生成 AI によるソフトウェアパターン適用の試行 ～ソフトウェア設計品質向上に向けたプロンプトの考察～

伊藤 弘毅
三菱電機株式会社
Ito.Hiroki@dr.MitsubishiElectric.co.jp

徳本 晋
富士通株式会社
tokumoto.susumu@fujitsu.com

栗田 太郎
ソニー株式会社
taro.kurita@sony.com

石川 冬樹
国立情報学研究所
f-ishikawa@nii.ac.jp

要旨

生成 AI は，ソフトウェア開発の生産性を向上させる効果が期待されている．しかし，製品開発では生産性の向上だけではなく，要求品質を満足するためにソフトウェアの構造や振舞いを設計段階で定めることも重要な視点である．ソフトウェアの設計品質の確保を実現する一つの方法に，ソフトウェアパターンの適用が挙げられる．生成 AI のチャットサービスでは，プロンプトで実現したい内容を適切に記述する必要があるが，設計指針が体系的に整理されたソフトウェアパターンは，品質向上に有効な設計施策を効果的に反映できる可能性がある．本論文は，ソフトウェアの設計書を対象として，生成 AI のプロンプトを使用することでソフトウェアパターンの解決策の反映が可能か実験した結果を記述する．また，当該実験を通じて得られた知見から，ソフトウェア設計品質の向上を目的としたプロンプトパターンを考察した．

1. はじめに

生成 AI はメールの文章や画像，音楽など様々なものを出力させることができ，人々の暮らしを大きく変化させる可能性を持っている．生成 AI は，特にソフトウェアエンジニアリングの分野に大きな影響を与えると考えられている．ソフトウェアエンジニアリングの分野について，現状の年間支出額における 20 から 45 パーセントが直接的に生産性に影響を与えるという試算も出されている[1]．実際に，実装内容を類推することでコーディングをサポートする GitHub Copilot も登場しており，ソフトウェア開発者の実装作業の生産性を向上させる機能をサービスとして提供する事例も出てきている．

上記の通り，生成 AI はソフトウェア開発の生産性を改善する効果が見込まれるが，一方で製品開発では最終的な品質の確保も重要な側面である．本論文は，生成 AI を製品品質の確保に活用する方法について，実験および考察した内容を記述する．

2. 背景

ChatGPT に代表されるチャットサービスの生成 AI を活用するためには，プロンプトと呼ばれる質問を適切に入力することにより，意図する出力を得られるように工夫する必要がある．これは，生成 AI に入力するプロンプトの記述の具体性や組立て方によって，得られる結果が変化するためである．例えば，プロンプトの記述が明確でない場合，プロンプトの内容が別の意味に解釈され，意図とは異なる結果が出力される可能性がある．生成 AI に対するプロンプトを作成する方法論は，プロンプトエンジニアリングと呼ばれる技術領域で議論されている．プロンプトエンジニアリングは，様々なアプリケーションや研究テーマを対象に言語モデルを効率的に使用するため，プロンプトの作成や最適化を実現する原則である[2]．例えば，DAIR.AI が公表しているプロンプトエンジニアリングガイドでは，複数回のプロンプトで文脈を理解させる Few-Shot Prompting や，推論の過程をプロンプトで指定することで正しい解を出力させる Chain-of-Thought Prompting などが紹介されている[2]．また，ChatGPT の開発元である OpenAI は，良い結果を得るためのプロンプティングの 6 つの戦略を公開している[3]．

ソフトウェア品質の確保が期待できる方法の一つに，ソフトウェアパターンの適用がある[4]．ソフトウェアパターンは，ソフトウェア開発の特定の文脈で繰り返し発生する問題とそれに対する解決策を体系化して整理したものである．特に，主にソフトウェアの詳細設計段階における構

造や振舞いの設計指針を示したものはデザインパターン，ソフトウェアの全体構造などアーキテクチャ設計時における設計指針を示したものをアーキテクチャパターンと呼ぶ．また，近年ではクラウドや機械学習に関連するソフトウェアパターンも多く発表されている．

生成 AI のソフトウェア開発への応用は，ソースコードを対象にした研究が進んでおり，JavaScript コードの生成の試行[5]のほか，GitHub Copilot のコード生成による生産性向上の評価[6]や，ソースコードの不具合修正の性能分析[7]など多くの研究結果が報告されている．一方で，ソフトウェア設計に対する活用に関しては，設計者と ChatGPT が連携したアーキテクチャ設計のケーススタディの報告[8]があるが，ソースコードへの適用と比較すると報告数は少ない．実際の製品開発では，設計なしにコードを書くことは稀であり，要求品質を満足するための構造や振舞いを設計段階で定めることが重要である．仮に生成 AI のプロンプトの指示で，先述のソフトウェアパターンを作成中の設計書に適用できれば，製品開発における生成 AI の活用の幅が広がると考えられる．具体的には，設計書の記述に対するソフトウェアパターンの適用に生成 AI を利用することにより，文章生成による生産性の向上や，適用方法の例示によるパターン適用の誤り防止などに役立つと考えられる．

そこで我々は，生成 AI のプロンプトによりソフトウェアの設計書にソフトウェアパターンの解決策が反映可能か評価する実験を行った．

本論文の研究課題を以下の通り定める．

RQ1：生成 AI は設計書の記述に対しソフトウェアパターンの解決策を反映できるか

RQ2：生成 AI のプロンプトにソフトウェアパターン名を指定することで，設計内容の反映を効率的に行えるか

RQ3：特定の言語モデルによらずソフトウェアパターンの解決策を設計書に反映できるか

3. 実験

3.1. 実験方法

我々は前述の RQ を検証するため，ソフトウェアパターンの一種であるアーキテクチャパターンを対象に，当該パターンの解決策を設計書の記述に対して反映可能か実験を行った．設計書は，本実験向けに作成した航空便検索システムにおけるフロントエンドとバックエンドモジュールの機能説明を記述したサンプルである．生成 AI に入力するのは 800 字から 1000 字程度の設計書の文章であり，出力された文章を確認してアーキテクチャパターンの解決策が反映されているか判断する．実験するアーキテクチャパターンは，Repository パターン，Service Stub パターン，Cache Aside パターンの 3 種類であり，それぞれ ISO25010 の品質特性／副特性における保守性，試験性，性能効率性の向上が期待される．これらは，航空便検索システムの設計に対する親和性と，対象とする品質特性／副特性を多様化させることを考慮して選定した．

なお，本論文の実験はプロンプトにより設計書にアーキテクチャパターンの解決策が反映できるかについて評価するものであり，パターンの反映の結果として実際にドキュメントやコードの品質が向上したかどうかは評価しない．

各々の RQ に対応する 3 つの実験をした．各実験の概要を以下に示す．

【実験1(RQ1)】

設計書に記述されたモジュールに対し，特定のアーキテクチャパターンの解決策を反映するようにプロンプトで指示し，当該パターンが反映された設計書の記述が出力されるか確認する．生成 AI は，ChatGPT GPT4 を使用する．

【実験2(RQ2)】

以下に示す 4 つの方法で作成したプロンプトを最初に入力し，その後対象のアーキテクチャパターンの解決策が設計書に反映されるまでに入力したプロンプトの回数を計測した．方法1が最も抽象的なプロンプトであり，数字が大きくなるにつれてより具体的なプロンプトとなっている．なお，方法4は実験1と同じ指示方法である．これらの方法は，対象モジュールやアプローチを明確にすることで，パターン反映までのプロンプト実行数を削減可能か確認することを目的に設計した．生成 AI は，ChatGPT GPT4 を使用する．

方法1：システムについてある品質特性／副特性を向上させる方法を聞く方法

方法2：対象モジュールのある品質特性／副特性を向上させる方法を聞く方法

方法3：対象モジュールのある品質特性／副特性を向上させるアーキテクチャパターンの候補を聞く方法

方法4：対象モジュールに特定のアーキテクチャパターンを適用するように指示する方法

【実験3(RQ3)】

実験1の内容について，ChatGPT GPT4 以外の言語モデルを用いてアーキテクチャパターンの解決策が反映可能か確認する．本実験では，以下の生成 AI を使用して RQ を検証した．

表 1 アーキテクチャパターンの解決策の反映結果(実験 2)

	方法1	方法2	方法3	方法4
Repositoryパターン	○8	○2	○2	○1
Service Stubパターン	△4	△2	○2	○1
Cache Asideパターン	○2	○2	○2	○1

(○:反映可, △:反映可(課題あり), 数字:反映までに要したプロンプトの数)

- ChatGPT GPT3.5
- Google Bard(PaLM2)
- Claude 3 Opus
- Claude 3 Sonnet

3.2. 実験結果

3.1 に記述した 3 つの実験結果を以下に示す.

【実験1】

いずれのパターンについても, パターン名を指定して設計書の変更を指示することで, パターンの解決策を反映したモジュール構造の記述を出力できることを確認できた.

【実験2】

本実験でアーキテクチャパターンの解決策の反映を試行した結果を表 1 に示す. 表 1 に, アーキテクチャパターンの解決策の反映可否と反映までに要したプロンプトの回数を記述した. なお, 表に記述したプロンプトの回数は, 反映までに要した入力回数であり, 仕様書の記述をフォーマットなどで整形する際のプロンプトの回数は含めていない. 本実験を通じ, 具体的なパターン名を指定することで, プロンプトの入力回数を減らすことができたことから, 効果的にアーキテクチャパターンの解決策を反映できることが分かった. ただし Service Stub パターンは, 反映できたものの一部意図と異なる出力を得た(表中△).

方法 1 で作成したプロンプトは特に要求が曖昧であるため, 最終的に目的のアーキテクチャパターンの反映までに要したプロンプトの回数は増加した. それぞれのパターンについて特記事項を以下に示す.

Repository パターンに関しては, 全ての方法においてデータベースへのアクセスを責務とするモジュールを追加した構成を出力させることができた. ただし, 方法 1 のプロンプトでは保守性を向上させる方法の候補が多数存在したため, ChatGPT が Repository パターンを提案して解決策を反映するまでに多くのやり取りが発生した.

Service Stub パターンに関しては, 全ての方法において実際のデータベースアクセスを模擬するモジュールを追加する方針が出力された. しかし, 方法1と2のプロンプトではスタブではなくモックオブジェクトと表現しており, スタブとモックの違いを区別せずに変更を反映していたため△の判定とした.

Cache Aside パターンに関しては, 全ての方法においてデータベースアクセスをキャッシュするモジュール構造を出力させることができた.

パターンの解決策を反映するよう設計方針を明確に示した場合でも, 反映の方法は一意に定まらないため, ChatGPT から出力される結果もまた変動する. Cache Aside パターンの場合, キャッシュ情報の照会処理を, フロントエンドからの要求を受け取るリクエスト処理モジュールが担当するか, データベースアクセスモジュールが担当するかの責務の判断が異なり, 出力結果が変動した.

【実験3】

ChatGPT GPT3.5, Google Bard, Claude 3 Opus, Claude 3 Sonnet いずれも 3 種類のアーキテクチャパターンを認識し, パターンの概要を出力可能であることを確認した.

ただし, ChatGPT GPT3.5 は GPT4 と比較すると, 設計書の反映の精度は劣る傾向が確認された. 例えば, Repository パターンの場合, パターンの概要は出力されるものの設計書のモジュール構成に反映ができず, 複数回同一のプロンプトを再実行する必要あった. また, 更新された設計書の記述に関しても, GPT4 と比較すると抽象的な記述となっているものが多く見られた. 例えば, Repository パターンの場合, パターン名を認識し変更するモジュールを判別することはできたが, 設計内容は「データベースアクセスを Repository パターンを使用して更新」のみ出力され, 詳細な内容を得られなかった.

Google Bard に関しては, 本実験に関しては ChatGPT GPT3.5 と比較すると具体的な記述を生成しているように見られた. しかし, 出力形式に関しては元の設計書記述を無視して箇条書きで出力したり, 要求していないサンプルコードを出力するため, 意図した設計書記述を取得するのが困難な場合があった.

一方, Claude 3 Opus に関しては, ChatGPT GPT4 と同等の精度で反映ができているように見られた. アーキテク

チャパターンを認識し，意図したモジュールに対して設計書の記述として反映結果を出力していることを確認できた．対して，速度重視の Claude 3 Sonnet は ChatGPT GPT3.5 や Google Bard と同等の精度のように見られた．Claude 3 Sonnet もアーキテクチャパターンの認識はできたが，モジュール名がクラス名のように出力されたり，反映後のモジュール名に揺らぎが起こったりすることが確認された．

3.3. RQ の検証

3.2 に記述した実験結果を踏まえ，各 RQ について検証する．

RQ1：生成 AI は設計書の記述に対しソフトウェアパターンの解決策を反映できるか

実験1の結果から，ChatGPT GPT4 は 3 種類のアーキテクチャパターンを認識し，当該パターンの設計内容を設計書に反映できることを確認できた．ただし，今回はサンプルの設計書に対して 3 種類のアーキテクチャパターンのみ実験したため，今後は実際の設計書に対して他の複数のソフトウェアパターンの解決策が反映可能か検証する必要があると考える．また，設計書だけでなくソースコードに対してソフトウェアパターンを適用可能かについても検証に値すると考える．

RQ2：生成 AI のプロンプトにソフトウェアパターン名を指定することで，設計内容の反映を効率的に行えるか

実験2の結果から，たとえパターン名を知らなくても，ChatGPT GPT4 とのやり取りでどの対象についてどのような方法を採用するか候補を絞り込むことで，対象ソフトウェアの設計品質を向上させることができると考える．しかし，パターン名を知らない場合は，要求する変更を反映するまでに多くのやり取りが発生するため，効率面では劣る．設計品質を向上させるには，どの対象に対してどの品質を作りこむ必要があるかという明確な意思をプロンプトに記述することが重要であり，そのためにはソフトウェア設計や用語の知識が必要となると言える．

一方で，生成 AI は目的を達成するための手段を複数提示するため，検討していない他の有用な手段を知ることができる可能性がある．例えば，Cache Aside パターンの場合，ChatGPT はキャッシュの概念の適用の他に，データベースのインデックスの設定やリードレプリカの導入を提案した．ソフトウェアパターンを適用した設計に対し，意図的に抽象的なプロンプトを入力することで，検討から漏れていた品質改善施策がないか確認する活用方法もあると考える．

RQ3：特定の言語モデルによらずソフトウェアパターンの解決策を設計書に反映できるか

実験3の結果から，実験で使用したいずれの生成 AI においても，アーキテクチャパターンを認識することは確認できた．ただし，適切に設計書に反映させる点においては，ChatGPT GPT3.5 や Google Bard，Claude 3 Sonnet については，プロンプトの工夫や人手による生成文章の修正が多く求められると考えられる．

4. 設計品質向上に向けたプロンプトパターンの考察

生成 AI に関しては，意図通りの出力を得るために効果的なプロンプトを作成することを目的とした，プロンプトパターンが多く発表されている．例えば，ChatGPT に関するプロンプトパターンカタログ[9]や，コード品質の改善やリファクタリングなどに役立つプロンプトパターン[10]が発表されている．

我々は，3 章に記述した実験において，ソフトウェアパターンの解決策を反映するプロンプトを作成した経験から，生成 AI にソフトウェアの設計品質の向上に寄与する施策を出力させるために効果的なプロンプトの記述方法を考察した．結果として，暗黙的にプロンプトに適用していた記述の型を抽出し，ソフトウェアの設計品質を向上させるためのプロンプトパターンを 4 つ作成した．以下に，作成したプロンプトパターンの概要とパターン記述について説明する．なお，実験におけるプロンプトで，各プロンプトパターンに示した例と同様の文章を使用している．

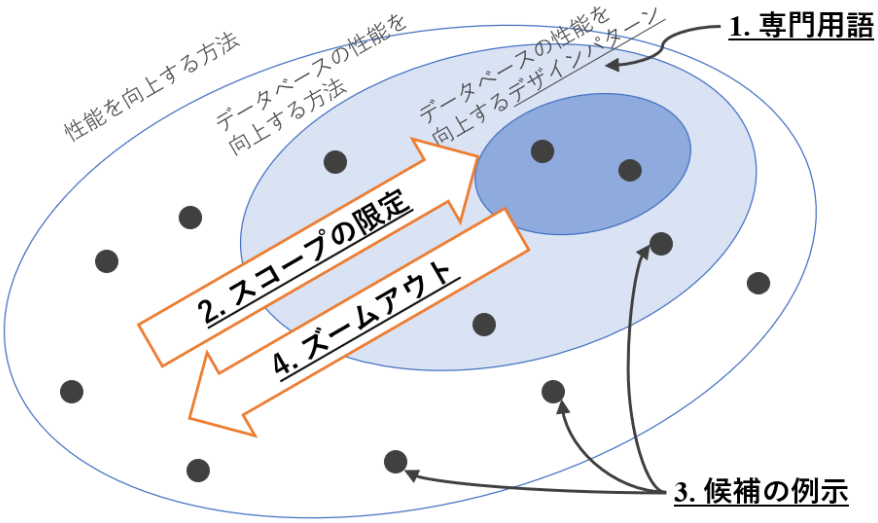

図 1 提案プロンプトパターンの概要図

4.1. パターンの全体像

作成したプロンプトパターンは，**専門用語**，**スコープの限定**，**候補の例示**，**ズームアウト**である．4 つのプロンプトパターンの概念を図式化したものを図 1 に示す．図はプロンプトにより設計品質を向上させる施策が抽出されるスコープを表現したもので，"性能を向上する方法"，"データベースの性能を向上する方法"，"データベースの性能を向上するデザインパターン"の順にスコープが縮小し，出力される候補が絞られる．"デザインパターン"のように**専門用語**を使用することにより**スコープの限定**が実現される．その際，プロンプトで**候補の例示**をするように指示すると，複数の選択肢から有用な施策の候補が出力されるため効率的である．最後に，**ズームアウト**によりスコープを広げることで，検討から漏れた施策がないか確認しておくと，さらなる品質向上を期待できる．

4.2. パターン記述

本論文で提案する4つのパターン記述を以下に示す．

Pattern 1：**専門用語**(Technical Term)

ソフトウェアの設計ドキュメントを作成している．

＜その状況において＞

プロンプトの内容が抽象的過ぎて出力される結果も抽象的となり，一向に設計品質の向上に繋がる修正が達成できない．または，意図していない施策を反映させてしまい，却って品質の低下を招く結果になる可能性もある．

＜そこで＞

ソフトウェア開発の専門用語や設計品質の向上に役立つ施策を知っているようであれば，その単語をプロンプトに入力して回答を誘導させよう．

プロンプトの例：

- 「○○の保守性を向上させるアーキテクチャパターンの例を挙げてください」
- 「○○に Repository パターンを適用してください」

＜その結果＞

具体的な専門用語をプロンプトで指定することにより，意図通りの出力を得られるようになる．その結果，設計品質を向上させる設計を，効率よくドキュメントに反映させることができる．このパターンは**スコープの限定**や**ズームアウト**のように，プロンプトの対象範囲の調整に役に立つ．

関連パターン：

- Write Clear Instructions[3]

Pattern 2：**スコープの限定**(Scope Limitation)

ソフトウェアの設計ドキュメントを作成している．

＜その状況において＞

設計の品質が向上するように漠然とドキュメントの修正を指示しても，一向に期待通りの出力が得られない．

＜そこで＞

意図したとおりの出力を得るために，漠然とした指示ではなく，目的や適用したい対象などを指定することで，スコープを絞った質問をしよう．スコープを絞るために，具体的なモジュールの名前を指定したり，**専門用語**をプ

ロンプトに入れることが有効である.

プロンプトの例:

- 「データベースの時間効率性を向上する施策の例を挙げてください」
- 「データベースアクセスモジュールに Cache Aside パターンを適用してください」

＜その結果＞

意図したとおりに設計ドキュメントを修正することができ, 効率よく品質を向上させる施策を反映させることができる.

関連パターン:

- Write Clear Instructions[3]
- The Context Manager Pattern[9]

Pattern 3:**候補の例示**(Candidate Examples)

ソフトウェアの設計ドキュメントを作成している.

＜その状況において＞

品質向上につながる具体的な設計施策を思いつくことができない. しかし, 具体的な施策をプロンプトで指示しないと, 誤った修正をドキュメントに施してしまう可能性がある.

＜そこで＞

初めに, 品質を向上させる手段の候補をいくつか出力させ, その中から適切な手段を成果物に反映させよう. その時に, 有用な候補を出力させるため, **専門用語**などを使って**スコープの限定**したプロンプトを入力するのが効果的である.

プロンプトの例:

- 「○○の保守性を向上させるアーキテクチャパターンの例を挙げてください」
- 「データベースの時間効率性を向上する施策の例を挙げてください」

＜その結果＞

提案された候補の中から有用な施策を選択することで, 品質を向上させる設計をドキュメントに反映することができる.

関連パターン:

- Split Complex Tasks into Simpler Subtasks[3]

Pattern 4:**ズームアウト**(Zoom Out)

スコープの限定したプロンプトを入力することで, 設計品質を高めたソフトウェアの設計ドキュメントを作成した.

＜その状況において＞

作成した設計ドキュメントについて, 本来必要な検討が漏れてしまっており, 最終的に品質の悪い製品を開発してしまう可能性がある.

＜そこで＞

さらなる品質向上のために, 観点や対策の抜け漏れがないか, あえて抽象的にプロンプトで質問して確認しよう. ただし, 抽象的過ぎると的外れな回答ばかり出力されるかもしれないため, **専門用語**などを使って適度な抽象度のプロンプトを作ろう. この時, **候補の例示**をするようにプロンプトを作成すると, その後の適用がスムーズになる.

プロンプトの例:

- 「保守性を向上させる方法の例を挙げてください」
- 「○○パターンの他に, ○○の保守性を向上させる方法の例を挙げてください」

＜その結果＞

考慮が漏れていた設計観点を知ることができ, さらなる設計品質の向上が期待できる.

関連パターン:

- The Alternative Approaches Pattern[9]

4.3. 既存プロンプトパターンとの関連

作成したプロンプトパターンには, 関連する既存のプロンプトパターンが存在する. 既存のプロンプトパターンは, 様々な文脈に汎用的に適用可能な内容を記述していると考える. 一方で作成したプロンプトパターンは, 設計品質向上を目的としたものであり, 既存パターンの内容を特化した記述になっている. 例えば専門用語パターンは, 明確な指示をプロンプトに記述することを提案する Write Clear Instructions について, ソフトウェア開発における技術用語を使用して指示を明確にすることを提案しており, 解決方法がより具体的になっている. 作成したプロンプトパターンは, 設計品質向上の文脈において具体化された問題と解決策を記述している点が, 既存のプロンプトパターンとの差異である.

5. おわりに

本論文では, 生成 AI を利用してアーキテクチャパターンを設計書に反映する実験結果を記述した. 実験の結果, 生成 AI がアーキテクチャパターンを認識し, 設計書の記述に対象のパターンの解決策が反映可能であることを確認した. また, 本実験を通じて得られた知見から, ソフトウェア設計品質を向上させるためのプロンプトパターンを作成した.

今後の展望を以下に記述する.

- 実際の設計ドキュメントに対するソフトウェアパターンの適用
- 他のソフトウェアパターンの適用可否の検証
- ソースコードに対するソフトウェアパターンの適用可否の検証

謝辞

本研究を進めるにあたり, 有意義な議論の場を提供頂いた一般財団法人日本科学技術連盟に深く御礼申し上げます. また, 議論の場において多くの有益な意見を頂戴した研究会の皆様に感謝いたします.

参考文献

[1] The economic potential of generative AI: The next productivity frontier, https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier (2024/01/05 参照).
[2] Prompt Engineering Guide, https://www.promptingguide.ai/ (2024/01/05 参照).
[3] OpenAI: Prompt engineering, https://platform.openai.com/docs/guides/prompt-engineering (2024/01/05 参照).
[4] 飯泉紀子, 鷲崎弘宜, 誉田直美, ソフトウェア品質知識体系ガイド(第 3 版)－SQuBOK Guide V3－, オーム社, 2020.
[5] L. A. Chauvet et al, "ChatGPT as a Support Tool for Online Behavioral Task Programming", 2023.
[6] P. Vaithilingam et al., "Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models", CHI EA '22, 2022.
[7] D. Sobania et al., "An Analysys of the Automatic Bug Fixing Performance of ChatGPT", 2023.
[8] A. Ahmad et al., "Towards Human-Bot Collaborative Software Architecting with ChatGPT", EASE '23, 2023.
[9] J. White et al., "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT", 2023.
[10] J. White et al., "ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design", 2023.