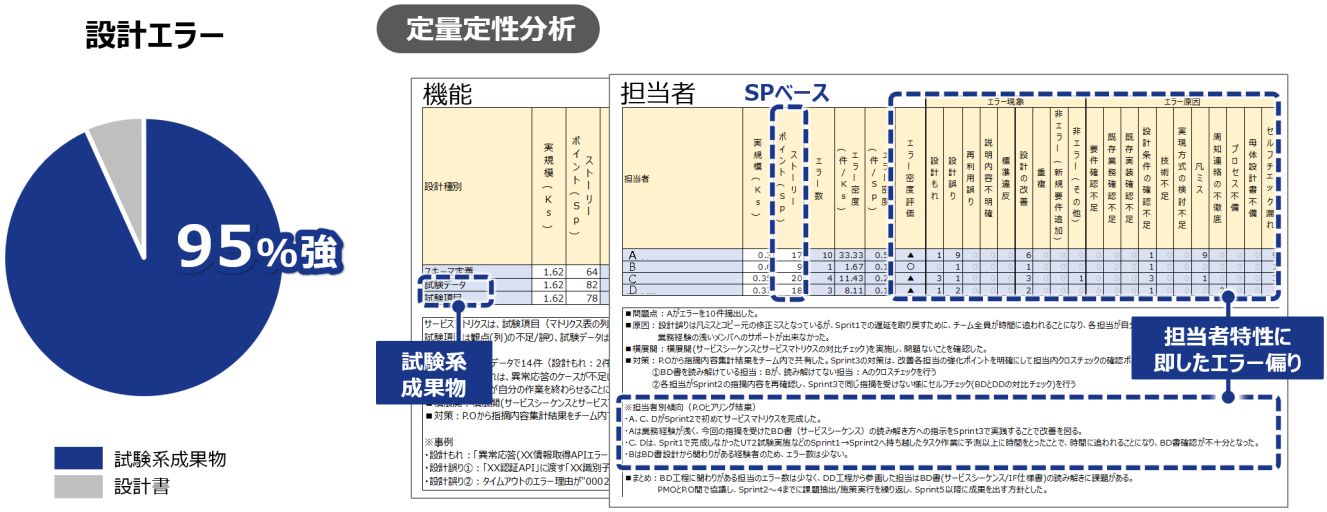


図 18 Sprint ごとの分析例

② 製造 UT Sprint の分析例：Sprint 横断

SP ベースのエラー密度で Sprint 横断の定量定性分析を行った。結果としては、SP ベースのエラー密度でも定量定性分析（特に時系列推移）は実態に即した傾向を示しており、分析の有効性を確認できた。下記の例では、立ち上げ期の Sprint のエラー密度が最も高く、習熟期は落ち着いているが、その後新規参画者が増えたタイミングで再びエラー密度が上昇しており、チーム状態とも整合する結果となっている。

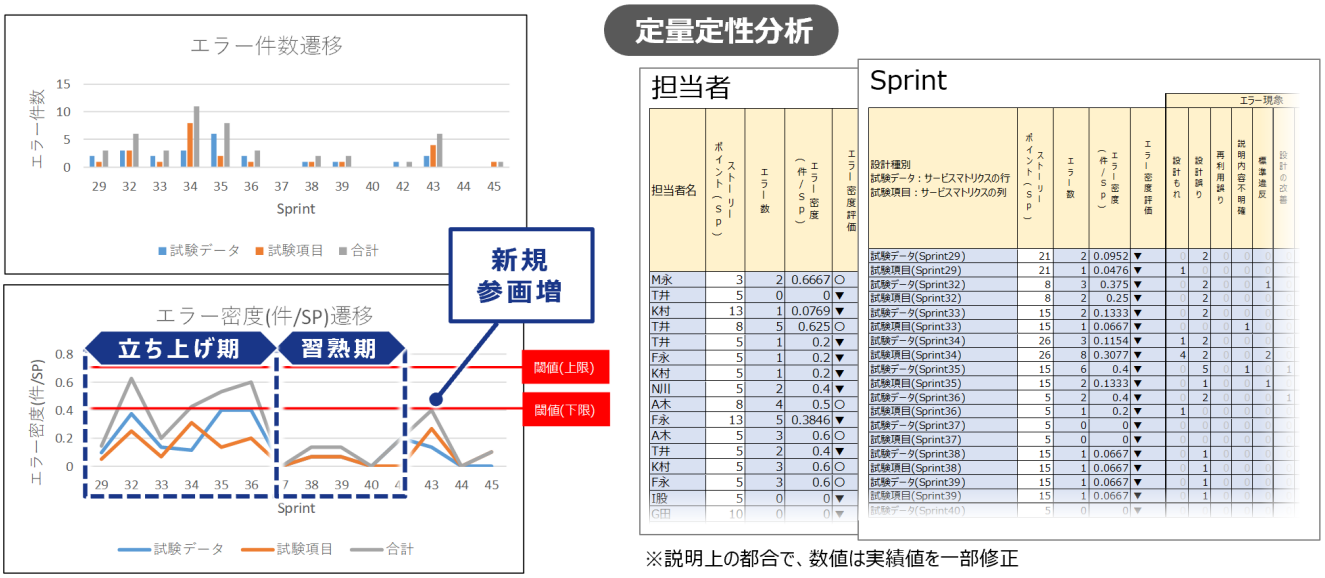


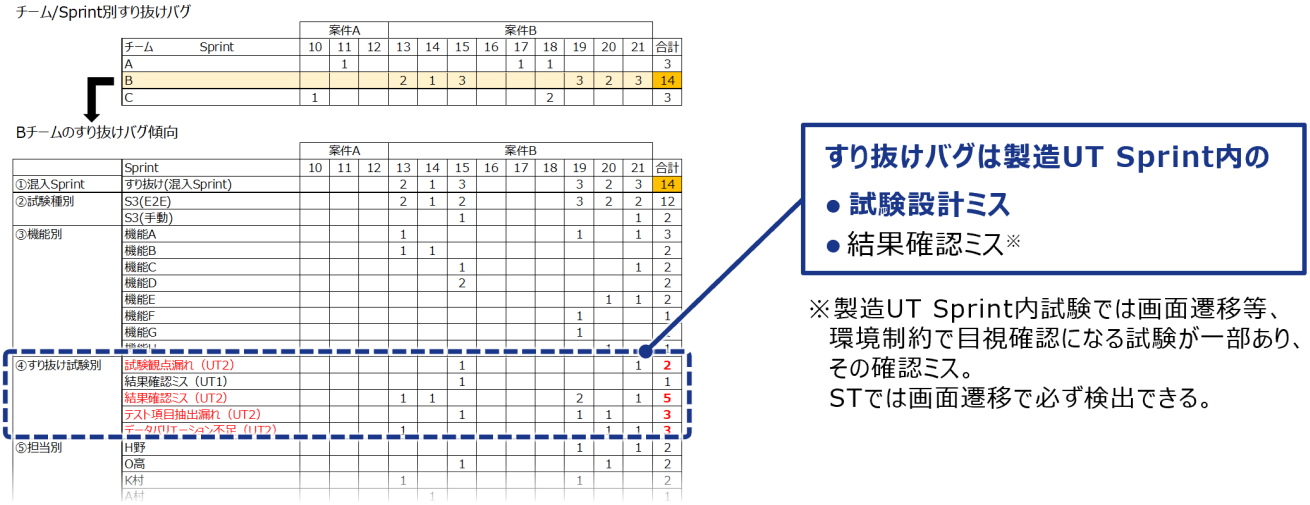
図 19 Sprint 横断分析例

5.3 ゲート型：ST

① すり抜けバグの傾向分析

ST 検出のすり抜けバグに対して、混入 Sprint/混入機能/すり抜け試験種別/担当者 etc の多角的な偏り分析を行った。結果としては、すり抜けバグは試験設計ミス（試験観点・項目・データバリエーション漏れ等）に偏りがあり想定通りの

結果となった。下記の例ではすり抜けバグはチーム B に偏っており、その内訳としては製造 UT Sprint 内の試験設計ミスが大半であった。



チーム/Sprint別すり抜けバグ

		案件A			案件B									
チーム	Sprint	10	11	12	13	14	15	16	17	18	19	20	21	合計
A			1						1	1				3
B					2	1	3				3	2	3	14
C		1								2				3

Bチームのすり抜けバグ傾向

		案件A			案件B									
	Sprint	10	11	12	13	14	15	16	17	18	19	20	21	合計
①混入Sprint	すり抜け(混入Sprint)				2	1	3				3	2	3	14
②試験種別	S3(E2E)				2	1	2				3	2	2	12
	S3(手動)						1						1	2
③機能別	機能A				1						1		1	3
	機能B				1	1								2
	機能C						1						1	2
	機能D						2							2
	機能E											1	1	2
	機能F										1			1
	機能G										1			1
	[illegible]													1
④すり抜け試験別	試験観点漏れ（UT2）						1						1	2
	結果確認ミス（UT1）						1							1
	結果確認ミス（UT2）				1	1					2		1	5
	テスト項目抽出漏れ（UT2）						1				1	1		3
	データバリエーション不足（UT2）				1							1	1	3
⑤担当別	H野										1		1	2
	O高						1					1		2
	K村				1						1			2
	A村					1								1

すり抜けバグは製造UT Sprint内の
- **試験設計ミス**
- 結果確認ミス※

※製造UT Sprint内試験では画面遷移等、環境制約で目視確認になる試験が一部あり、その確認ミス。
STでは画面遷移で必ず検出できる。

図 20 ST 傾向分析例

② すり抜けバグのすり抜け原因分析

製造 UTSprint からのすり抜けバグについて全件すり抜け原因分析を行った。結果として、成果物ベースで問題を適切に把握し、具体的な品質アクションに繋げることができた。すり抜けバグ一件一件に対して、下記の例のイメージですり抜け原因分析を行い、サービスマトリクスの改善等具体的な品質アクションにつなげた。

すり抜けバグ分析例

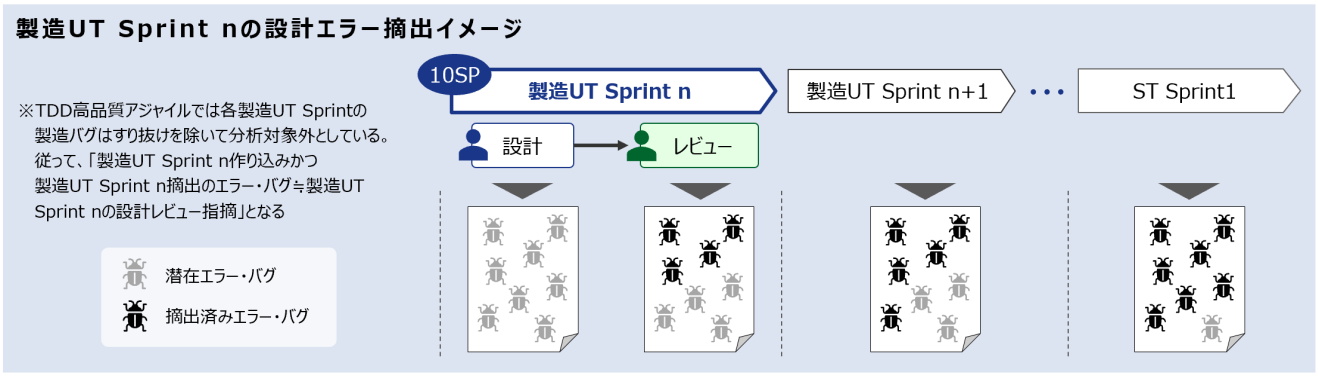
傾向分析用のメタ情報 ／ すり抜け原因を1件ずつ深堀

故障事象	一次解析		混入分析			すり抜け分析（概要）				すり抜け分析（詳細）			
概要	直接原因	故障機能	混入工程	混入Sprint	混入原因	要摘Sprint	要摘テスト	レビューすり抜け原因	試験すり抜け原因	大分類	要因分析	すり抜けバグに対する品質強化施策	今後の施策（恒久対処）
XX APIからXXリスト参照時に400応答	XXX	機能A	製造UT Sprint（製造）	19	要件確認不足	19	製造UT Sprint UT2	レビュー時の根拠ドキュメント漏れ	テスト項目抽出漏れ	サービスマトリクスで誤った定義をした（空文字）	空文字の解釈が曖昧であったため、担当者/レビュー間で意識齟齬が生じた	サービスマトリクスで空文字のデータが入るルートについて、ログアサーションを追加し、値チェックを行う	・デフォルト値をundefined、デフォルト型はstring/undefinedにする ・共通でSET/GETを作る

図 21 ST すり抜け原因分析例

5.4 モニタリング型：Sprint 横通しの俯瞰分析

モニタリング型の分析として、各製造 UTSprint 毎の設計（作り込み）品質、レビュー品質をそれぞれ累積エラー・バグ密度、すり抜け率で可視化することとした。ここで、累積エラー・バグ密度は累積エラー・バグ数を SP（or プログラムコード行数）で除算した値、すり抜け率はすり抜けバグ数を累積エラー・バグ数で除算した値である。累積エラー・バグ数は各製造 UTSprint 内で作り込んだ不具合のうちレビューまたは試験で摘出された不具合の総数なので、累積エラー・バグ密度は作り込み品質を表す。更に言えば、TDD 高品質アジャイルでは各製造 UTSprint における非すり抜け製造バグは起票・分析対象外としているため、累積エラー・バグ密度は実質的に、各製造 UTSprint の設計の作り込み品質を表す。そして、すり抜け率は設計で作り込んだ不具合のうちレビューで摘出できなかった不具合の割合を示すことから、各製造 UTSprint の設計レビュー品質を表す。



			すり抜けバグ	
	エラー・バグ数	4	1	2
	累積エラー・バグ数	4	5	7
設計品質	累積エラー・バグ密度 [件/SP]	0.4	0.5	0.7
レビュー品質	すり抜け率 すり抜けバグ数÷累積エラー・バグ数	—	20%	43%

図 22 製造 UTSprint の設計エラー摘出イメージ

結果として、作り込み or レビュー品質が悪い Sprint を定量情報のみで効率的に特定できた。一方、各 Sprint の開発機能が独立している場合、製造 UTSprint 内ですり抜けバグは出にくいため、タイムリーな品質問題の検知という点では課題が残る。

下記は ST 時点での各製造 UTSprint 毎の累積エラー・バグ密度、すり抜け率である。例えば、Sprint1 は Sprint1 作り込みの累積エラーバグ数が 25 件で、内訳として Sprint1 で摘出されたエラーが 18 件、Sprint2 以降で摘出されたすり抜けバグが 7 件となっている。下記の例から以下の示唆が得られる。

Sprint1：設計/レビュー品質ともに×→設計、レビューともに再点検

Sprint3：設計品質×だがレビュー品質〇→収束

Sprint5：設計品質〇だがすり抜けが多い→レビュー再点検

	設計（作り込み）品質			レビュー品質			
	累積エラーバグ数	累積エラーバグ密度	定量評価	累積エラーバグ数内訳 作り込みSprint	累積エラーバグ数内訳 すり抜け	すり抜け率	定量評価
BD	53	0.70	▼	51	2	3.8%	〇
Sprint1	25	5.97	▲	18	7	28.0%	▲
Sprint2	13	2.01	〇	11	2	15.4%	〇
Sprint3	67	15.69	▲	66	1	1.5%	〇
Sprint4	5	1.71	〇	5	0	0.0%	〇
Sprint5	9	2.74	〇	6	3	33.3%	▲
Sprint6	3	2.38	〇	3	0	0.0%	〇

図 23 モニタリング型：Sprint 横通し分析例

6. 改善による変化や効果

2 章であげた TDD 高品質アジャイルに最適化した実践的な品質管理プロセスの 3 要件はいずれもクリアしており、狙い通りの品質管理プロセスを構築することができた。詳細を以下に述べる。

① ミッションクリティカルシステムに求められる高品質を実現可能な品質管理 ➡ 評価：〇

商用リリース後 6 カ月で商用 AP 品質は既存ミッションクリティカルシステムのウォーターフォール開発以上で安定している。具体的には、商用 AP バグ密度は 0.01 件/Kstep で、当 PJ の WF 開発の目標値 0.03 件/Kstep をクリアしている。また、商用 AP バグはサービスマトリクスの試験パターン抽出漏れ、試験データ誤り等、試験設計ミス数件で製造バグはゼ

ロ件であり、TDD の特性に合わせて試験設計を重点監視対象とした品質管理プロセスの妥当性を示している。

② アジャイルの柔軟性/アジリティを阻害しない品質管理 ➡ 評価：○

アジャイル開発の特徴を踏まえた上で、先行事例や WF 開発における知見を活用した以下の取り組みにより、アジャイルの柔軟性/アジリティを阻害しない品質管理を実践できた。

- 小規模かつイテレーティブな開発スタイルを踏まえたゲート型、モニタリング型評価の併用
- グラフやマトリクスを多用した定量評価主体の分析
- SP ベースの定量評価
- Sprint 単体に加え、Sprint 横断の時系列推移も踏まえた評価
- すり抜けバグの重点監視

③ TDD の高生産性を阻害せず、テスト駆動開発に最適な品質管理 ➡ 評価：○

TDD の特徴を踏まえた取り組みにより、TDD の高生産性を阻害せず、テスト駆動に最適な品質管理プロセスを構築できた。具体的には、「【方針 1】非すり抜け製造バグは品質分析対象外とする。また、自動生成のテストコードも分析対象外とする。」により TDD の高生産性を損なわない効率的な品質管理になった。一方で、「【方針 2】BD および製造 UTSprint の試験系成果物含む設計成果物を品質評価対象とし、既存 WF の設計工程と同等の定量定性分析を行う。」を実践した結果として、想定通り一定量の試験設計エラー摘出があった。特に、製造 UTSprint ではエラーの大半が試験設計エラーであり、試験設計を重点監視対象とした品質管理プロセスの妥当性を示す結果となった。更に、「【方針 3】すり抜け率をリアルタイム監視するとともに、すり抜けバグは傾向分析に加えて 1 件ずつすり抜け原因を深堀することで、前提の有効性を監視する。」についても、ST 検出のすり抜けバグや商用バグは想定通りほぼ試験設計ミスであり、また、適切な品質対策のためのすり抜け原因分析の重要性も確認できた。

7. 改善活動の妥当性確認

7.1 総評

大規模ミッションクリティカルシステムの高品質、アジャイルの柔軟性/アジリティ、TDD の高生産性という一見相反する要件を高次元で両立するという非常に難易度の高い課題であったが、前章で述べた通り狙い通りの結果を出すことができた。この課題解決は顧客は元より当 PJ のプロジェクトマネージャや開発者を含む利害関係者全員の要望であったため、本活動を通じて彼らの期待に応えることができた。後述の残課題を解決して改善活動を継続していきたい。

7.2 考察 -WF 開発でもアジャイル開発でも変わらない品質管理プロセス構築の勘所-

最後に、品質管理プロセスを構築する上で最も基本的かつ重要な考えについて一段抽象度を上げて考察する。まとめると、以下の 3 点に集約される。

① 品質リスクの見極め

まず、品質リスクを見極めることが重要である。品質リスクとは、プロジェクトの成功を脅かす可能性のある要因であり、これを正確に把握することが品質管理プロセス構築の第一歩である。これを見極めるためには、開発初期段階から詳細なリスク評価を行い、潜在的な問題点を洗い出す必要がある。本活動では、TDD 高品質アジャイルの開発プロセスの特徴を踏まえて試験設計の品質が最重要かつ高リスクであると判断した。一方、自動生成されるテストコードや自動試験でバグ検出可能な製造品質のリスクは低いと判断した。

② リスクに応じたメリハリ

次に、リスクが高いところは重点監視し、リスクが低いところはほどほどに監視するメリハリが大切になる。品質管理のリソー

スは限られているため、すべての工程やプロセスを同じレベルで監視することは現実的ではない。TDD 高品質アジャイル開発においては、試験設計やすり抜けバグを重点監視する一方、非すり抜け製造バグや自動生成のテストコードは品質分析対象外とすることで、効率的な品質管理を実現した。

③ 効率的かつ高度な品質評価の仕組み

最後に、重点監視対象の品質を効率的かつ高度に可視化・評価する仕組みが重要である。品質最優先の場合、品質状況の可視化はともすると定性評価主体の非効率な方法になりがちだが、アジリティが求められるアジャイル開発には合わない。定量評価主体の多角的な分析観点と機械化の徹底により、効率的かつ高度な品質分析の仕組みが求められる。本活動では、小規模かつイテレーティブな開発スタイルを踏まえたゲート型、モニタリング型評価の併用やグラフやマトリクスを多用した定量評価主体の分析、SP ベースの Sprint 横断の時系列評価などがこの取組に該当する。これにより、効率的に品質問題の早期検知と対策が可能となった。

以上 3 点は実は WF 開発における品質管理プロセス構築の際に筆者が意識していたポイントであったが、今回の活動を通じて、これらは WF 開発でもアジャイル開発でも変わらない品質管理プロセス構築の勘所であることがわかった。1 章で述べた通り、昨今はビジネス環境の変化に伴い、従来 WF 型の開発が主流であった大規模ミッションクリティカルシステムにおいてもアジャイル開発へのシフトが進んでいる。そんな中、WF 開発の経験しかない開発者がアジャイル開発に習熟するには一定のハードルがある。品質管理においてもそれは例外ではなく、WF 開発で培ったノウハウを全て活かせるわけではなく、アジャイル向けに一定のテーラリングが求められる。しかし、WF とアジャイルは二律背反ではなく、表面的な違いはあれどその根底にある本質は共通していることも多い。本事例もその一つと考えられる。

7.3 残課題

- 各製造 UTSprint の開発機能が独立している場合、後続の製造 UTSprint 内ですり抜けバグは出にくいことから、本稿で提案したモニタリング型分析では品質問題の検知が遅れるという課題がある。分析観点やメトリクスの拡充も見据えてモニタリング精度を改善していきたい。
- バグ票起票から分析までの間にいくつかトイルが残っており、タイムリーかつ効率的な分析を行う上では品質情報の集計・可視化の更なる自動化、モダナイズが必要である。

参考情報

[1] 綾野未来, 滝澤健人, 杉原直樹, 宮本充, 高橋僚史, 内山航輔, 渕上恭平 (2024). ストーリーポイントを用いた品質評価手法の適用事例. プロジェクトマネジメント学会春季研究発表大会予稿集, 443-450.

[2] 清水歩, 加藤大受 (2022). アジャイル開発における E2E テスト自動化の効率的なテスト設計手法の研究. ソフトウェアエンジニアリングシンポジウム 2022 論文集, 29-35.

[3] 中本傑, 宮島雄一, 岡村龍也, 鈴森康弘 (2022). エンタープライズ向けアジャイル開発におけるスコープマネジメントの考察. プロジェクトマネジメント学会秋季研究発表大会予稿集, 340-350.

[4] 射場千尋 (2023). Sprint ごとの不具合状況推移に着目した品質管理手法の適用事例. プロジェクトマネジメント学会春季研究発表大会予稿集, 177-190.

[5] 別府薫, 佐伯明音, 遠藤圭太, 仁尾圭祐 (2023). ウォーターフォール開発に照らしたアジャイル開発の品質管理事例と考察. プロジェクトマネジメント学会秋季研究発表大会予稿集, 158-165.

[6] 町田欣史 (2021). リソース品質に着目したアジャイル開発における品質管理. プロジェクトマネジメント学会秋季研究発表大会予稿集, 301-304.

[7] 山本椋平, 石津大輔, 桑田直樹, 後藤卓司, 浅田隼人, 山村喜恒 (2023). アジャイル開発の品質確保に向けたマネジメントポイント. プロジェクトマネジメント学会春季研究発表大会予稿集, 219-225.

[8] 掛川悠，大貫正也，米原大輔(2024)． TDD ベースの高品質アジャイルにおける品質管理プロセス構築事例．プロジェクトマネジメント学会秋季研究発表大会予稿集，591-605．

3A1　生成 AI によるプロセスセルフアセスメント支援の実現性評価　池永直樹（株式会社デンソークリエイト）

＜タイトル＞

生成 AI によるプロセスセルフアセスメント支援の実現性評価

＜サブタイトル＞

なし

＜発表者＞

氏名(ふりがな)：　池永　直樹(いけなが　なおき)
所属：　株式会社デンソークリエイト

＜共同執筆者＞

なし

＜主張したい点＞

プロセス改善活動を活性化させる目的で、ソフトウェア技術者自らが実施者となりソフトウェア開発プロセスのチェックリストを用いて自己診断形式で実施するプロセスアセスメント(セルフアセスメント)の展開を考えている。セルフアセスメントの課題として、結果が実施者のプロセスに関する知識に大きく依存してしまう点がある。この課題の解決に生成 AI の活用を考え、生成 AI によるセルフアセスメント支援の実現可能性を評価した。結果として、生成 AI は初級アセッサーと同等レベルで納得感のあるアドバイスの出力が可能であり、生成 AI によるセルフアセスメント支援が実現可能であることが分かった。

＜キーワード＞

プロセスアセスメント、アセッサー、生成 AI、プロセス改善

＜想定する聴衆＞

プロセス改善への生成 AI の活用を検討又は推進されている方
プロセス診断への生成 AI の活用を検討又は推進されている方

＜活動時期＞

2024 年 6 月～

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
■改善活動を実施したが、結果はまだ明確ではない段階
□ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1.背景

筆者は、自組織のプロセス改善活動やプロセスアセスメントを担当している。プロセス改善活動をより活性化させるための手段一つとして、開発現場の技術者自らが開発プロセスを自己診断形式で実施するプロセスアセスメント(セルフアセスメント)の展開を検討している。

2.問題と課題

開発現場の技術者によるセルフアセスメントの結果が、開発プロセスの実態を表したものになりにくい、という問題がある。なぜなら、プロセスアセスメントの実施にはプロセスに関する十分な知識が必要であるが、開発現場の技術者はそのドメイン知識には精通してもプロセス知識は不足していることが多い。プロセスアセスメントモデル(以降、PAM)はそもそも抽象度が高く、たとえ PAM を基に作ったプロセスのチェックリストを用いても、技術者のプロセスに関する知識不足を十分補完できず、チェック項目の意味を誤解釈又は達成度合いを正しく理解せず自己診断してしまうためである。

そこでこの問題を解決し、組織のプロセス改善を推進させるために、「**セルフアセスメント実施者のプロセスの知識レベルに寄らず、開発プロセスの実態を表した診断結果を得られる**」ようになることを目指して取り組みを実施した。

3.解決策の提案

3.1 着想

生成 AI は要件定義、議事録管理、プログラム開発(コード生成、ペアプログラミングなど)、レビュー、テストなどソフトウェアエンジニアリングへの適用が進んでいる[5]。

その状況のもと、プロセスアセスメントへの適用事例はまだ少ないが、生成 AI がアセッサーを支援するアセスメントツール[6]などが登場し始めている。これらのように、生成 AI の特徴である自然言語処理、知識の広さ、文章読解力、コンテンツ生成は、自然言語中心で扱われるセルフアセスメントの精度向上に寄与する可能性が高いのではないかと考えた。

そこで、**セルフアセスメント実施者のプロセス知識が不十分でも、生成 AI のサポートがあれば自己診断の精度が向上するのではないか？**と考え、プロセスセルフアセスメントへの生成 AI 活用の実現性を評価することにした。

3.2.セルフアセスメントへの生成 AI の活用方法

生成 AI の活用方法として、チェックリストを用いてセルフアセスメントを実施することを前提とし、“最終的な達成度の判断はセルフアセスメント実施者に任せ、チェック項目に対するセルフアセスメント実施者の回答に対して生成 AI にアドバイスさせる”という、**セルフアセスメントの実施を生成 AI に支援させる**活用方法を考えた。

生成 AI を活用したセルフアセスメントを実施するために、「生成 AI を活用するプロセス」「チェック項目の構成」「プロンプトテンプレート」を作成した。なお、チェック項目は「設問」「設問の補足」「評定」「評定の根拠」で構成する。

4.評価

4.1.評価観点

生成 AI によるプロセスのセルフアセスメント支援が実現可能であるかどうかを以下の 3 つの観点で評価する。

観点 1：生成 AI が出力するアドバイスがアセッサーと同レベルであるか

生成 AI が出力したアドバイスに見逃しや不要な指摘がないかをアセッサーが評価する。

観点 2：生成 AI の評定がアセッサーの感覚と合っているか

「評定の根拠」に基づく生成 AI の評定とアセッサーの評定の一致度を評価する。生成 AI を活用したセルフアセスメント

実施手順としては最終的な評定はセルフアセスメント実施者に任せるが、生成 AI がどの程度の評定と考えてアドバイスを出力しているのかを確認する。

観点 3：セルフアセスメント実施者にとって納得感のあるアドバイスが出力されるか

生成 AI のアドバイスをセルフアセスメント実施者が利用することから、納得感があり受け入れられるものかを評価する。

“納得感”は ISO25010 の利用時品質のうち有効性、効率性、満足性を用いて確認項目を定義し、「良い/やや良い/やや悪い/悪い」の 4 水準で評価する。

4.2.評価方法と環境

セルフアセスメントを実施する立場であるプロジェクトマネージャー(又はリーダー)を対象に試行する。実際のプロジェクトを診断してもらう。それに対して生成 AI が出力したアドバイスをアセッサー及びセルフアセスメント実施者が評価する。アセッサーは、intacs 認定 Automotive SPICE Principal Assessor 資格[11]を保有する筆者が担当する。

生成 AI は GPT-4o(Azure OpenAI)を使用する。

4.3.対象プロセスとチェック項目

試行対象とするプロセスは Automotive SPICE V4.0 から、管理プロセス群の MAN.3 プロジェクト管理、支援プロセス群の SUP.8 構成管理、エンジニアリングプロセス群の SWE.1 ソフトウェア要求分析を選択する。

各プロセスの「設問」は、Automotive SPICE 4.0 実践ガイドブック[12]に掲載されているチェック項目を使用する。各プロセスのチェック項目数は、MAN.3：12 項目、SUP.8：12 項目、SWE.1：9 項目の合計 3 3 項目である。

また、「設問の補足」は実践ガイドブックの解説文に筆者が加筆修正を加えたものである。

4.4.評価データ

セルフアセスメント実施者 7 名が実際のプロジェクトを対象に自己診断した回答、及び筆者が恣意的に作成した回答の合計 161 件の評価データを得た。

5.評価結果と考察

5.1.観点 1 の評価結果と考察

試行で得られたアドバイスに見逃しや不要な指摘がないかを計測した。**アセッサー(筆者)と比較すると**見逃しは 45/161 項目(28%)、不要な指摘は 50/161 項目(31%)と**それぞれ 3 割程度多かった**。初級アセッサーのデータがあり、見逃し率 28%・不要な指摘率 4%である。これと比較すると見逃しは同水準であり、不要な指摘は多い結果となった。なお、この初級アセッサーのデータは，筆者が所属する組織の Automotive SPICE Provisional Assessor 資格[11]を保有し、プロセス改善業務経験は十分ありアセスメント経験が数回のメンバーの平均値である。

セルフアセスメントの性質上、**初級アセッサーと同等レベルであったので実務で活用可能なレベルと判断する**が、精度向上の余地はある結果となった。

5.2.観点 2 の評価結果と考察

生成 AI の評定とアセッサー(筆者)の評定の混合行列を作成し、評定の一致度を確認した。結果は次の通りである。

- 「完全一致」の割合：58%（94/161）
- 「完全一致＋ 1 水準違い」の割合：86%（139/161）

また、一致度の高さとそれが偶然ではないことを確かめるために、加重カッパ係数[13]を用いて検証したところ、カッパ係数=0.7494、Z 値=16.526、p 値＝2.372e-61 の結果が得られた。カッパ係数=0.7494 であり、生成 AI とアセッサー(筆者)の評定は「かなりの一致(substantial)[14]」が認められた。また、Z 値=16.526、p 値«0.001 であり、両者間

の一致は偶然によるものではなく、統計的に有意であることが示された。

上記の結果から、「完全一致＋１水準違い」の割合は 86%であり、**アセッサーに近い感覚で評定できる**と考える。

5.3.観点 3 の評価結果と考察

「やや良い」以上が、有効性:79%、効率性:93%、満足性:76%であった。

”良い”が”悪い”を明らかに上回っていることを確認する基準値を地方自治体などで重要事項を決議する場合に採用される「特別多数決」の 2/3 と考えたとき、確認項目すべてが基準値を大きく上回っている。

セルフアセスメント実施者から「知識が得られた」「間違いに気づいた」とのポジティブなコメントも得られている。

以上から、**生成 AI が出力するアドバイスは納得感があると**判断できる。

6. 生成 AI モデルの比較

4.1 の評価観点１, 2 について、より新しい生成 AI モデルである GPT-4.1 を使って再評価した。プロンプトは GPT-4o と同じものを使用した。

【観点１の結果】

- 出力されるアドバイス項目の総数が１.５倍に増加：１５２個→２２０個
- 見逃したアドバイス項目数が６０%減少：４６個→２０個
- 不要な指摘のアドバイス項目数に変化なし：６２個→６２個

【観点２の結果】

- 評定の一致度はどちらもかなり高く偶然でもないが、GPT-4o の方がより一致度は高い
- Not Rated でないのに Not Rated と評定するチェック項目が増加：２件→１７件

以上の結果から、新しい生成 AI モデルを使用するだけでアシスタントとしての能力が向上するが、より能力を引き出すにはプロンプトや設問などの調整が必要ということが分かった。

7. まとめと今後の展望

ソフトウェア技術者自らが自己診断の形式でプロセスアセスメントを行う方法（セルフアセスメント）があるが、「開発プロセスの実態を表したものになりにくい」という問題があった。そこで、セルフアセスメントの実施を生成 AI に支援させる方式を考え、その実現可能性を評価した。試行の結果、生成 AI は初級アセッサーと同等レベルで納得感のあるアドバイスの出力が可能であり、**生成 AI によるセルフアセスメント支援が実現可能**であると判断できる。

今後は以下に取り組み、アドバイスの精度向上及び実運用につなげたい。

<アドバイス精度の向上>

- さらに新しい生成 AI モデル（GPT-5 など）の活用
- プロンプトチューニング、設問の見直し

<実運用に向けた取り組み>

- セルフアセスメントを手軽に実施できる環境（ツール）の実装
- 標準プロセス定義書との統合（生成 AI に与える専門知識として標準プロセスの定義内容を用いる）

参考情報

[1] IPA/SEC, プロセス改善ナビゲーションガイド ～虎の巻編～, 2009/2/25
[2] IPA/SEC, プロセス改善ナビゲーションガイド ～プロセス診断活用編～, 2007/3/30
[3] SPICE Center　Presents SPICE Assessment Models and Reference Models, https://intacs.info/index.php/spice-center
[4] Jan Morenzin, Automotive SPICE® News and data from VDA QMC, 1st Asia SPICE Conference
[5] AI を用いたソフトウェア開発, https://www.ipa.go.jp/digital/ai/software-engineering.html
[6] Assessor Academy, AXIOM Co-Assessor -次世代型アセスメントツール-, https://assessor.co.jp/axiom/
[7] 多田麻沙子,徳本晋,栗田太郎,石川冬樹, ISO27017 に基づくクラウドセキュリティ監査業務に対する LLM の性能, ソフトウェア・シンポジウム 2024
[8] OpenAI prompt engineering, https://platform.openai.com/docs/guides/prompt-engineering
[9] Prompt Engineering Guide, https://www.promptingguide.ai/
[10] Jules White et al., "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT", 2023.
[11] international Assessor Certification Scheme, https://intacs.info/
[12] ビジネスキューブ・アンド・パートナーズ, Automotive SPICE 4.0 実践ガイドブック, 日経 BP, 2024 年 1 月 22 日
[13] カッパ係数（Cohen's kappa statistic）の情報まとめ, https://qiita.com/kazuo_reve/items/384c29d46cb3a18f827a
[14] J. Richard Landis and Gary G. Koch, The measurement of observer agreement for categorical data., Biometrics, Vol. 33, No. 1 (Mar., 1977), pp. 159-174

3A2 バラバラな設計書フォーマットを全社で統一！CommonStyle Methodology 普及展開活動と統一効果 東芝デジタルソリューションズ（株）の取り組み 東井礼佳（株式会社東芝）

＜タイトル＞
バラバラな設計書フォーマットを全社で統一！CommonStyle Methodology 普及展開活動と統一効果

＜サブタイトル＞
東芝デジタルソリューションズ（株）の取り組み

＜発表者＞
氏名(ふりがな)：東井 礼佳（とうい あやか）
所属：(株)東芝 総合研究所 デジタルイノベーション技術センター ソフトウェアプロセスイノベーション部

＜共同執筆者＞
氏名(ふりがな)：牛山 佳奈（うしやま かな）
所属：(株)東芝 総合研究所 デジタルイノベーション技術センター ソフトウェアプロセスイノベーション部
氏名(ふりがな)：大森 麻理（おおもり まり）
所属：(株)東芝 総合研究所 デジタルイノベーション技術センター ソフトウェアプロセスイノベーション部
氏名(ふりがな)：宮田 真紀子（みやた まきこ）
所属：(株)東芝 総合研究所 デジタルイノベーション技術センター ソフトウェアプロセスイノベーション部

＜主張したい点＞
　プロジェクト間での情報共有におけるコミュニケーションの齟齬を解消し、品質管理の強化と開発効率の向上を図るためには、社内における開発のプロセスや設計の粒度を統一する必要がある。東芝デジタルソリューションズ(株)では、これまで開発プロセスや設計書のフォーマットなどの設計開発標準を整備してきた。本発表では、開発プロセスや設計書のフォーマットなどの設計開発標準の整備と普及展開活動について紹介する。結果として、設計開発標準の適用率が拡大し、不具合密度の減少などソフトウェア開発における品質の改善に繋がった。

＜キーワード＞
プロセス定義、設計開発標準、ソフトウェア開発、開発効率、普及展開活動、テンプレート、メタモデル、テーラリング

＜想定する聴衆＞
システム／ソフトウェア開発の改善を推進する役割の方（SEPG/SQAG）

＜活動時期＞
2000 年に整備開始。2017 年、2020 年と段階的に弊社設計開発標準に定め、整備及び普及展開活動を現在も継続中。

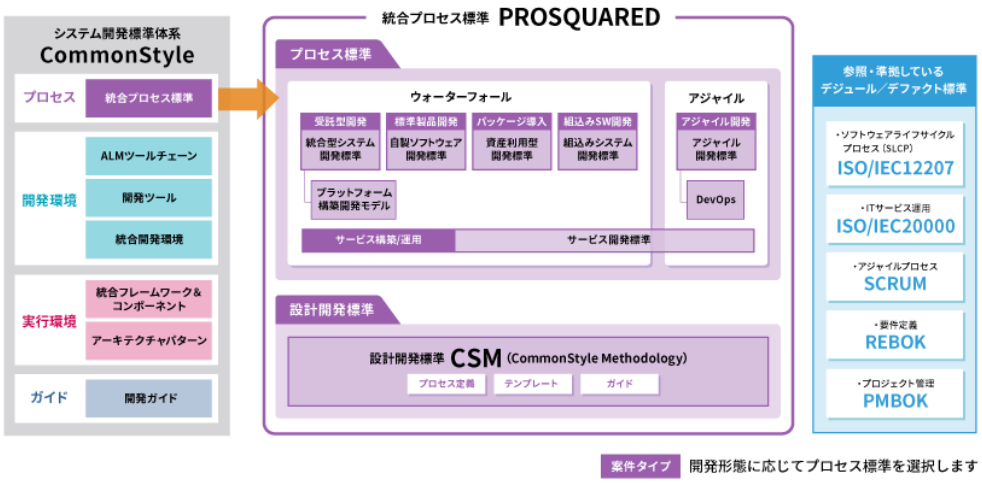
＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。
- ☐ 着想の段階(アイデア・構想の発表)
- ☐ 改善活動を実施したが、結果はまだ明確ではない段階
- ■ 改善活動の結果が明確になっている段階
- ☐ その他(　　)

＜発表内容＞
1.背景
東芝デジタルソリューションズ(株)では、ソフトウェア開発の設計書について、プロジェクトごとに独自のフォーマットを用いており、設計項目の記載場所や記載粒度がプロジェクトによって異なっていた。このため、設計情報の漏れや曖昧さが生じ、品質管理が難しくなっていた。さらに、設計書のフォーマットが統一されていないことが、プロジェクト間での情報共有や引き継ぎを困難にしていた。
また、オフショア開発拠点や協力会社との連携においても、設計書のフォーマットや内容が統一されていないため、協力会社との情報共有がスムーズに行われず、情報共有の際に誤解やミスといったコミュニケーションの齟齬が発生することがあった。このような齟齬は、プロジェクト計画の遅延や品質の問題を引き起こす原因となっていた。
ソフトウェア開発の複雑化と多様化に伴い、上記のような問題を解消するために標準化された設計書のフォーマット統一に取り組んできた。

2.改善したいこと
ソフトウェア開発において、プロジェクトごとに設計書の名称や設計項目の適用範囲に差があった。たとえば「要件定義書」に含まれる設計項目や、システム構成の記述場所・記述方法が異なるなど、プロジェクトごとに設計のばらつきが見られた。加えて、国内外のデファクト／デジュール標準が存在するにもかかわらず、プロジェクト独自の表記が用いられているケースもあり、対外的に提出する文書としての品質に課題があった。顧客ごとに要求が異なる場合もあるが、当社としての標準を明確に定義する必要があった。
このような状況では、類似プロジェクト間での設計書の流用が困難であり、他プロジェクトからメンバーをアサインした場合でも、設計書の構成や記述内容を理解するのに時間を要するという非効率が生じていた。

3.改善策を導き出した経緯
前述の課題を解消するためには、設計書の構成や記述粒度を統一し、「何を」「いつ」「どこに」検討・記述すべきかを明確にすることが重要である。これにより、設計情報の漏れや曖昧さを防止し、情報の一貫性と再利用性を高めるとともに、品質管理の強化と開発効率の向上を図ることができる。
そこで、国内外の標準的な開発手順、考え方、成果物記述方法といったデファクト／デジュール標準、および社内のソフトウェア開発に関する知見を基に、開発プロセスや設計書のフォーマットの検討を行った。社内における開発のプロセスや設計の粒度を統一することで、設計情報の漏れや曖昧さを低減し、設計情報の一貫性の確保やソフトウェア開発における品質管理の強化を目指すものとした。

4.改善策の内容
まず、開発プロセスについて、「PROSQUARED（プロスカード)」という名前で統合プロセス標準を策定した(図 1)。これは、受託型システム開発や組込みソフトウェアの開発、アジャイル開発など、システムを開発する形態ごとに開発の流れを定義（プロセス標準）するとともに、各形態のプロセス標準の設計工程で共通的に利用できるテンプレートなどを準備したものである。システムを開発する際に考慮するべき、要件や機能、画面といった事柄は、設計に落とし込むことが重要である。そのため、設計工程において、設計する順番や、どんな設計書が必要かなどを指南する「プロセス定義」、設計書や仕様書などの内容に必要となる要素や図表のフォーマット、記載例などを記した「テンプレート」、そしてシステム開発に必要な知識を解説した「ガイド」、システム開発時に考慮するべき事柄とその事柄同士の関係を定義した「メタモデル」に基づき整備した。これらを設計開発標準「CommonStyle Methodology(CSM)」として展開した(図 2)[1]。



ALM: Application Lifecycle Management，DevOps: システムの開発(Development)と運用(Operations)を組み合わせた言葉，REBOK: Requirement Engineering Body of Knowledge，PMBOK: Project Management Body of Knowledge

※東芝デジタルソリューションズでは、システム開発における開発技術の標準を「CommonStyle(コモンスタイル)」として体系化し、2006年から運用しています。

図 24　システム開発のプロセスを標準化した東芝の PROSQUARED と CSM

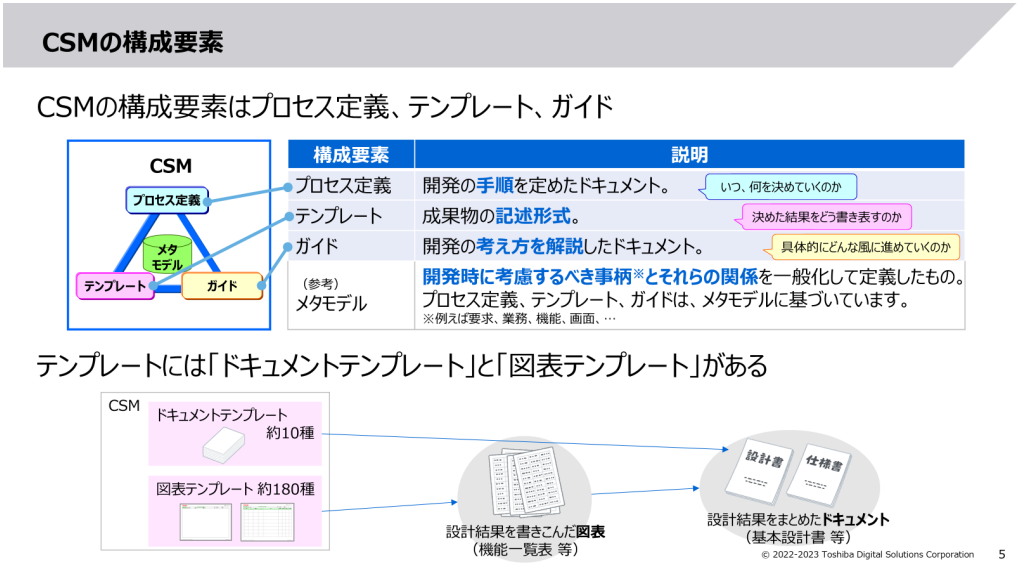


図 25　CSM の構成要素

さらに、整備した設計開発標準 CSM の成果物を、ソフトウェア開発における各工程においてどの順番で作成すればよいのかを整理して示すことで、標準的な仕事の流れを定義した。また、主要な成果物と補足的な成果物に分けることで、プロジェクトの特性に応じて設計項目のテーラリングを行うことができるようにした。

5. 改善策の実現方法

整備した CSM を現場に定着させるため、段階的かつ多面的な普及展開活動を実施している(図　3)。まず、会社の方針に CSM 適用を掲げ、半年ごとの各部門のキックオフや事業部長、技師長のメッセージで繰り返し展開した。さらに、社内に推進・改善チームを構築し、現場の代表者も参画する体制を整えた。このチームは、新規案件への適用を起点として CSM の導入を進めており、すべての対象プロジェクトに対して適用されることを目指している。

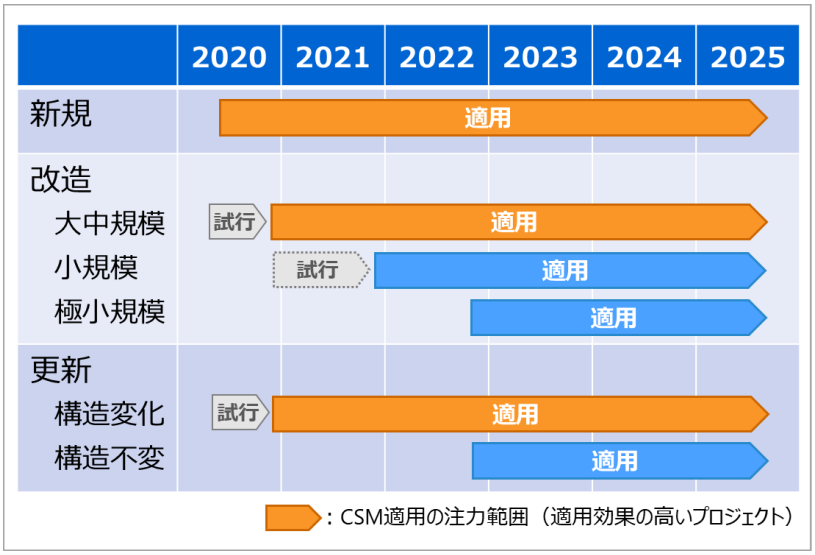


図 26　CSM の適用期間

さらに、CSM の適用状況を把握するため、全プロジェクトに対して適用の有無を申告させ、ヒアリングを実施している。設計書の使いづらさや使用方法の習得など、ヒアリングの中で出た意見についても推進・改善チームにて議論しつつ、CSM の改善を実施している。
また、協力会社やオフショア開発拠点への展開を行うため、自主学習用の教材を整備・展開するとともに、オフショア開発拠点向けの教育プログラムを実施している。オフショア開発拠点において、現地内で CSM を指導できる体制を構築するため、専任のスペシャルチームを編成した。
一方で、プロジェクトの規模によっては、すべての設計書を網羅的に作成することが困難なケースもある。そのため、テーラリング（適用範囲の調整）を支援するためのガイドを整備し、各プロジェクトが自律的に適用範囲を判断できるようにした。
CSM の改善活動については定期的に説明会を実施し、社内の展開状況や CSM の改善状況の周知を行っている。トップダウンによる方針展開と、現場に寄り添った地道な活動を両立させることで、CSM の定着と品質向上を推進している。

6. 改善による変化や効果

普及展開活動により、ソフトウェア開発プロジェクトにおける CSM の適用率は着実に向上している。図 4 に示すように、オフショア開発拠点では、CSM の適用率が 2020 年から 2023 年までの 3 年間で約 2.4 倍に増加した。特に新規案件を中心に、標準化されたプロセスや設計書フォーマットの活用が進み、プロジェクト間での設計情報の一貫性と再利用性が高まっている。

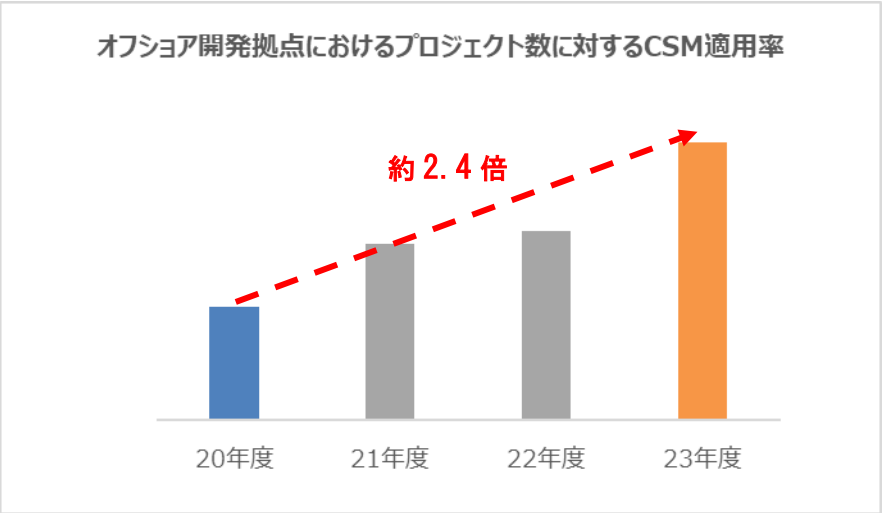


図 27　オフショア開発拠点における CSM 適用率の推移

CSM 活用については現場からはポジティブな反応が得られている。たとえば、オフショア開発拠点からは、「CSM の設計書は構造が明確で理解しやすい」「見積齟齬が減少した」との声が寄せられている。定量的な評価を見ても、図 5 に示すように従来の自由記述型の文書に比べて、新規案件におけるオフショア拠点からの受け入れ後の不具合密度の平均値が-36.8%、中央値が-59.3%と、ともに CSM を適用した方が削減傾向にある。

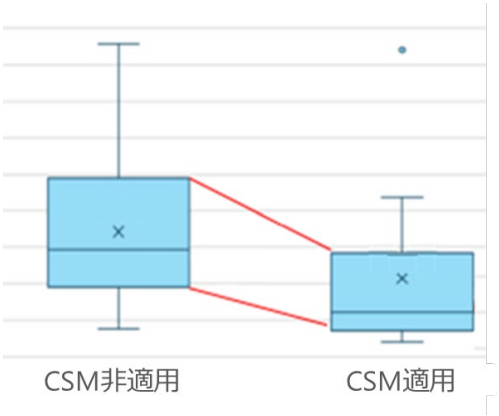


図 28　オフショア拠点における、新規案件に対する CSM 非適用プロジェクトと適用プロジェクトの不具合密度の比較

7. 改善活動の妥当性確認

CSM の導入および普及活動においては、利用者の声を継続的に収集・分析している。活動初期には、CSM の採用に対する懐疑的な意見や、従来の開発手法への固執、CSM 自体の認知度の低さといった課題が見られた。しかし、現場代表者を含めた推進・改善チームの体制整備と、推進・改善チームによる地道な説明活動を通じて、CSM の有用性が徐々に浸透し、図 3 の分類における CSM 適用の注力範囲である「新規」「改造（大中規模）」「更新（構造変化）」において、適用率が 81.3%に上がるなど、「CSM を使うのが当たり前」という認識が定着しつつある(図 6)。

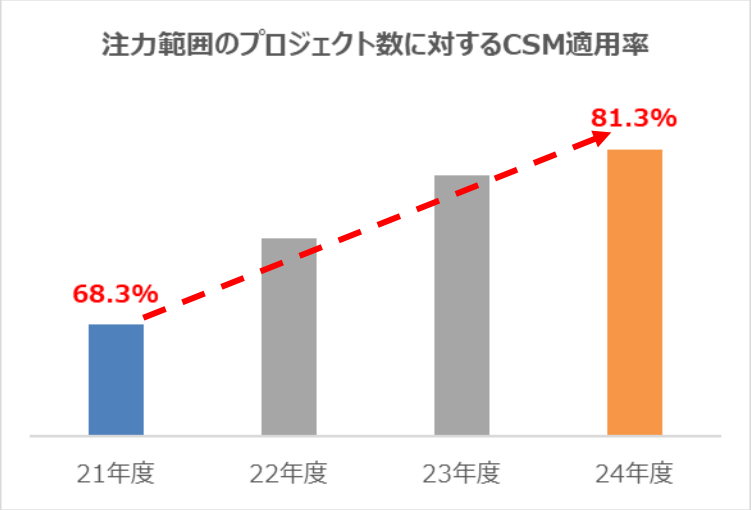


図 29　注力範囲に対する CSM 適用率

また、CSM の適用方針説明会では、標準化の必要性に対する賛同の声が多く寄せられた。特に、開発現場の高齢化が進む中で、属人性を排除し、設計情報を形式知として蓄積・共有できる CSM の仕組みは、今後の知識継承や教育の観点からも有効であると考えられる。
さらに、CSM の適用を前提として、生成 AI などの支援ツールの活用も進められており、設計書の自動生成や文書チェックなど、次のステップに向けた取り組みがすでに開始されている[1]。これにより、CSM は標準化の枠を超え、開発効率と品質の両立を実現するための基盤として、今後さらに発展させていくものである。今後も現場の声を取り入れながら、継続的な改善と活用拡大に取り組んでいく。

参考情報
[1] プロセス標準と生成 AI の融合で開くソフトウェア開発の新たな境地 | DiGiTAL T-SOUL | 東芝デジタルソリューションズ

3A3　場当たり的改善からの脱却：IDEAL モデル×Four Keys による科学的プロセス改善　～　Four Keys を組織に定着させる体系的フレームワーク　～　高橋裕之（ファインディ株式会社）

＜タイトル＞

場当たり的改善からの脱却：IDEAL モデル×Four Keys による科学的プロセス改善

＜サブタイトル＞

～　Four Keys を組織に定着させる体系的フレームワーク　～

＜発表者＞

氏名(ふりがな)：高橋裕之（たかはしひろゆき）

所属：　ファインディ株式会社

＜共同執筆者＞

氏名(ふりがな)：

所属：

＜主張したい点＞

多くの組織でプロセス改善が場当たり的になる根本原因は「体系的フレームワークを知らない」ことにあると考えている。本発表では、科学的測定（Four Keys、または DORA Metrics）と体系的改善（IDEAL モデル）を統合した実践的フレームワークを提示する。IDEAL モデルの 5 段階構造を現代的に再解釈し、各フェーズでの科学的測定活用法とケイパビリティアセスメントを具体化することで、「明日から実践できる改善手法」を提供する。聴衆は、自組織の現状診断から改善計画策定、効果測定まで一貫したアプローチを習得でき、技術投資の ROI 可視化と経営層への説明責任を果たす手法を持ち帰ることができる。

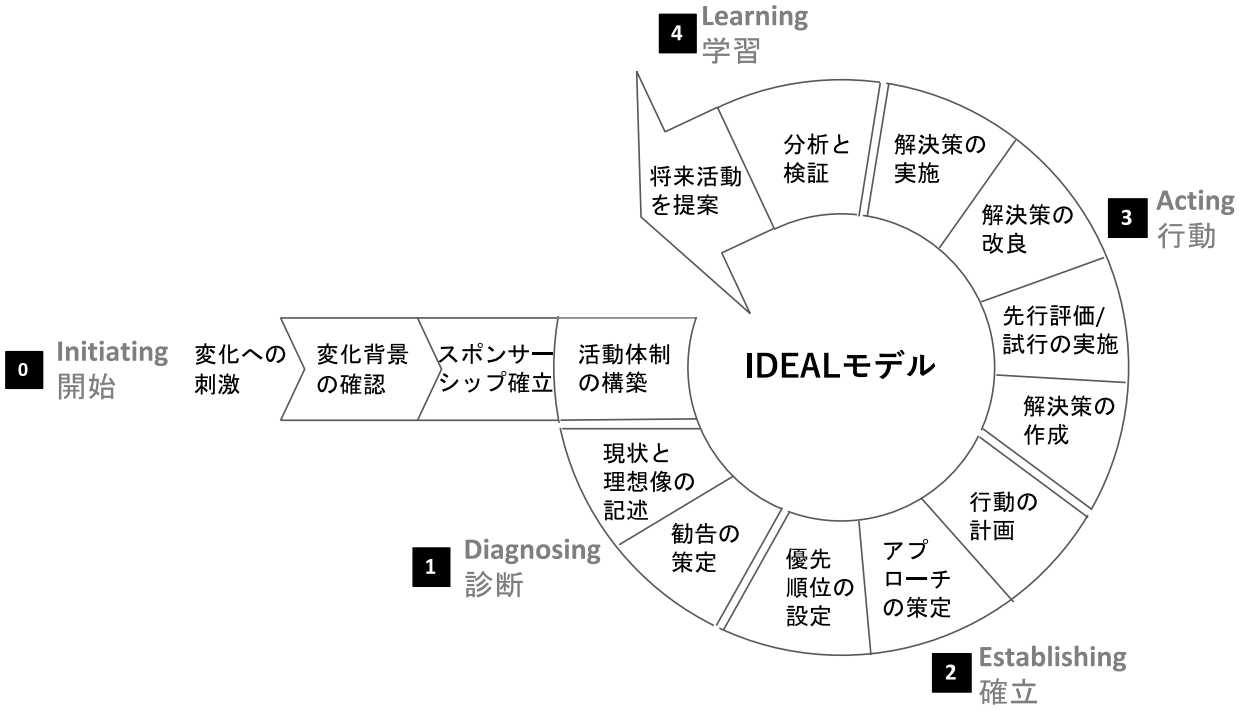


図 1 IDEAL モデル

＜キーワード＞

プロセス改善、Four Keys、DORA Metrics、IDEAL モデル、アジャイル開発、品質保証、組織変革、科学的測定、DevOps

＜想定する聴衆＞

- ソフトウェア開発組織でプロセス改善に取り組む管理者・リーダー
- DevOps/アジャイル導入を検討中または実践中の組織
- 技術的改善と事業成果の関連性を明確化したい実践者
- 従来の SPI アプローチに課題を感じている改善推進者
- 生成 AI 導入により開発プロセス変革に取り組む組織・個人

＜活動時期＞

2020 年～現在（継続中）

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)

□ 改善活動を実施したが、結果はまだ明確ではない段階

☑ 改善活動の結果が明確になっている段階

□ その他(　　)

3A4　システム開発・サービス提供における原理原則の標準化　三角英治（株式会社 NTT データグループ）

＜タイトル＞

システム開発・サービス提供における原理原則の標準化

＜サブタイトル＞

なし

＜発表者＞

氏名(ふりがな)：三角　英治（みすみ　えいじ）
所属：　ＮＴＴデータグループ　技術革新統括本部　品質保証部

＜共同執筆者＞

氏名(ふりがな)：　なし
所属：　なし

＜主張したい点＞

- システム開発・サービス提供の各フェーズにおける再発防止・予防を全社的に徹底するため、過去に発生した重大システム故障と不採算案件から得られた知見・教訓を「原理原則」として標準化した
- 「原理原則」を全社の品質マネジメントシステムに組み込み、展開・定着を図っている

＜キーワード＞

再発防止，システム故障，プロジェクト問題化，品質マネジメントシステム，標準化

＜想定する聴衆＞

システム故障やプロジェクト問題化の原因に対して、その再発防止・予防を全社的な仕組みの中で徹底しようと考えている方

＜活動時期＞

FY2024 ～ FY2025（取組中）

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
■ 改善活動を実施したが、結果はまだ明確ではない段階　※一部の結果は見えてくる段階
□ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1.背景

昨今発生した重大システム故障と不採算案件の原因を分析すると、過去にも類似の原因で問題が発生していることが明らかになった。

これまでも重大システム故障、不採算案件が発生した際には、都度再発防止策を検討、実行してきた。しかし、これらの再発防止策は社内に展開されるものの、時間の経過とともに効果が失われていく傾向があった。そのため、他組織において類似の原因による重大システム故障、不採算案件が発生していた。

本稿では、過去事例に対する再発防止策の全社的な徹底に関する課題、システム開発・サービス提供における原理原則（以下、「原理原則」とする）の標準化プロセス、およびその内容について論述し、今後の課題を考察する。

2.改善したいこと

これまでも、重大システム故障と不採算案件が発生する度に、再発防止策が講じられ、それらの再発防止策は社内に展開されてきた。しかし、組織を跨いだ全社的な徹底は十分ではなく、他組織において類似の原因による重大システム故障、不採算案件が発生していることが、分析を通じて明らかになった。そこで、再発防止策を全社的に徹底させるため、過去に発生した問題から得られた知見・教訓を定着させることが課題であった。なぜなら、現状、前述の再発防止策をはじめとした社内のナレッジは点在しており、適時適切な利用が難しい状態であった。そのため、再発防止・予防に資する知見・教訓を全社的に標準化することで、その問題が解決されると考えた(飯塚ら,2021)(飯塚,2023)。

3.改善策を導き出した経緯

品質保証部として全社の品質マネジメントシステム（QMS）を担っている立場から、再発防止・予防を仕組みとして取り込むことを考えた。そこで、過去に発生した重大システム故障と不採算案件から得られた知見・教訓を標準化するため、下記の 3 つのプロセスを実施することにした（図 1）。

- 社内に蓄積されたナレッジをインプットとして活用
- インプットから再発防止・予防に資する共通性の高い内容を抽出・分類
- 抽出・分類した内容からから汎用性の高い知見・教訓を考案

各プロセスの詳細を 4.1 から 4.3 に示す。

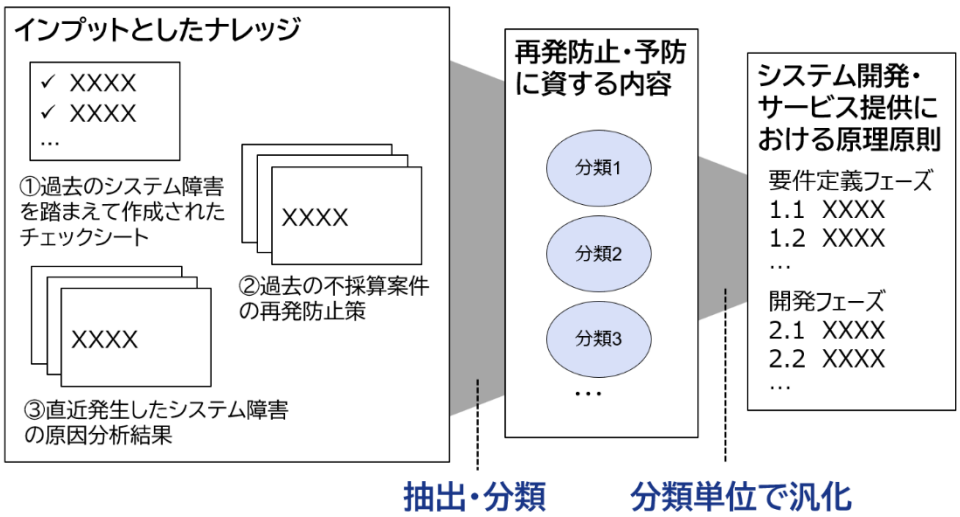


図 1：原理原則の作成プロセス

4.改善策の内容

再発防止・予防に資する汎用的な内容を各開発/運用フェーズにおける原理原則として明らかにした。

4.1.インプットとして採用した情報

インプットの候補として、社内に蓄積された様々なナレッジに着目した。数多く存在するナレッジの中から、重要性の高さを考慮し、下記の 3 つをインプットとして採用した。

- 過去のシステム障害を踏まえて作成されたチェックシート
- 過去の不採算案件の再発防止策
- 直近発生したシステム障害の原因分析結果

4.2.再発防止・予防に資する内容の抽出・分類

上述のインプットの中から、再発防止・予防に資する内容に焦点を当て、約 150 件の情報を抽出した。次に、それらの情報を下記の 5 つの観点で約 80 件に絞り込んだ。

- プロジェクト固有の性質に依らず共通性が高い
- 特定のケースに限定されない
- 細かすぎず、かつ具体的すぎない
- 現行の全社ルールに記載されていない
- 個々のシステム開発・サービス提供プロジェクトによる実施事項である

さらに、それらの情報を類似性の高さで分類した。

4.3.汎用性の高い知見・教訓の考案

前述の方法で得られた情報を、プロジェクトの特性に依らない共通かつ有益な知見へ汎化し、要件定義、開発、移行、サービス提供といったシステム開発・サービス提供プロジェクトのフェーズ別に整理し、それぞれのフェーズで徹底すべき事項として明らかにした（図 2）。汎化された 31 件の知見・教訓を原理原則と呼称し、形式知化した。

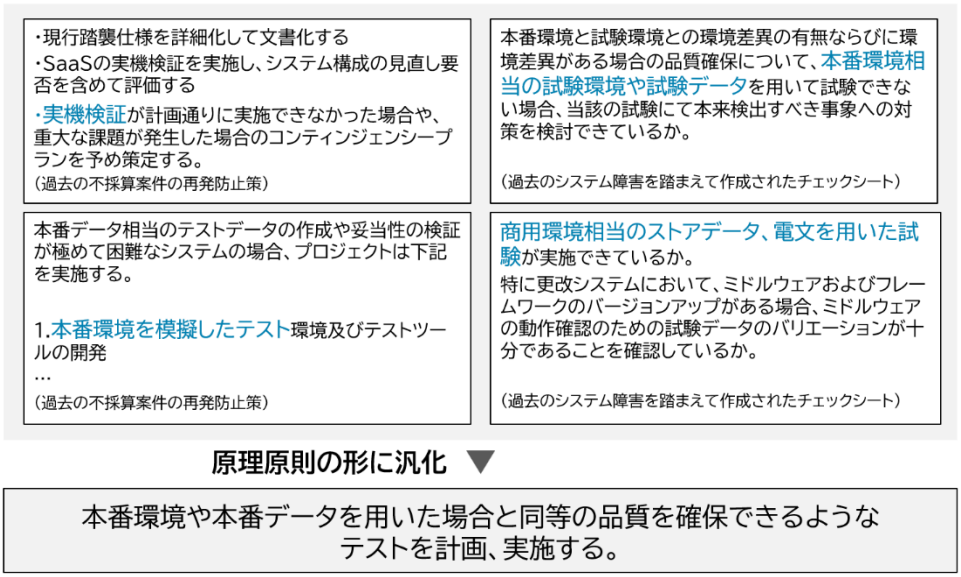


図 2：再発防止・予防に資する情報から作成された原理原則の一例

5.改善策の実現方法

形式知化した原理原則を社内標準化するため、原理原則を記載した全社ルールの改正案を作成した。その上で、品質マネジメントシステムの運営に関わる各事業部門の代表者をはじめとする利害関係者に対して、原理原則を記載した全社ルール改正案に関する意見照会を実施した。意見照会を通して、第三者による原理原則の妥当性評価を受け、評価結果を踏まえた対応を行った。
以上の PDCA サイクルを回すことで、システム開発・サービス提供プロジェクトの各フェーズにおいて徹底すべき原理原則を全社ルールとして標準化した。

6.改善による変化や効果

全社ルールとしての標準化に続いて、下記の取組みを実施している。
(1)現場への浸透・定着
実際のシステム開発・サービス提供を行う現場への浸透・定着のためには、現場の要員が原理原則を認識し、それらの必要性を理解、実践する必要がある(飯塚,2014)。そのための取組みとして、全社ルールを補足する、実践的・具体的なナレッジを原理原則ガイドラインに集約し、展開している。当該ナレッジでは、原理原則のソースとなった事例における問題や原因を交えて、原理原則の必要性を解説することで、現場への浸透・定着を図っている。また、当該ナレッジは社内セミナー、ニュースレター、ポータルサイト等の複数チャネルを通して情報発信し、現場に直接情報を届けている。
(2)浸透状況および効果の確認
原理原則が浸透・定着し、徹底されているか、また有効性を確認することは、継続的な改善に向けて重要なプロセスである。そこで、QMS 内部監査によって各プロジェクトにおける原理原則に対する取組み状況を確認することとしている。原理原則に関する監査項目を設け、インタビューを通して具体的な取組み状況や改善点といった情報を収集することで、現場への浸透状況や有効性の確認及び継続的な改善を図っている。

7.改善活動の妥当性確認

本稿では、再発防止策を全社的に徹底させるため、社内に蓄積された様々なナレッジのうち重要性が高い情報をインプットとし、再発防止・予防に資する内容を抽出、分類、汎化した。そこから得られた共通的な知見・教訓を、システム開発・サービス提供プロジェクトの各フェーズで徹底すべき原理原則として明らかにした。加えて、利害関係者による意見照会を通じて、原理原則の妥当性を客観的に評価した。
引き続き、原理原則の現場への浸透・定着を図りつつ、内部監査等を通して浸透状況及び再発防止・予防に対する効果の確認を進めていく。

参考情報

飯塚悦功，金子雅明，住本守，山上裕司，丸山昇(2014)「進化する品質経営-事業の持続的成功を目指して」株式会社日科技連出版社
飯塚悦功，金子雅明，平林良人(2021)「TQM みんなの“ 大誤解”を斬る！顧客満足は正義なのか？」株式会社日科技連出版社
飯塚悦功(2023)「マネジメントシステムに魂を入れる」株式会社日科技連出版社

3B1 シフトレフトで組織を「つなぐ」プロセス変革　テストプロセスの平準化で残業を減らし安定した品質をつくりこむアジャイル

小坂淳貴（KDDI アジャイル開発センター株式会社）

＜タイトル＞

シフトレフトで組織を「つなぐ」プロセス変革

＜サブタイトル＞

テストプロセスの平準化で残業を減らし安定した品質をつくりこむアジャイル

＜発表者＞

氏名(ふりがな)：小坂　淳貴(こさか　じゅんき)
所属：　KDDI アジャイル開発センター株式会社

＜共同執筆者＞

氏名(ふりがな)：
所属：

＜主張したい点＞

- スクラムを採用していてもプロセスが不適切だと社内でも受発注のような関係にしかならないこと
  - 適切なプロセスを描けないと品質担当者が休暇すら取れない状況に陥る
- アジャイルにおいても V 字モデルのような考え方は大切であること
  - コミュニケーションの仕組みや頻度についてアジャイルを踏襲し、組織をアップデートする

＜キーワード＞

シフトレフト/アジャイル/テスト/プロセス

＜想定する聴衆＞

- アジャイル・スクラムで品質を作り込むことが難しそうだと考えている人
- アジャイル・スクラムに挑戦しているが適切な開発プロセスが設計できないと感じている人

＜活動時期＞

2025 年 1 月～現在

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
□ 改善活動を実施したが、結果はまだ明確ではない段階
□ 改善活動の結果が明確になっている段階
☐ その他(改善の結果が見えているが、継続的に改善を進めている状態)

＜発表内容＞

1. 参画したプロジェクトが以下の状態だった
   - アジャイルは取り組んでいる
   - スクラムを採用しているが、スクラムを実践したい人が誰もいない
   - リリース日が決められていて、期日までに開発を間に合わせる受身型開発
   - 与えられた期日を目指して開発する以上のマネジメントがなく、開発プロセスがない
2. 開発した成果物に対してテストをする人たちがいたが、非常に負荷の高い状態だった
   - 作業着手のタイミングでは仕様の詳細が決まっておらず、テスト仕様の作成が出来ない
   - 開発後期に仕様が確定し、リリース規模によっては膨大な作業量が発生する
   - リリースのためのテストをする別部隊もいるが、どんなテストをするのかコミュニケーションなされていない
3. 安定した品質で開発を継続するために、プロセス改善を実施
   - スクラムのリズムに追いついていない状態の是正
     - 受け入れ条件の設定を早期化するための企画とのコミュニケーション変革
       - 繁忙で時間が取れないビジネス側の業務を減らし成果にフォーカスできる状態へ
     - 開発とテスト担当者のコラボレーション促進
     - テスト作業ピークの平準化
       - 受け身でテスト作業を行うリズムのシフトレフト化
     - V 字モデルとスクラム運用
   - (スクラムの経験がないことに伴う既存の知識・経験をスクラムにそのまま当てはめるリスク)
4. プロセスだけでは不具合を生み出す
   - 開発力不足に伴う不具合
     - リファクタリングによるバグ混入(論理反転など)
     - ベストを尽くしても結果的に質より量に陥りマンパワーでこなしてもすり抜ける不具合
   - 仕組みのなさに伴う(人的)不具合
     - ブランチ戦略不足によるデグレ
     - コミュニケーション不足による作業のコンフリクト
5. 安定した品質を作り込むためにアジャイルプラクティスを活用する
   - Extreme Programming
   - テストピラミッド
   - アジャイルテストの 4 象限
6. 安定した開発を継続するには関係者全体をつなぐプロセスと適切な知識が重要
   - 短いウォーターフォールに見立てて反復的な開発を行うことは無理が生じやすい
   - 人と人をつなぐことと、既存の知識と新しい知識をつなぐことで価値の高い活動を行える組織へ

参考情報

[1] シフトレフト https://service.shiftinc.jp/column/10905/
[2] スクラムガイド 2020 https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Japanese.pdf
[3] エクストリームプログラミング http://www.extremeprogramming.org/

3B2 デグレ防止の自動テストと即時リリースの取り組み 中村幸太（株式会社パラミックス）

＜タイトル＞

デグレ防止の自動テストと即時リリースの取り組み

＜サブタイトル＞

＜発表者＞

氏名(ふりがな)：中村　幸太（なかむら　こうた）
所属：　株式会社パラミックス

＜共同執筆者＞

氏名(ふりがな)：　中村　伸裕（なかむら　のぶひろ）
所属：　住友電工情報システム株式会社　QCD 改善推進部

＜主張したい点＞

(1) 自動テストはデグレ防止に効果的であることは一般的に知られているが、自動テストの作成及び保守の実践が難しい。今回の取り組みでは、テストフレームワーク Jest に独自のライブラリを追加し効率的に自動テストが作成できるようにした。また、自動テストは GitLab のパイプラインに組み込むことでエラーが可視化され、常に最新の状態が保たれるようになった。
(2) 24 時間いつでもリリースは、当組織のシステムでは不可能と思われていたが、工夫することで実現できた。

＜キーワード＞

継続的デリバリー、テスト自動化、静的解析ツール、Blue-Green Deployment、GitLab

＜想定する聴衆＞

継続的デリバリー、自動テストの導入を検討している人

＜活動時期＞

2024 年 10 月 ～

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

□ 着想の段階(アイデア・構想の発表)
■ 改善活動を実施したが、結果はまだ明確ではない段階
■ 改善活動の結果が明確になっている段階
□ その他(　　)

＜発表内容＞

1.背景

私の所属している開発チームで開発しているシステムは、ソースのコード量が年々増加しており、機能追加や改修のたびに既存機能へのデグレが発生している。そのため、検出と修正に工数を要しており、新機能の開発やユーザ要望の改善に使える時間が減少している。また、不具合を発見して修正しても定時後にしかリリースできない等の問題も抱えており、結果として利用者満足度の低下につながっている。

2.改善したいこと

(1) デグレの未然防止

新規機能追加時のデグレをリリース前に検出し、7 割削減する。

(2) 即時リリース

従来定時後にしかリリースできなかったのを、24 時間いつでもリリースできるようにする。

(3) 保守性の向上

バグ修正および新規機能追加の工数を 3 割削減する。

3.改善策を導き出した経緯

SPI Japan 2024 で行われた招待講演『組織に自動テストを書く文化を根付かせる戦略 2024 秋』[1]を聴講したことが、本取り組みの出発点となった。特に印象的だったのは、継続的デリバリーの能力指標として 4 つのキーメトリクス（リードタイム、デプロイ頻度、平均修復時間、変更失敗率）を重視するという考え方である。これらの指標は、まさに私たちのチームが抱えている課題（リリースの遅延、修正対応の長期化、デプロイ後の不具合）と一致していたため、継続的デリバリーと自動テストの導入は、有効な改善策であると認識を持つに至った。

この講演を受けて、継続的デリバリーの実践的な取り組みをより深く理解するために、書籍『継続的デリバリー』[2]を読了。同書では、継続的デリバリーを効果的に運用するための自動化の重要性について解説されており、改善を進めるうえでの参考になる知見が多く得られた。これらをもとに、目標と具体的施策の関係性を整理した「目標施策関連図」を作成した。

図 1 は、利用者に対する目標施策関連図で「利用者がいつでも快適に使えること」をビジネスゴールとして掲げ、その実現に向けた施策を整理している。本発表の冒頭で述べた「改善したいこと」では、デグレの未然防止、即時リリース、保守性の向上という三つの課題を挙げているが、図 1 ではこのうち特にデグレの未然防止と即時リリースに関わる技術的な施策を中心に構成している。図 1 には「開発したソースにほとんどバグがない」ことを目指す改善策も含まれているが、本発表ではそちらには触れない。

図 2 は、開発者に対する目標施策関連図で「保守効率の向上」に向けた施策を整理している。この図は、JIS X 0129 における品質特性のうち「保守性」に該当する 4 つの副特性 ― 「解析性」「変更性」「安定性」「試験性」 ― をベースに構成されている。それぞれの副特性に対して、実際の課題から改善の方向性を明示したものである。本発表では、特に「解析性」と「試験性」に関する取り組みに焦点を当てており、「変更性」と「安定性」については対象外としている。これらは今後の改善余地として、引き続き検討していく予定である。

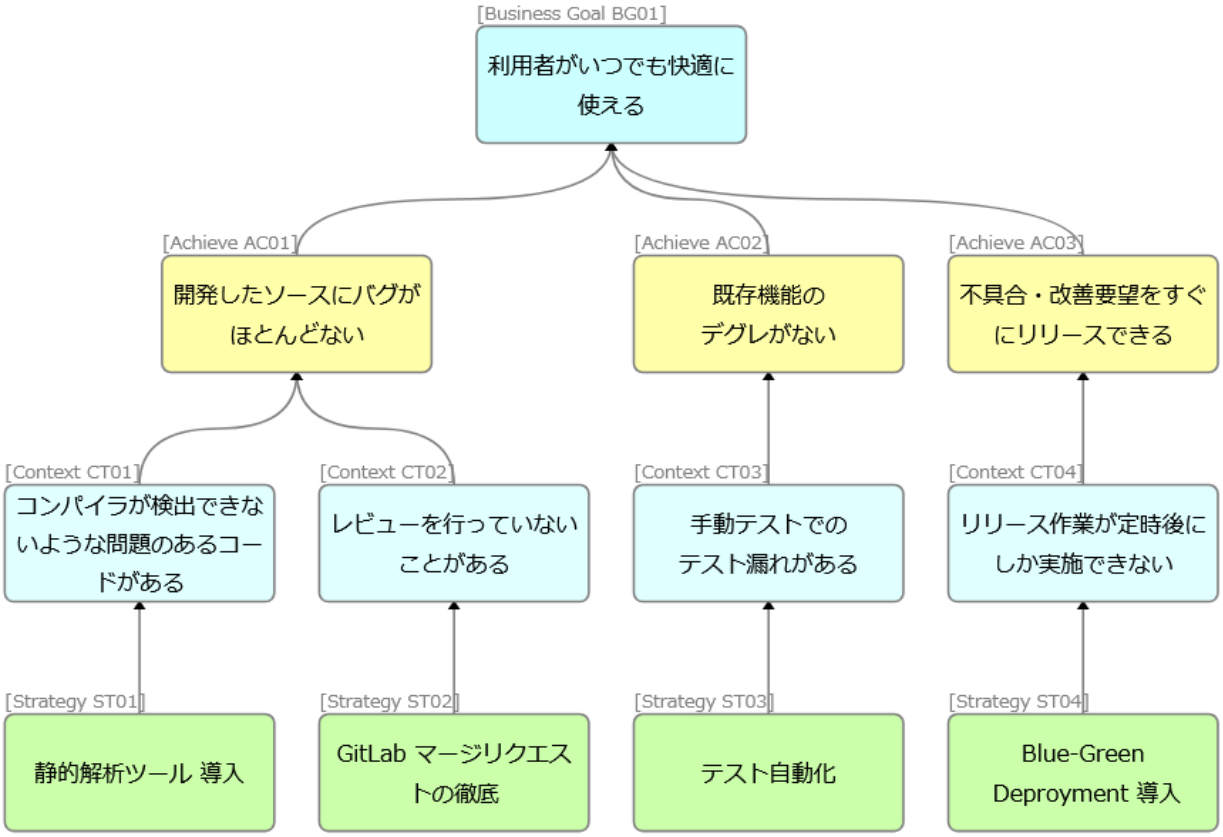


図１.利用者に対する目標施策関連図

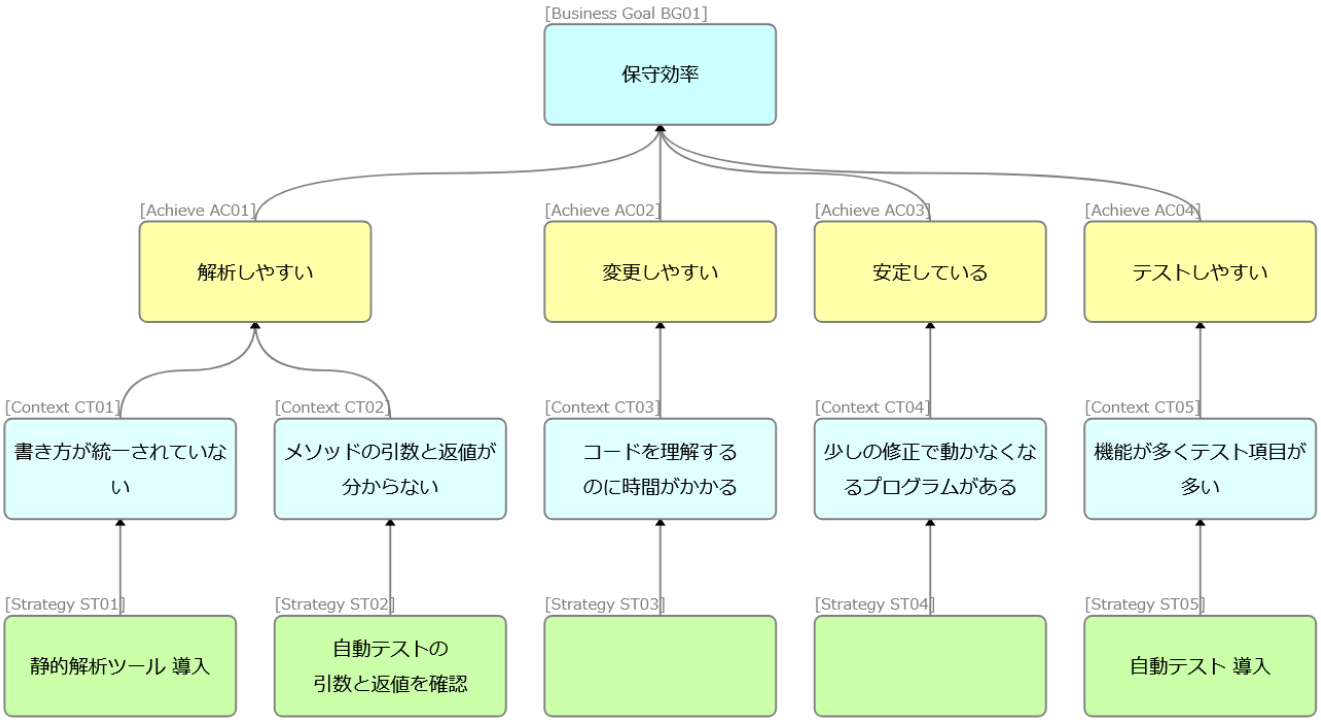


図２.開発者に対する目標施策関連図

4.改善策の内容

4.1 自動テストの導入

当初は、リリース後の既存機能へのデグレ対策として、Selenium を用いた E2E テストの仕組みを導入したが、バグの早期検知にもつながらず、十分な効果が得られていなかった。また、自動テストの品質が測定できていなかった。

そこで、自動テストの品質指標としてコードカバレッジ（Coverage）を採用することにした。従来の Selenium の自動テストは JavaScript の Coverage が測定困難であったため、Coverage が計測可能な Jest によるユニットテストへの移行を決定した。現在、段階的に自動テストの整備を進めており、全体の Coverage は約 65%に達している。現在の Coverage 状況を図 3 に示す。図 3 は SonarQube の Tree Map 表示を用いて自動テストの Coverage を視覚化している。Coverage の低い領域は赤、高い領域は緑で示されており、コード全体のテスト状況を色の変化によって直感的に把握できるようになっている。

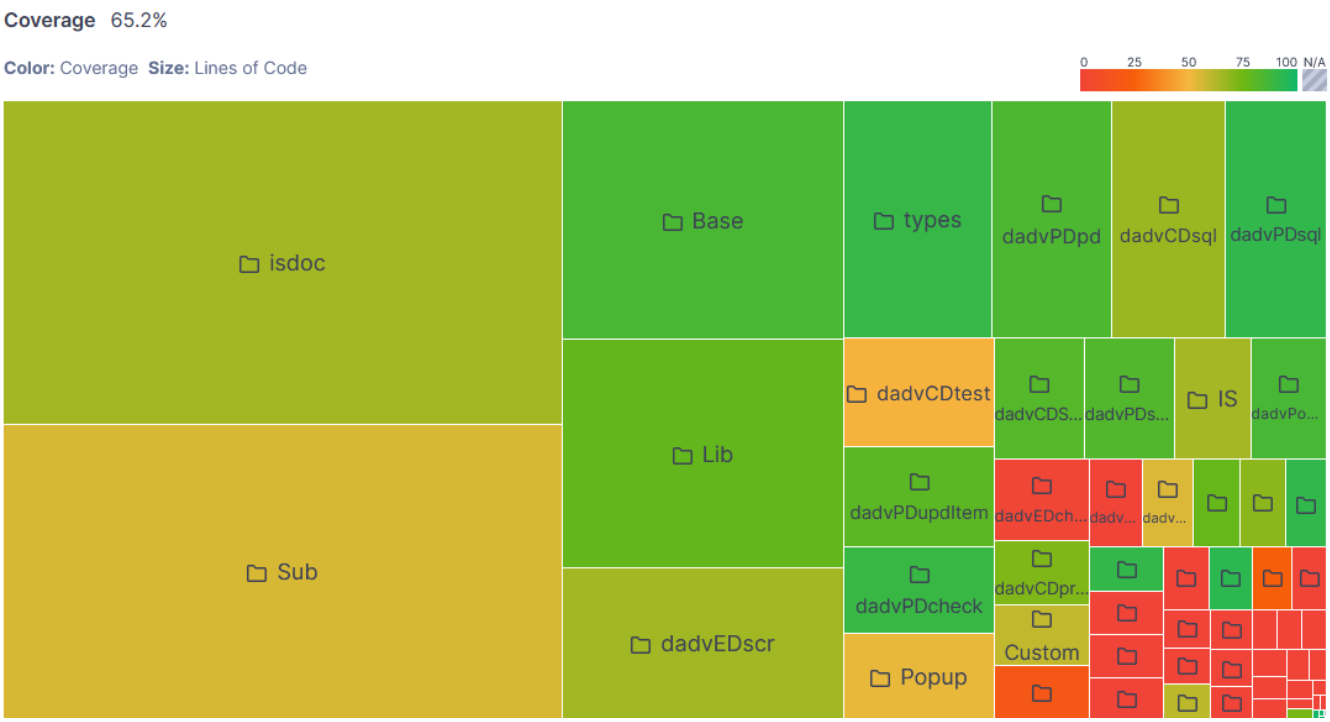


図 3 .自動テスト導入後のカバレッジ

4.2 Blue-Green Deployment の導入

(1) システム構成

このシステムは Client & Server で構成されている。クライアントはブラウザ上で実行される JavaScript によって動作し、サーバーは Node.js 上で JavaScript により処理を行う。システムの構成を図 4 に示す。ユーザがシステムにアクセスすると、以下のような流れで処理が行われる。

- ユーザがブラウザでアクセスすると HTML / CSS / JavaScript などの静的ファイルがダウンロード(①)され、画面が表示される。
- ユーザがデータを登録すると API (②)を通じてデータベースに登録される。
- 使用する API の URL（ポートを含む）は JavaScript に埋め込まれている。

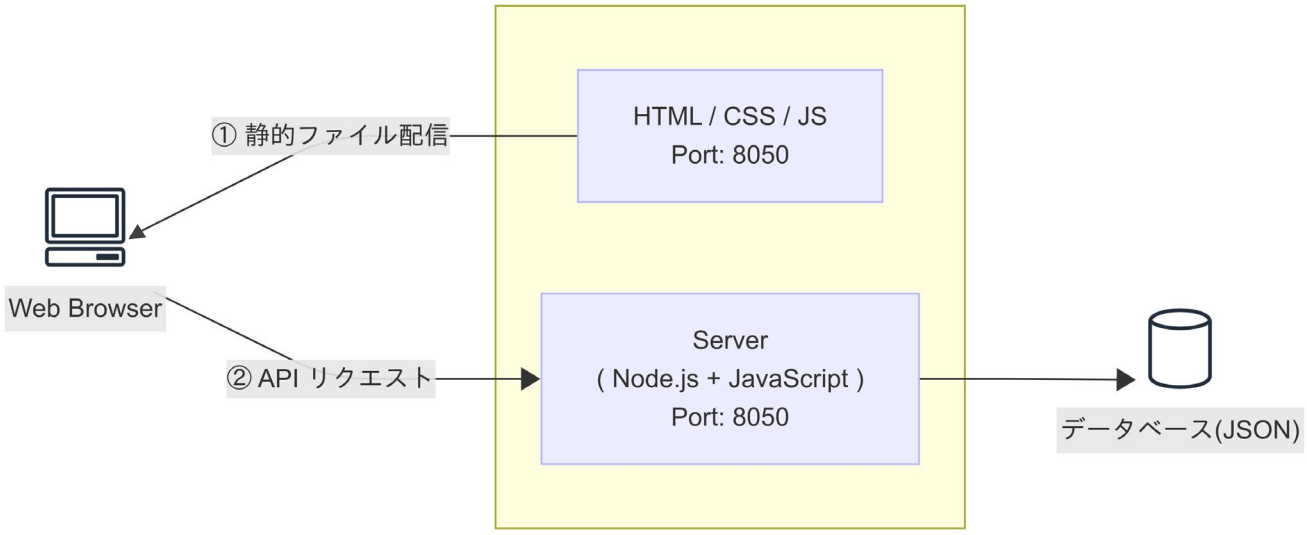


図４．従来の構成図

(2) 従来のリリース方法と課題

従来のリリース方法を図５に示す。

① 静的ファイル（ HTML / CSS / JavaScript ）をデプロイする。

② Server 側 のコードをデプロイし、 Node.js を再起動する。

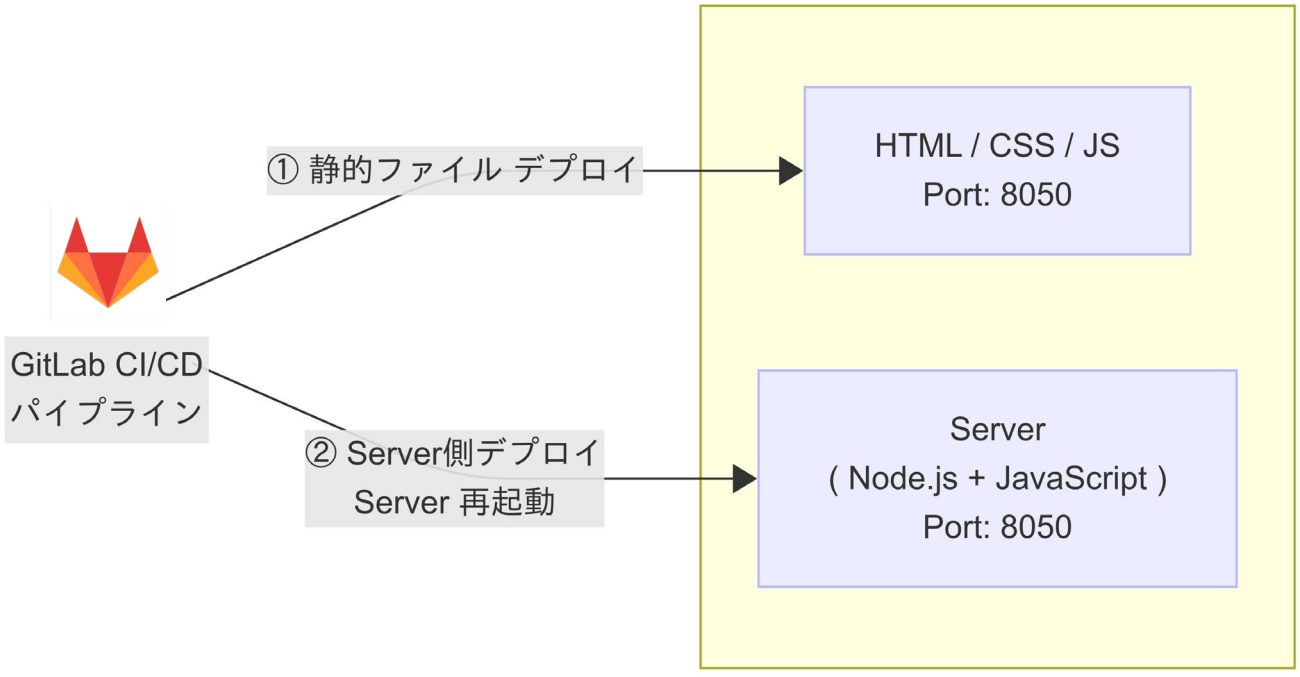


図５．従来のリリース方法

Node.js Server は固定ポート（例：8050 番）で稼働していたため、リリース時には稼働中のプロセスを再起動する必要がある。このため、業務時間中にリリースすることができず、定時後に限定してリリース作業を行う必要がある。

(3) Blue-Green Deployment 導入後のシステム構成

Blue-Green Deployment とは、システムの異なる 2 つの環境（Blue 環境と Green 環境）を用意し、稼働中（Blue 環境）のサービスを稼働させながら、もう一方（Green 環境）に新しいバージョンをデプロイする手法です。これにより、サービスを中断することなく新バージョンへのスムーズな切り替えが可能になる方式である。

本システムでは、24 時間いつでもサービスを停止せずにリリースを行えるようにするため、Blue-Green Deployment をベースとした構成を導入した。それぞれの環境が複数ポート （例：8050～8059） のいずれかで待機しており、新しいバージョンは空いているポートに配置される。デプロイ中でも既存のバージョンは継続して稼働しているため、サービスが中断なしに利用できる。

この方式では、切り替え直後の一時的な状態として旧バージョン（Blue 環境）と新バージョン（Green 環境）が並行して利用されることがある。既に画面を開いていたユーザは旧バージョンを使用し続け、メニュー操作のタイミングで新バージョンへ切り替わる運用となっている。旧バージョンを使用している状態を図 6 に、新バージョンへ切り替わった後の状態を図 7 に示す。

なお、ユーザ同士が同じデータに同時アクセスするケースはほとんどないため、データベース（JSON）における競合が発生する可能性は低いと考えている。

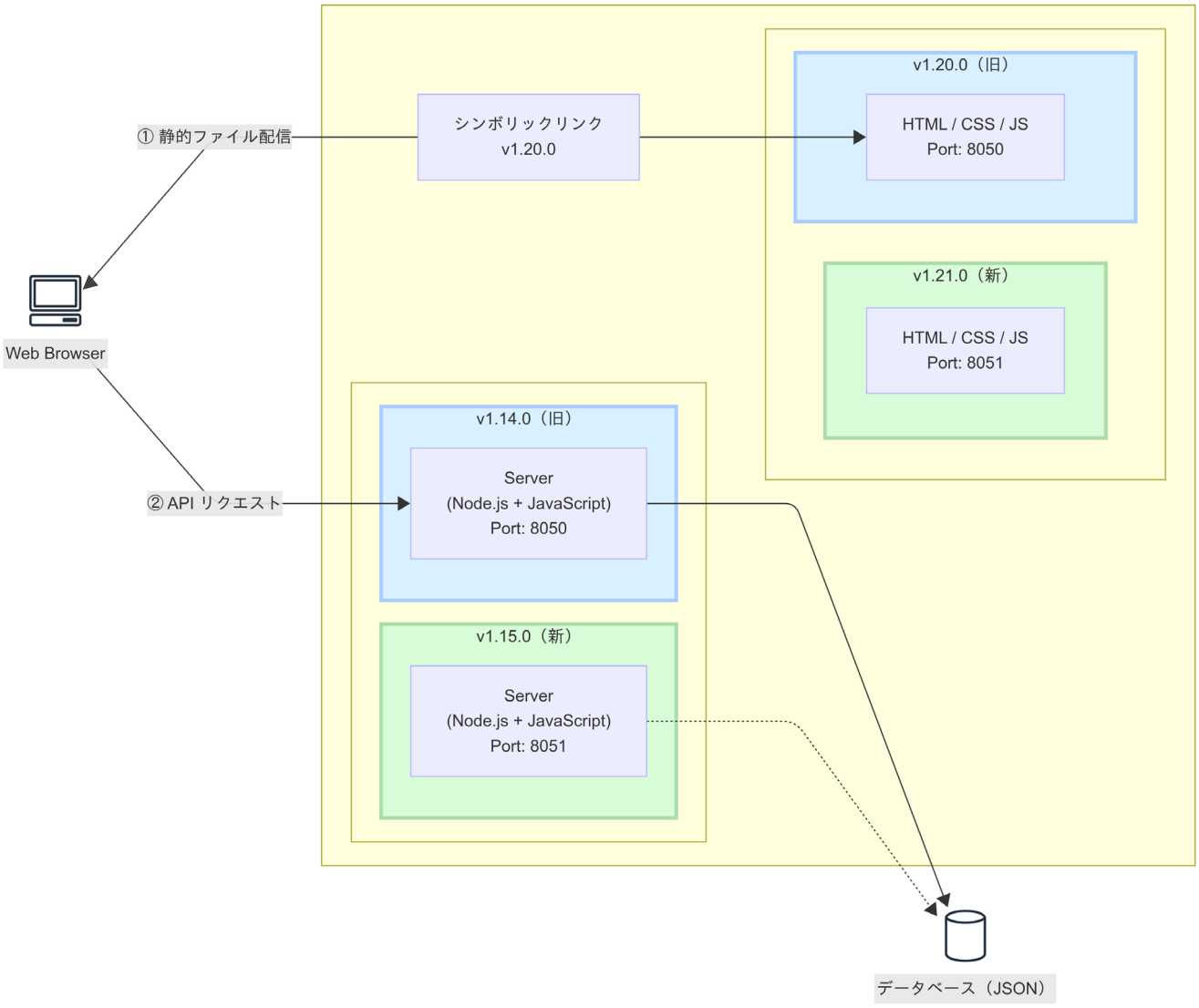


図 6．旧バージョン使用時の状態

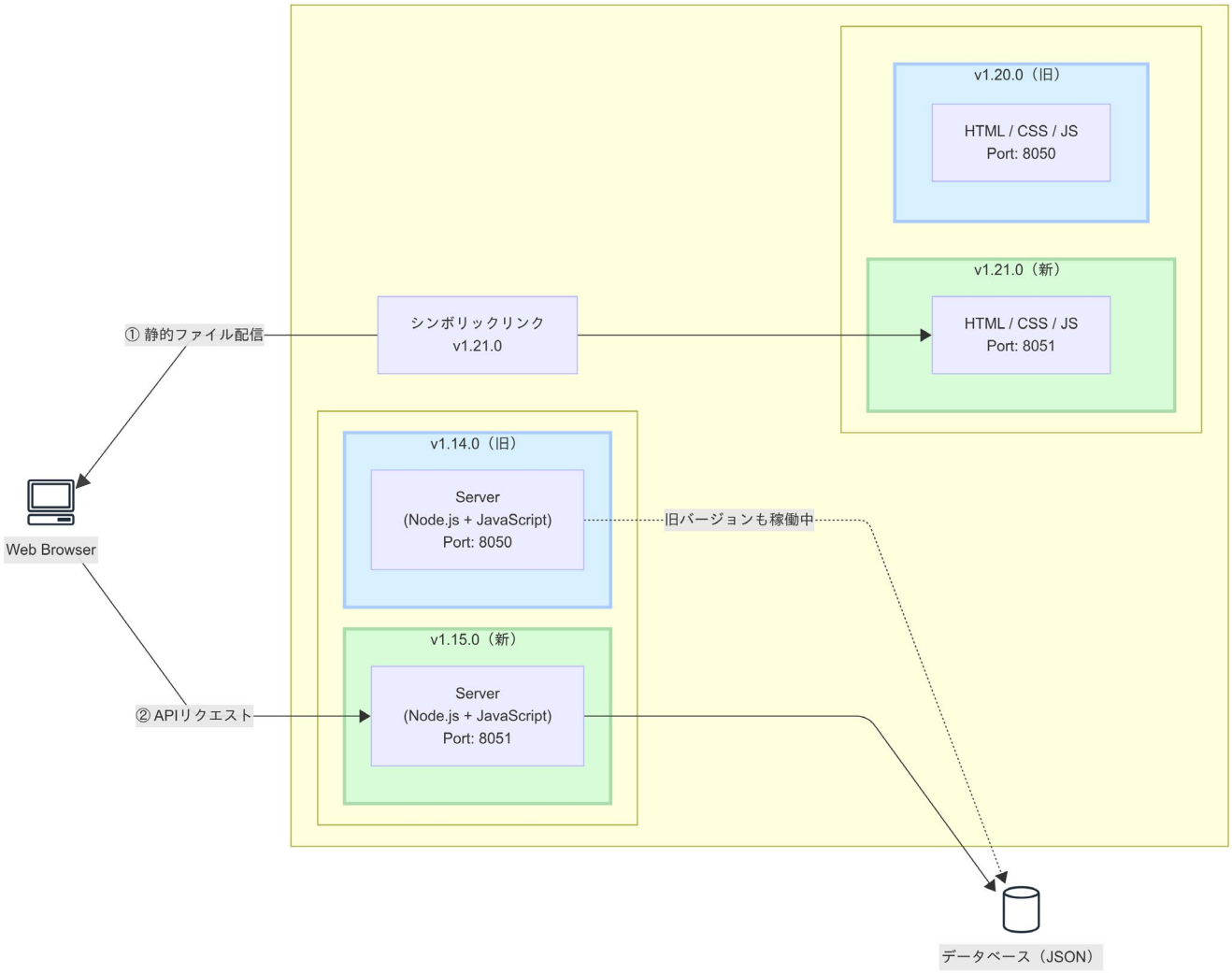


図 7．新バージョン切り替え後の状態

(4) Blue-Green Deployment　導入後のリリース方法

Blue-Green Deployment の導入後の、リリース手順を図 8 に示す。

① Server 側のコードを新バージョンのディレクトリに配置し、Node.js を起動する。

② 静的ファイル（HTML / CSS / JavaScript）を新バージョンのディレクトリに配置する。

③ シンボリックリンクを、新バージョンの静的ファイルディレクトリに切り替える。

なお、ポートは Server 側のビルド時に決定されるため、Client 側のビルドはその後に行う必要がある。

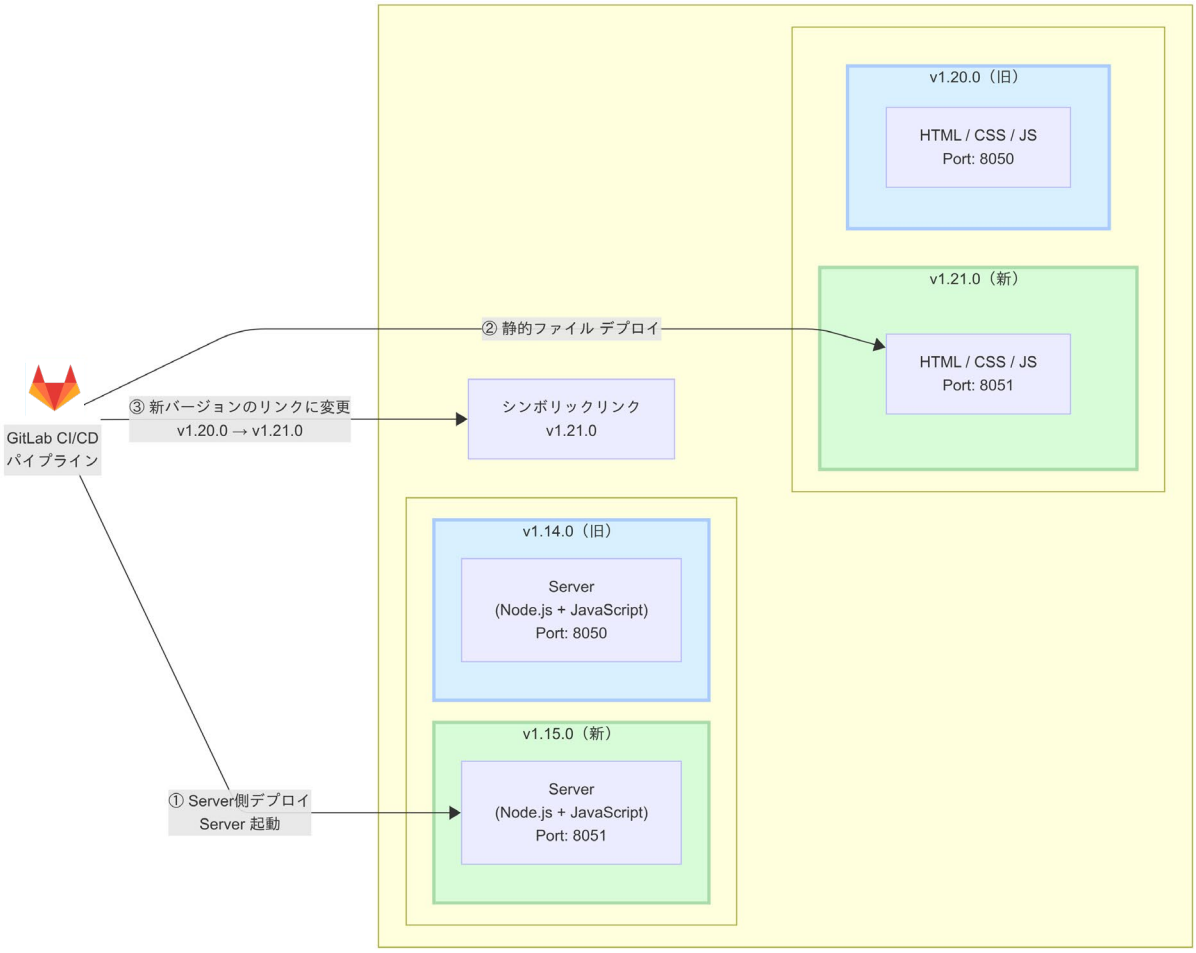


図 8．Blue-Green Deployment　導入後のリリース方法

4.3　静的解析ツールによる保守効率の向上

欠陥削減および保守効率の向上を目的として、静的解析ツールである SonarQube を導入した。SonarQube は Reliability（信頼性）、Maintainability（保守性）、Security（セキュリティ）といった品質指標を可視化できるほか、自動テストのコードカバレッジ情報も取り込み、テストの網羅状況を視覚的に把握することができる。

本取り組みでは、Reliability と Maintainability に着目し、これらに関する指摘事項の修正を実施した。導入初期段階では、Client 側で約 5,000 件、Server 側で約 600 件の指摘が検出された。そこで、表 1 に示す目標を設定し、指摘事項の修正作業を進めた。

表 1．SonarQube　指摘修正目標

モジュール	指標	修正後件数
server	Reliability	0 件
	Maintainability	100 件以下
client	Reliability	0 件
	Maintainability	1000 件以下

初期段階から修正後までの指摘件数の遷移を図 5、図 6 に示す。

図５．Server 側の指摘件数遷移

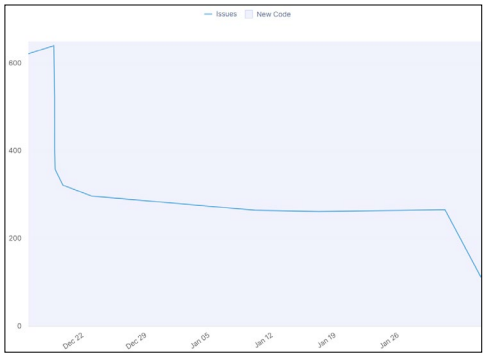


図６．Client 側の指摘件数遷移

5.改善策の実現方法

5.1 自動テストとコード解析の自動実行

自動テストとコード解析を継続的に行う構成を導入した。構成全体を図７に示す。開発者はソースコードを編集・Push し、GitLab がそれをトリガーとして パイプラインを実行する。Jest による自動テストと SonarQube によるコード解析が行われ、それぞれの結果は GitLab および SonarQube 上で可視化される。これにより、解析結果やテストカバレッジを即時に確認できる環境が整った。

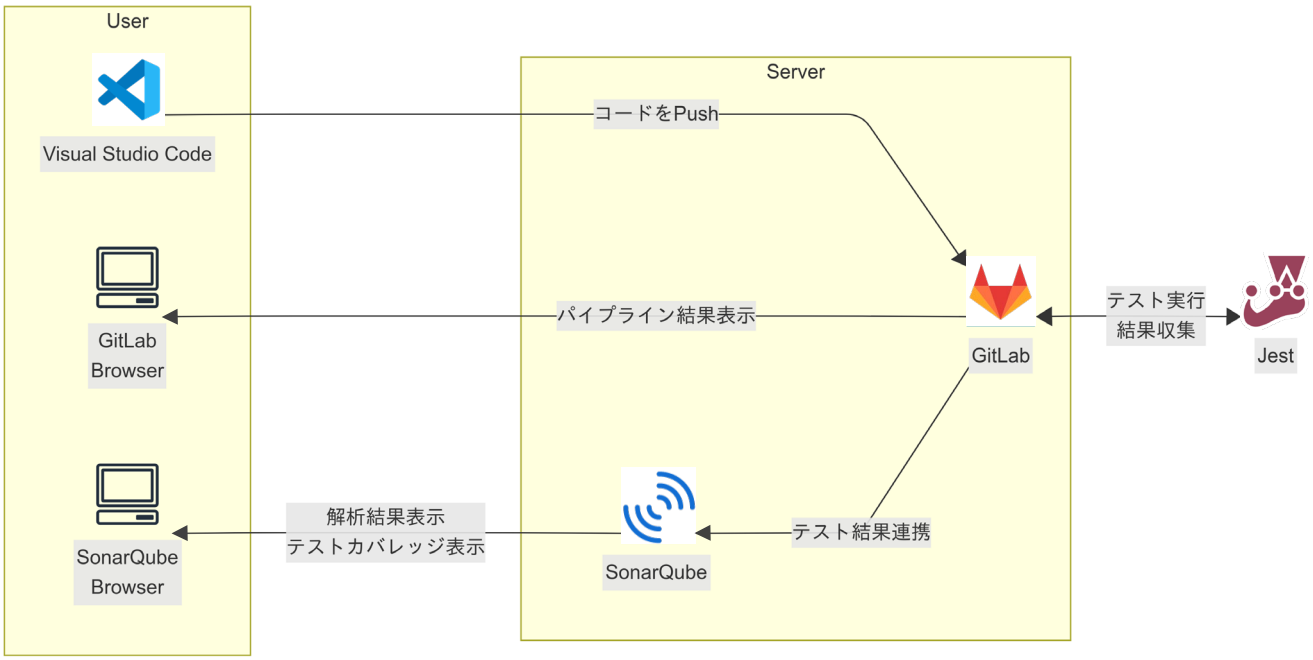
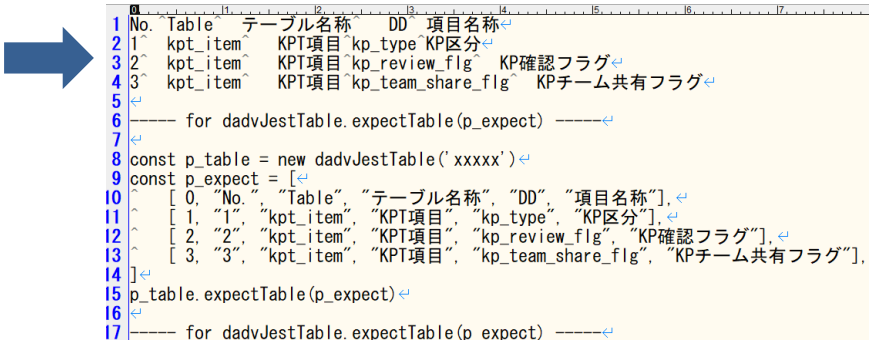


図７．自動テストとコード解析の自動実行構成

5.2 自動テスト作成効率化

既存機能の自動テストを効率的に開発するために、Sakura エディタのマクロを作成した。開発中のブラウザに表示される HTML テーブルをドラッグ＆ドロップでコピーし、Sakura エディタに貼り付け、マクロを起動すると Jest のテストコードが自動生成される。これにより、開発者が手動で記述する手間を削減することができる。実際の HTML テーブルと Sakura エディタで生成したテストコードを図８に示す。

No.	Table	テーブル名称	DD	項目名称
1	kpt_item	KPT項目	kp_type	KP区分
2	kpt_item	KPT項目	kp_review_flg	KP確認フラグ
3	kpt_item	KPT項目	kp_team_share_flg	KPチーム共有フラグ



```
No.^Table^テーブル名称^DD^項目名称
1^kpt_item^KPT項目^kp_type^KP区分
2^kpt_item^KPT項目^kp_review_flg^KP確認フラグ
3^kpt_item^KPT項目^kp_team_share_flg^KPチーム共有フラグ

----- for dadvJestTable.expectTable(p_expect) -----

const p_table = new dadvJestTable('xxxxx')
const p_expect = [
^    [ 0, "No.", "Table", "テーブル名称", "DD", "項目名称"],
^    [ 1, "1", "kpt_item", "KPT項目", "kp_type", "KP区分"],
^    [ 2, "2", "kpt_item", "KPT項目", "kp_review_flg", "KP確認フラグ"],
^    [ 3, "3", "kpt_item", "KPT項目", "kp_team_share_flg", "KPチーム共有フラグ"],
]
p_table.expectTable(p_expect)

----- for dadvJestTable.expectTable(p_expect) -----
```

図８． HTML テーブルから自動生成された Jest テストコード

5.3 マージリクエスト時における自動テストのカバレッジ確認の工夫

マージリクエストのレビュー時には、修正箇所に対する自動テストが適切に記述されているかを確認する一環として、カバレッジの状況も確認している。 しかし、実際の運用においては、他の開発者の Push によってカバレッジ情報が上書きされてしまい、対象の変更に対する正確なカバレッジ確認が困難になるという課題が発生した。この課題に対応するため、coverage 専用のブランチを作成し、各開発者が作成した自動テストはこのブランチに反映する運用に変更した。さらに、coverage ブランチに対する変更時のみ自動テストジョブを実行するようパイプラインを設定することで、変更対象のカバレッジ確認が可能となった。

5.4 開発ツールと SonarQube の連携による指摘対応の効率化

SonarQube によって検出された指摘を確認するために毎回 Web 画面を開き、該当するソースコードを手作業で探しに行く必要があり、開発効率の低下が課題となっていた。そこで、検出タイミングの早期化と対応の効率化を目的に、開発ツールである Visual Studio Code に SonarQube 専用のプラグインを導入した。このプラグインにより、図 7 に示すようにコード記述中にその場で指摘がハイライトされるようになり、開発者が早い段階で問題点に気づきやすくなった。

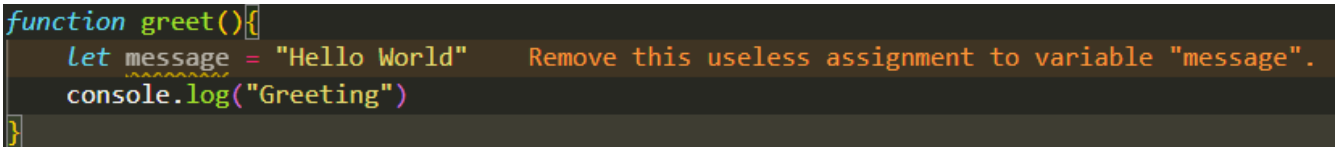


図７． SonarQube プラグインのリアルタイム解析表示

6.改善による変化や効果

6.1 自動テスト導入によるデグレの減少

自動テストの整備により、新機能追加や修正の際に、既存機能に対するデグレを事前に検知・防止できるようになった。その結果として、IT 時に発見されるデグレ件数は減少傾向を示している。表 2 は、自動テスト導入前後におけるデグレ件数の推移を示す。

表２．自動テスト導入前後におけるデグレ件数の推移

期間	自動テスト導入	デグレ件数	備考
2025 年 2 月	未導入	15 件	自動テスト未導入
2025 年 3 月	導入中	11 件	テストカバレッジ約 30%
2025 年 4 月	導入中	7 件	テストカバレッジ約 40%
2025 年 5 月	導入中	5 件	テストカバレッジ約 50%
2025 年 6 月	運用定着	0 件	テストカバレッジ約 60%
2025 年 7 月	運用定着	1 件(*1)	テストカバレッジ約 60%
2025 年 8 月	運用定着	1 件(*1)	テストカバレッジ約 65%
2025 年 9 月	運用定着	2 件(*1)	テストカバレッジ約 65%

(＊1)　4 件中 3 件は自動テスト未実装の機能

6.2 リリースの即時性向上

Blue-Green Deployment の導入により、本番環境への修正反映にかかる時間が大幅に短縮された。実際に、ユーザからのバグ報告に対して、半日以内に修正リリースを完了した事例も確認している。

6.3 保守効率向上

(1) 自動テスト導入によるコード理解性の向上

自動テストを整備した結果、メソッドの引数や返値の仕様がテストコードを通じて即座に把握できるようになったという副次的効果も得られた。従来は実装コードを読み解くしかなかったが、テストコードがそのまま利用例となることで、実装意図の理解速度が向上した。

(2) 静的解析ツールの導入による可読性の向上

静的解析ツールの導入し、指摘項目を改善した結果、コードの構造や意図が把握しやすくなった。具体的な数値による効果測定は難しいものの、チーム内からは以下のような改善により、コードの理解が容易になったとの声が上がっている。

- メソッドの長さを適切に管理し、処理のまとまりが明確になった
- 重複コードを削減し、無駄な記述が減ったことで理解しやすくなった
- 使われていない変数を削除し、コードがシンプルになった
- 深すぎる if 文の入れ子構造を解消し、処理の流れが追いやすくなった
- 正規表現の不要なエスケープを削除し、表現が簡潔になった

また、以下のような潜在的な不具合も検出・修正することができた。

- case 文の重複を解消できた
- case 文での break 文の書き忘れを修正した
- 不適切に || 演算子を使用していた箇所を見直し、意図しない動作を防止した
- if (a = b) のように、比較演算子 == と代入演算子 = の誤用を修正した

7.改善活動の妥当性確認

7.1 自動テスト導入の妥当性

自動テスト導入によるデグレ件数の減少は、表 2 の結果から明確に確認できる。導入前は毎月 10 件前後のデグレが発生していたが、自動テスト導入拡大に伴い、6 月以降はほぼ発生していない。特に運用定着期に発生したデグレの大半は自動テスト未実装の機能に起因しており、導入済みの機能では再現性のある防止効果が確認された。この結果、目標であったデグレの 7 割削減は達成されており、自動テスト整備はデグレ抑制という目的に対して妥当な施策であると判断できる。

7.2 Blue-Green Deployment 導入の妥当性

ユーザ報告に対する修正リリースが半日以内に完了した事例が複数あり、従来のリリース手順と比較して大幅な時間短縮が実現できた。これにより、本番環境への修正反映を迅速に行うという目的に対して、Blue-Green Deployment の導入は妥当である。

7.3 保守効率向上の妥当性

自動テスト整備や静的解析ツールの導入により、バグ修正や新規機能追加の作業が以前より理解しやすくなったとの開発者の声が上がっている。現時点では実際の工数削減を示すデータはないが、作業の効率化や理解速度の向上といった効果が実感されており、保守性向上に寄与していることが確認できる。

参考情報

[1] 和田卓人, “組織に自動テストを書く文化を根付かせる戦略 2024 秋”, SPI Japan 2024, 2024

[2] Christie Wilson, “入門 継続的デリバリー ―テストからリリースまでを安全に自動化するソフトウェアデリバリーのプロセス”, O'Reilly Japan, 2024

3B3　テスト自動化を前提としたテスト並列設計プロセスの導入と検証結果　鈴木貴広（株式会社デンソー）

＜タイトル＞

テスト自動化を前提としたテスト並列設計プロセスの導入と検証結果

＜サブタイトル＞

＜発表者＞

氏名(ふりがな)：鈴木　貴広(すずき　たかひろ)
所属：　株式会社デンソー

＜共同執筆者＞

氏名(ふりがな)：　堀川　まゆみ(ほりかわ　まゆみ)
所属：　株式会社デンソー

＜主張したい点＞

SDV 時代の車載ソフトウェア開発ではリリース速度と品質要求が同時に高まっている。従来の V 字モデルではテスト工程が後工程にあることから、手戻りや品質リスクが発生しやすい。また自動テストは開発後に後追いで作られるケースが多く、効率化の効果を得る時期が遅れてしまうケースもある。そこでソフトウェア開発とテストを並行して進める「テスト並列設計プロセス」を提案する。

「テスト並列設計プロセス」は V 字モデルから派生した W 字モデルに対して、自動テスト工程実施を前提とし、実施タイミングと各工程の入力となる成果物、テストエンジニアという役割をそれぞれ定義する。また成功率を上げるためにコミュニケーションガイドラインも定義する。この「テスト並列設計プロセス」では 2 人のエンジニアで協調して開発を進める必要があり難易度は高いが、このプロセスを利用することでリリース速度と品質を両立するだけでなく、設計時に自動テスト資産を作り出すことでより効果的に工数削減が可能となる。

＜キーワード＞

開発プロセス、効率化、自動テスト、マインドセット

＜想定する聴衆＞

ソフトウェアエンジニア、プロジェクトマネージャ、SEPG、SQA

＜活動時期＞

2024 年 5 月～

＜活動状況＞：発表内容に複数の事例が含まれる場合は複数選択可能です。

☐ 着想の段階(アイデア・構想の発表)
☑ 改善活動を実施したが、結果はまだ明確ではない段階
☐ 改善活動の結果が明確になっている段階
☐ その他(　　)

＜発表内容＞

1.背景

近年車載システムの高度化が進み、それに伴い搭載されるソフトウェアの複雑さも増している。また SDV 時代ではリリース速度の高速化が求められている。弊社はこれまで世界中のカーメーカー様に向けて、幅広い製品開発をすることで貢献してきた。車載ソフトウェア開発は特に安全性・信頼性が重要であり、高品質かつ効率的な開発プロセスが求められている。一般的に車載ソフトウェアは Automotive SPICE に準拠して V 字モデルを導入していることが多い。

また私たちの部署は中規模 ECU 開発部隊の中にあり、2021 年から現在まで「テストからソフトウェア開発を変える」をスローガンに様々な製品横断で活用できる共通テスト環境の構築、そのテスト環境で動作する自動テストを開発してきた。

2.改善したいこと

従来の V 字モデルではテスト工程が後工程に位置するため、テスト工程でバグを発見した際の手戻りや品質リスクの影響が大きくなってしまう。

また自動テストのテストシナリオを作成するためには自動テストに関する特別なスキルが必要だが、開発エンジニアがスキルを保有していないことが多く、ソフトウェアリリースタイミングではマニュアルでテストを実施、リリース後に自動テストスキルを保有したテストエンジニアが自動テスト環境を作るケースが多い。後追いになることで効率化効果を刈り取るタイミングを逸することもあり、自動化が進まない要因の 1 つになっている。タイムリーにテスト資産の棚入れがされないことでテスト再利用が難しく、自動テスト環境を活用した開発ができず、同じ機能に対する変更要求でも工数が増加してしまうケースがあった。

3.改善策を導き出した経緯

V 字モデルから派生した W 字モデルから着想を得て、自動テストを前提とし「テスト並列設計プロセス」を定義した。W 字モデルではテストを早期に設計・実行できるメリットはあるが、2 人のエンジニアで協調して開発を進めるため難易度が高いこと、コミュニケーションオーバーヘッドによる工数増加の課題もあった。そこで仮説を立て、各工程の実施タイミング、入力となる成果物、テストエンジニアという役割を定義することで課題の解消を狙った。

4.改善策の内容

「テスト並列設計プロセス」の実施タイミングと各工程の入力成果物を以下に示す。ソフトウェア開発は 開発エンジニアと テストエンジニア の 2 名で対応する。下図の黄色い部分は開発エンジニア、青い部分はテストエンジニアが担当する。それぞれのエンジニアはプロセスについて正しく理解すること、期日厳守で開発を進めることを意識して取り組むこととする。

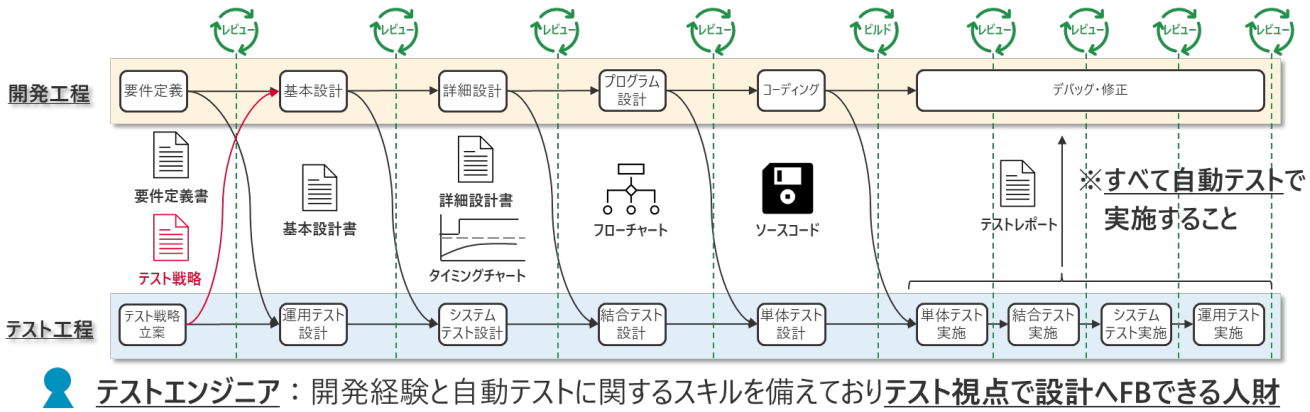


図 1. テスト並列設計プロセス

5.改善策の実現方法

定義した「テスト並列設計プロセス」について実プロジェクトの開発業務にて適用、効果について検証した。

■対象

・X 社向け エンジン制御 ECU ソフトウェア ダイアグ通信機能に対する変更

■実施条件

・開発フェーズ　：量産前の試作フェーズ

・開発規模　：V 字モデルで 8 週間の見積り

・体制　：デンソー社員 A(開発工程)， BP 社員 B(テスト工程)， BP 社員 C(クロスチェック)

・スキル　：設計担当は開発経験あり、テスト担当は開発経験と自動テスト知識を保有

・検査環境　：テストは自動化を前提とする

■評価指標

・定量評価　：納入までのリードタイム(**見積り：8 週間**，**目標：4 週間**)，工数，テストスクリプト数

・定性評価　：コミュニケーションの観点で振り返り実施

6.改善による変化や効果

実プロジェクトの開発業務に適用した結果を以下に示す。プロジェクト途中での要求変更があったが 0.5 週間増の 4.5 週間のリードタイムで実施することができた。これにより「テスト並列設計」でリードタイム短縮の効果を確認できた。

■定量評価

・リードタイム：

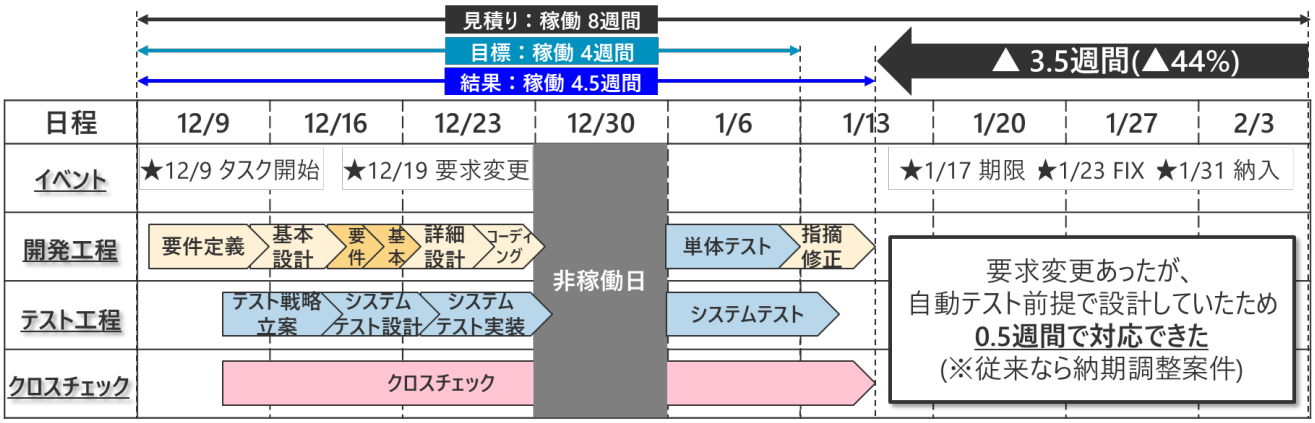


図 2. 検証結果

・工数　：開発工程 125h，テスト工程 96.25h，クロスチェック 49.5h

・テストスクリプト数　：19 個

■定性評価

・コミュニケーション　：事前に関係構築済み、タスク開始時にグループチャット作成したことでスムーズに連携できた

7.改善活動の妥当性確認

適用結果から、「テスト並列設計プロセス」を活用することで開発リードタイムを短縮、リリース速度の向上が可能であることが確認できた。しかし開発エンジニアとテストエンジニアに認識のズレやコミュニケーション不足が発生すると手戻りリスクが高いことも振り返りで挙げられた。そこでプロセス定義だけでなく、コミュニケーションガイドラインを設定することが重要だと考え、以下のように定義した。

■コミュニケーションガイドライン

表 1. コミュニケーションガイドライン

基本方針	具体的なアクション	マインド醸成
・お互いに対等な関係であること ・お互いに積極的な意見交換をすること ・全員がリードタイム短縮を意識すること	・プロジェクト開始時の顔合わせ ・定期的な進捗確認 ・成果物イメージの事前整合 ・専用チャットグループ設置とFB即時共有	・心理的安全性を確保すること ・ポジティブFBを心掛けること ・オープンな議論をすること

またテストエンジニアが早期にテスト設計、設計工程へ FB することから手戻りも減り、品質向上が確認できた。そのため「テスト並列設計プロセス」においてテストエンジニアという役割が非常に重要になることが明確になった。そこで「テスト並列設計プロセス」を他のプロジェクトに展開していくだけでなく、テストエンジニアの育成にも注力する必要があると考えられる。現状では経験の浅いエンジニアがテスト工程を担当することも多く、テストエンジニアが不足している状況である。そこで短期的には通信などの共通部品を中心にテストエンジニア部隊を別で用意することで対応、長期的には各プロジェクトにテストエンジニアを配置できるように教育を実施していく。今後も様々なプロジェクトに対して「テスト並列設計プロセス」の妥当性を検証していく。

参考情報

[1] ソフトウェアエンジニアリング講座<1> ソフトウェア工学の基礎 IT トップガン育成プロジェクト 著
[2] 情報工学レクチャーシリーズ ソフトウェア工学 高橋直久、丸山勝久 著
[3] ソフトウェア品質知識体系ガイド(第 3 版) 飯泉紀子、鷲崎弘宜 著
[4] IV&V ガイドブック【導入編】Ver2.1 宇宙航空研究開発機構(JAXA) 著
[5] SEA SPIN Meeting May 2012 W モデル
[6] 2024 DENSO DRIVEN BASE モビリティの潮流がわかる「5 つのポイント」
[7] 2024 株式会社デンソー ソフトウェア戦略説明会