生成 AI を活用したプロセス定義レビューにおける構造化データ形式と文書表現形式の精度比較

研　究　員　：池永 直樹（株式会社デンソークリエイト）
主　　査　：田中 桂三（オムロン株式会社）
副　主　査　：白井 保隆（株式会社東芝）
アドバイザー：中森 博晃（ビースラッシュ株式会社）

研究概要

本研究は,継続的に改善される標準プロセスのレビューに生成 AI を活用するにあたり，よりレビュー精度を高めるレビュー対象の入力データ形式を検証することを目的とし，構造化データ形式(YAML)と文書表現形式(Markdown)を比較した．実験の結果，構造化データ形式(YAML)は，ノード(プロセス定義要素)間の関係理解が不要な観点で文書表現形式(Markdown)と同等以上，関係理解が必要な観点で高精度を示し，さらにトークン数が少ないことが判明した．これにより，生成 AI を活用したプロセス定義のレビューにおいて，構造化データ形式(YAML)の採用が有効であると判断する．

1.　はじめに

筆者は自動車部品サプライヤーにおいて，自組織におけるプロセス改善活動を推進している．自動車を取り巻く環境変化に伴うビジネス環境や開発技術の変化，法規・国際／業界標準対応の増加に対応するために，標準プロセスを継続的に改善している．一方で，プロセス改善の作業工数は増加している．この課題に対し，作業工数を抑制する対策の一つとして，プロセス定義のレビューに生成 AI を活用し，レビュー工数を削減することを考えている．そこで本研究では，プロセス定義レビューに適したデータ形式を検討する．

以降 2 章で研究の背景を説明し，3 章では仮説立案からプロセス定義レビューの入力データ形式を提案する．4 章で実験方法と結果を示し，5 章では実験結果を分析し，入力データ形式の有効性を評価する．最後に 6 章で研究成果のまとめと今後の展望を述べる．

2.　研究の背景

2.1　標準プロセスの役割

標準プロセスは，各開発プロジェクトの QCD 目標を安定的に達成するために重要な役割を果たしている．プロセスの標準化により，担当者によるバラツキを低減し，不要な作業を抑制することで，生産性と品質を同時に高められるからである．短周期で変化するビジネス環境や開発技術の進化に追従し，競争力を維持・向上するために，標準プロセスの継続的改善は必要不可欠である．

また，筆者が従事する自動車業界では，技術潮流の変化から機能安全やサイバーセキュリティ，AI 品質保証，Automotive SPICE など，対応が求められる法規・国際／業界標準が増加している.これらの要求事項を標準プロセスへ取り込み,実務で対応し易くしている．

2.2　標準プロセスの記述やレビューの現状

前節の結果，短いサイクルのプロセス改善に加え，標準プロセスの情報量と複雑度が増大し，プロセスの定義・記述及びそのレビューに要する工数は増加傾向にある[1].

筆者の組織では工数増加を抑制するために，標準プロセス定義の効率化，品質及び保守性の向上を狙ったプロセス記述のデジタル化手法[1]を適用し，システム・ソフトウェア設計ツールである Next Design[2]を使用している．

この手法及びツールでは図１に示したように，プロセス定義のメタモデルに基づいてプロセスを定義する．定義されたプロセスは「ノード＋リンク」のグラフ構造として保存される．ノードはアクティビティや成果物などプロセス定義要素のインスタンスで，プロパティ(例:ID，名称，目的)を持つ．リンクは，ノード間の関係性で，所有(例:プロセスとアクティビティの所有関係)，参照(例:アクティビティと成果物との入出力関係)など複数の種類がある．また，ビュー(画面表示形式)を定義しておき，グラフ構造で保存されたデータはビュー定義に従い人が理解しやすい形式に整形されて表示される．

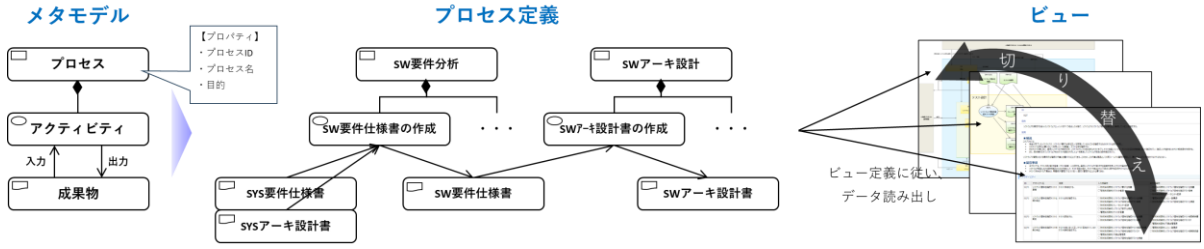

図１ プロセス記述のデジタル化手法におけるプロセス定義のイメージ

この手法により，データは一元的に保持され，データとビューが分離されることから，ID・名称等のプロパティの不整合，同一プロセス定義に対する異なるビュー間(例:ドキュメント形式と PFD[3])での定義内容の不整合など表現品質の一部の問題が予防でき，プロセスの記述やレビュー作業の工数も削減できている．しかし，プロセス定義内容そのものの読み易さ・表現設計などの表現品質，有効性・適切性・規格類への準拠性・正確性・完全性などの内容品質の確保は対象外であり，これらは人手で保証している．

2.3　課題

前節で述べたプロセス記述のデジタル化手法では対象外である側面のうち，機械的検証に親和性のある表現品質・準拠性・正確性・完全性などの観点に生成 AI を活用し，価値判断を要するプロセスの有効性(例:プロジェクト目標達成に対して有効か)や適切性(例:製品や組織の特性にあっているか)などの観点に人手を集中させたいと考えている．

生成 AI を活用したレビューの先行事例では，セルフチェックの自動化を実証した研究[4]，ソフトウェア要求仕様のレビュー／自動修正に生成 AI を活用する事例[5]など，ソフトウェアエンジニアリング成果物のレビューへの生成 AI の活用の有効性が示されている．しかし，プロセス定義そのものを生成 AI でレビューした事例は見当たらない．

Next Design は「ノード＋リンク」のグラフ構造を構造化データとして保持している．一方で，生成 AI は自然言語を含む大規模なデータセットで学習されており，自然言語処理に強みを持つ．従って，生成 AI によるプロセス定義のレビューを高精度で実現するために，プロセス定義をどんなデータ形式で入力するか見極める必要がある．

よって，本研究の課題を以下と設定する．

課題：自然言語処理を得意とする生成 AI に，どのようなデータ形式でプロセス定義内容を入力すればよいか

2.4　生成 AI への構造化データ入力に関する先行研究

生成 AI への構造化データ形式のデータ入力については様々な研究がされている．

データ記述言語では，YAML や JSON などに対する LLM の構造理解が体系的に評価され，比較的簡単な構造では高精度に解釈できる一方で，深い入れ子や長文では性能が低下することが示されている[6]．グラフ記述言語では，GraphML 等を入力とする構造・意味理解タスクで LLM の能力が検証され，一定の性能は示すものの専門モデルに劣り，入力設計やプロンプトの工夫が性能向上に重要であるとされている[7]．また，表形式の設計文書を JSON 形式と Markdown 形式へ変換し見出し・値の関係を明示化することで，GPT 系モデルによる

欠陥検出の再現率が向上する事例も報告されている[8].

3. 仮説と提案手法

3.1 仮説

先行研究では，構造化データ形式(データ記述言語，グラフ記述言語)を生成 AI に入力したとき，標準的なデータ量・複雑度のデータセットでは良好な解釈精度を示すことが報告されている．一方で，大量データや関係性が複雑なデータセットでは，解釈精度が低下することも指摘されている．

プロセス定義書は，プロセス定義要素間に所有・参照などの有向関係を含む構造化されたデータである．よって，先行研究で示された構造化データ入力の有効性はプロセス定義書に対しても期待できる．一方，記述量の多さや関係の複雑さに起因する精度低下も報告されていたが，プロセス記述のデジタル化手法が持つメタモデル(上位概念スキーマ)を併せて入力し，構造的・意味的制約を明示することで対策できる可能性がある．

また，構造化データ形式はノードとリンクを明示的に一意に管理できるため，同一情報の重複を少なく記述できる．一方，自然言語での文書表現では，同一情報を文書構造や記述の繰り返しによって表現するため冗長性が高く，文脈依存による曖昧性も生じやすい．この差は，レビュー実行時の入力トークン数(コスト)や再現性に影響すると考えられる．

よって，本研究の仮説を以下と設定する．

仮説：構造化データ形式は，トークン効率が高くノード間の関係理解を必要とするレビュー観点において，文書表現形式より高い精度を発揮するのではないか

3.2 研究課題

本研究では，生成 AI を用いたプロセス定義のレビューにおいて，明示的な構造を持つデータ入力(構造化データ形式)と，生成 AI が得意とする自然言語での文書入力(文書表現形式)の性能を比較する．比較における研究課題(RQ:Research Question)を以下と設定する．

RQ1：ノード間の関係の理解が不要なレビュー観点において，構造化データ形式は文書表現形式と同等以上の精度であるか？

プロセス定義において，生成 AI が文書表現形式だけでなく構造化データ形式も適切に扱えるかを確認するため，ノード間の関係理解が不要な単純な検索型のレビュー観点で両形式の精度を比較する．

RQ2：ノード間の関係の理解が必要なレビュー観点において，構造化データ形式の方が文書表現形式より精度が高いか？

プロセス定義の構造を明示的に保持した構造化データ形式の優位性を確認するため，ノード間の関係理解が必要なレビュー観点で両形式の精度を比較する．

RQ3：構造化データ形式は文書表現形式よりトークン効率が高いか？

トークン数の観点で構造化データ形式の優位性を確認するために，同一のプロセス定義書に対する両形式のトークン数を比較する．

4. 提案手法

4.1 構造化データ形式(YAML の採用)

プロセス定義のメタモデル及びメタモデルに基づいて定義されたプロセスの「ノード＋リンク」のグラフ構造を，YAML 形式に変換する方法を採用する．

JSON などの類似のデータ記述言語も存在するが，本研究ではテストデータ作成時の可読性や注釈記述の容易性を重視し，YAML を採用した．トークン数の観点では，JSON は {}(オブジェクト)や [](配列)，さらに ：・，といった構造記号を用いて階層を明示的に表現する．一方，YAML は主としてインデントと改行により階層を暗黙的に表現し，： と － を補助的に用いる．どちらがトークン数を少なくできるかはケースによる[9]ため，本研究では

前述の可読性と注釈記述の利点を優先した．

さらに，GraphML のようなグラフ記述言語も候補となり得る．GraphML はノードとリンクの関係を厳密に表現できる．しかし，XML ベースでタグや属性を多用するため冗長性が高くトークン数が増加する懸念があることから，本研究では採用しなかった．

YAML 形式の記述例を図 2 に示す．

図 2 YAML 形式の記述例

4.2 文書表現形式(Markdown の採用)

ツール(Next Design)画面に表示される人が読める形に整形されたビューの内容を Markdown 形式に変換する方法を採用する．

プロセス記述のデジタル化手法では利用者の関心事に応じたビューを定義し，そのビュー定義に従い「ノード＋リンク」のデータを表示している．このビューが，技術者をはじめとする利用者が閲覧するプロセス定義書であり，プロセス定義者がレビューする対象でもある．この人が読めるプロセス定義の内容を Markdown 形式にする．例を図 3 に示す．

図 3 Markdown 形式の記述例

5. 実験

5.1 実験方法

(1) 評価指標

3.2 節で述べた各 RQ の評価指標を表 1 に示す.

表 1 RQ ごとの評価指標

RQ	評価指標
RQ1	レビュー精度を適合率(Precision), 再現率(Recall)で評価する
RQ2	レビュー精度を適合率(Precision), 再現率(Recall)で評価する
RQ3	トークン効率をプロセス定義のそれぞれの形式でのトークン数で評価する

(2) 実験対象のプロセス定義書

筆者の組織内にあるメタモデル, ビュー定義, ならびにプロセス定義内容が異なる 3 つのプロセス定義書を使用する.

(3) 生成 AI

生成 AI は筆者の所属組織で利用が認められている GPT-5(Azure OpenAI)を使用する.

(4) レビュー観点

3.2 節で述べた RQ1, RQ2 を評価するために, 筆者の組織でプロセス定義書をレビューする際に用いられている代表的なレビュー観点をピックアップして使用する.

表 2 実験に用いるレビュー観点

RQ	ID	レビュー観点	必要なノード間の関係
1	1-1	活動が定義されているか	なし
	1-2	活動の具体的な実施方法(手順・基準・観点等)が定義されているか	なし
	1-3	成果物が定義されているか	なし
2	2-1	入力成果物が活動間の順序と整合しているか	活動と成果物の入出力関係 活動間の順序関係 活動間の集約関係
	2-2	プロセス・アクティビティなどの各レベルで定義されている入出力成果物が整合しているか	活動間の集約関係
	2-3	活動の開始基準が活動間の順序と整合しているか	活動間の順序関係 活動間の集約関係

(5) 実験手順

次の手順で実施する. なお, 本実験ではプロセスの専門知識を生成 AI に与えない.

① プロセス定義書(Next Design)から YAML 形式及び Markdown 形式を作成する.
② ①で作成した YAML 形式及び Markdown 形式の実験データのトークン数を OpenAI 社が提供しているトークナイザー[10]で計測する.
③ 筆者がレビューして指摘を出す.
④ 生成 AI に①で作成した YAML 形式及び Markdown 形式を入力し, 出力を得る.
⑤ ③を正答として, ④で得た生成 AI の出力を評価する.

5.2 実験結果

3.2 節で述べた各 RQ の実験結果を示す.

(1) RQ1 の実験結果

実験手順③～⑤で得られたノード間の関係理解を必要としない 3 つのレビュー観点(ID 1-1～1-3)に関する実験結果を表 3 に示す. 実験結果の詳細と使用したプロンプトは「付録 1.RQ1 のプロンプトと実験結果」を参照.

表 3 RQ1 の実験結果

ID	試行数	YAML 形式(構造化データ)		Markdown 形式(文書表現)	
		適合率	再現率	適合率	再現率
1-1	20	0.87	0.57	0.76	0.54
1-2	20	0.67	0.78	0.62	0.78
1-3	20	0.69	0.69	0.49	0.69
合計	**60**	**0.71**	**0.71**	**0.61**	**0.70**

(2) RQ2 の実験結果

実験手順③～⑤で得られたノード間の関係理解を必要とする 3 つのレビュー観点(ID 2-1～2-3)に関する実験結果を表 4 に示す．実験結果の詳細と使用したプロンプトは「付録 2.RQ2 のプロンプトと実験結果」を参照．

表 4 RQ2 の実験結果

ID	試行数	YAML 形式(構造化データ)		Markdown 形式(文書表現)	
		適合率	再現率	適合率	再現率
2-1	20	0.89	0.77	0.88	0.67
2-2	17	0.97	1.00	0.93	0.93
2-3	11	0.89	0.52	0.70	0.45
合計	**48**	**0.91**	**0.76**	**0.86**	**0.68**

(3) RQ3 の実験結果

実験手順②で得られた実験対象 3 件のプロセス定義書の YAML 形式及び Markdown 形式のトークン数を表 5 に示す．比率は，Markdown 形式に対する YAML 形式の比率である．

表 5 RQ3 の実験結果

プロセス定義書	YAML 形式(構造化データ)	Markdown(文書表現)	比率
#1	70,189	76,571	92%
#2	104,074	193,450	54%
#3	145,672	150,995	96%
合計	**319,935**	**421,016**	**76%**

6. 考察

6.1 RQ の考察

(1) RQ1 の考察

レビュー観点 1-1,1-2,1-3 の合計では,適合率は YAML 形式=0.71／Markdown 形式=0.61,再現率は YAML 形式=0.71／Markdown 形式=0.70 となり，適合率は YAML 形式の方が高く，再現率はほぼ同じ値となった．よって，ノード間の関係の理解が不要なレビュー観点において，YAML 形式(構造化データ)は Markdown 形式(文書表現)に全体として同等以上の精度を示していることから，RQ1 は有効だったと判断できる．

全体的には，本実験ではプロセスの専門知識を与えなかったことから，意味的な解釈(同義語，言い換え，など)が必要なケースにおいて，適合率と再現率が低下したと考える．

各レビュー観点の試行結果から，以下の傾向が確認できた．

<YAML 形式(構造化データ形式)>

- 関係のない箇所を検出することが少ない(誤検出するケースが少ない)
- 記載箇所が分散している場合に見逃しやすい

<Markdown 形式(文書表現形式)>

- 複数の候補がある場合において関係のない箇所を検出する(誤検出しやすい)
- 意味的に近い記述や複数箇所に分散した記述を“広く拾う”傾向があり，見逃しは減

るが正確性に欠ける(誤検出となる)場合がある

YAML 形式は，階層や属性が明示的であり，構造的に記述箇所が明確なため検索対象のインスタンスを限定でき，誤検出が抑制されたと考えられる．一方で，レビュー対象の情報が分散して記載されている場合に見落とす傾向にあり，検索範囲のプロンプト依存性が高いと考えられる．実施方法の記載箇所(レビュー観点 1-2)は一か所に書かれず，複数のプロパティ(ユーザ視点では“欄”)や階層(プロセス・アクティビティなど)を跨ぐなど分散することがあり，Markdown 形式の方が見逃しは少なく，再現率で優れることが多かった．

Markdown 形式は，プロセス定義が冗長化されて記述されていること，及び，生成 AI がプロセス定義の構造を推論してその意味を理解しなければならないことから，誤検出が起こりやすいと考えられる．関連情報と非関連情報を区別しにくく，例えば，成果物定義を特定すべきところを同一名称のテンプレートを特定するなど，意味的に異なる箇所を誤って特定するケースが見られた．一方で，冗長化により対象語句が繰り返し登場するため，それを拾いやすいことから，再現率は確保される傾向にあると考えられる．

(2) RQ2 の考察

レビュー観点 2-1,2-2,2-3 の合計では,適合率は YAML 形式=0.91／Markdown 形式=0.86,再現率は YAML 形式=0.76／Markdown 形式=0.68 となりいずれも YAML 形式の方が高い結果となった．よって，ノード間の関係の理解が必要なレビュー観点において，YAML 形式(構造化データ)は Markdown 形式(文書表現)に比べ誤検出と見逃しを低減できる傾向が一貫して確認できたことから，RQ2 に対して有効だったと判断できる．

各レビュー観点の試行結果から，以下の傾向が確認できた．

<YAML 形式(構造化データ形式)>

- 活動間の順序の理解が必要な場合の見逃しが少ない
- 成果物の入出力・活動の上下階層の理解が必要な場合の誤検出が少ない

<Markdown 形式(文書表現形式)>

- 活動間の順序の理解が必要な場合の見逃しが多い
- プロセス定義の文書構成(関連語句の出現順)により，誤検出・見逃しが生じる

YAML 形式では，メタモデルと入出力関係・順序関係・集約関係などのリンク情報を明示的に保持しているため生成 AI がプロセス定義の構造を把握し易く，誤検出・見逃しが抑制されたと考える．特に，複数のプロセスに跨った活動間の順序性を読み解く必要があるケースや“成果物.成果物特性”の集約関係を扱わなければならないケースでは，YAML 形式が Markdown 形式を上回る傾向が強かった．

Markdown 形式では，自然言語の線形構造に依存し，プロセス定義要素の関係の復元を文脈から行う必要があるため，見逃しや誤検出が増加すると考えられる．特に活動間の順序の理解を含む場合(レビュー観点 2-1，2-3)にその傾向が強かった．一方で，記述が複数箇所に冗長化されている場合にはその冗長性がプラスに働き，冗長箇所を拾うことで再現率が YAML 形式を上回るケースもあった．

また，レビュー観点 2-3(開始基準と活動順序の整合)について両形式の再現率を下げた共通要因として，開始基準の記載ルールに関する指摘の検出漏れが挙げられる．これはプロンプトでこの観点を明確に指示していなかったことが原因と考えられる．これを対策すると YAML 形式の再現率は 0.52→0.87，Markdown 形式の再現率は 0.45→0.74 に上昇する．

(3) RQ3 の考察

実験対象 3 件のプロセス定義書では，3 件とも YAML 形式(構造化データ)の方が Markdown(文書表現)よりトークン数が少なく，合計で 24%の削減が確認できたため，RQ3 に対して有効だったと判断できる．

プロセス定義書別では，定義書#2 で 54%と顕著な削減が見られ，定義書#1 と#3 は 10%未満と軽微な削減であった．定義書#2 は活動名を「*プロセス名_アクティビティ名_タスク名*」で連結する命名規則を採用しており，Markdown 形式ではこの長い活動名が複数箇所に繰り

返し登場していた．定義書#1 は冗長化の対象である参照関係が少ないメタモデルであったため差が小さくなった．また，定義書#3 では使用されていないプロセス・アクティビティなどのプロパティが多数あり，これらが YAML 形式には含まれていたが Markdown 形式に含まれておらず，Markdown 形式の冗長化を相殺しており差が小さくなっていた．

6.2　妥当性への脅威

前節の考察から，分散された記述や順序性定義の有無や登場頻度などのプロセス定義書の特性によって両形式の指摘検出力が変わる．これらの特性の偏りによっては構造化データ形式(YAML)の優位性が下がる可能性があるが特性が与える影響を十分検証できていない．

7.　まとめ

7.1　研究成果

本研究は，プロセス定義レビューの工数削減に向け，2 章で述べたプロセス記述のデジタル化手法の実現方法である“Next Design による「ノード＋リンク」のグラフ構造でプロセス定義を保持する仕組み”を前提にプロセス定義レビューへの生成 AI の活用を考え，「課題：自然言語を得意とする生成 AI に，どのようなデータ形式でプロセス定義内容を入力すればよいか」の解決を目的とした．そこで，構造化データ形式として YAML，文書表現形式として Markdown の二つの形式を提案し，RQ1～3 でレビュー精度とトークン効率を比較した．実験の結果，構造化データ形式(YAML)は文書表現形式(Markdown)に比べ，ノード間の関係理解が不要なレビュー観点では同等以上，ノード間の関係理解が必要なレビュー観点ではより高精度であり，さらにトークン数も少ないことを確認した．以上より，妥当性への脅威は残るが，RQ1～3 の結果により仮説は支持されたため，プロセス定義書のレビューの入力として構造化データ形式(YAML)を採用することが有効と判断する．

7.2　今後の展望

今回の実験では妥当性への脅威が残っているため，以下に取り組む．

(1) 生成 AI でのレビュー精度を考慮したメタモデルやプロセス定義の記述ルールの定義
(2) 記述内容(分散記述や冗長化など)によるレビュー精度への影響の測定

参考文献

[1] 池永直樹， DX アプローチによるプロセス記述 -プロセス記述の効率化，品質および使用性向上に向けた取り組み-，4th NSPICE Conference
[2] 株式会社デンソークリエイト，システム・ソフトウェア設計ツール Next Design，https://www.nextdesign.app/
[3] 清水吉男，PFD の基本，AFFORDD カンファレンス資料，2018
[4] 西本 真由，生成 AI を活用したレビュー効率の向上と有効性の検証，SPI Japan 2025
[5] Lucas Koch, Towards AI-ready Software Engineering Bridging Research and Industry, 10th Korean SPICE Network International Conference x 2nd Asian SPICE Conference
[6] Zhouhong Gu et al., StrucText-Eval: Evaluating Large Language Model’s Reasoning Ability in Structure-Rich Text, arXiv:2406.10621
[7] Jiayan Guo et al., GPT4Graph: Can Large Language Models Understand Graph Structured Data? An Empirical Evaluation and Benchmarking, arXiv:2305.15066
[8] Takasaburo Fukuda et al., Development of Automated Software Design Document Review Methods Using Large Language Models, SANER 2025
[9] YAML vs JSON Performance [Benchmarks], https://jsontoyamlconverter.com/yaml-vs-json/performance/
[10] https://platform.openai.com/tokenizer