

ソフトウェアエンジニアリングの 海外動向調査

2026 年 4 月

独立行政法人情報処理推進機構

目次

第1章 はじめに	5
1.1 AI 時代におけるソフトウェアエンジニアリングの変化	5
1.2 本調査が着目する 5 つの課題	5
1.3 本調査が分析する 3 つの観点	7
第2章 調査の目的	9
2.1 目的	9
2.2 調査目標	9
第3章 調査の方法と対象範囲	10
3.1 調査手法	10
3.2 対象範囲	10
3.3 分析上の基本方針	11
3.4 分析上の留意点	11
第4章 組織設計	12
4.1 経営戦略	12
4.1.1 レガシーとモダナイゼーション	12
4.1.2 内製化	13
4.1.3 開発プロセス	15
4.2 組織構造	16
4.2.1 共通基盤	16
4.2.2 OSS	17
4.2.3 体制	18
4.2.4 AI 時代を前提とした組織構造	19
第5章 スキル体系・人材育成	22

5.1 役割の変化.....	22
5.2 スキル体系.....	23
5.2.1 上流スキル.....	23
5.2.2 下流スキル.....	24
5.2.3 横断的スキル.....	24
5.3 育成モデル.....	25
5.4 個人外要素.....	27
5.4.1 組織戦略.....	27
5.4.2 組織基盤.....	27
5.4.3 開発プロセス.....	28
5.4.4 OSS・標準化.....	28
第6章 開発プロセス変革.....	29
6.1 開発プロセスの見直し.....	29
6.1.1 反復プロセス.....	29
6.1.2 導入効果.....	30
6.2 上流工程での AI 活用.....	31
6.2.1 品質保証（QA）の高度化.....	31
6.2.2 要件分析の高度化.....	32
6.2.3 設計の高度化.....	32
6.3 下流工程での AI 活用.....	34
6.3.1 コードレビュー.....	34
6.3.2 テスト.....	35
6.4 LLM のアウトプットの高精度化.....	36
第7章 結論と今後の展望.....	39

7.1 調査目標に対する結論	39
7.2 今後の展望	41
参考文献	42

第 1 章 はじめに

1.1 AI 時代におけるソフトウェアエンジニアリングの変化

近年に公表された海外の調査・研究からは、AI 支援開発、クラウドネイティブ、OSS 活用の進展が、近年のソフトウェアエンジニアリングを特徴づける潮流として繰り返し報告されている。DORA の「State of AI-assisted Software Development 2025」 [1]では、AI はソフトウェア開発に広く浸透し、生産性向上に寄与し得る一方で、ソフトウェアデリバリーの不安定性や検証負荷の増加も伴い得ると整理されている。また、AI の主要な役割は、組織の強みと弱みを増幅する“amplifier（増幅器）”であり、AI 投資の効果はツールそのものではなく、基礎となる組織システムや開発能力に左右されると報告されている。

CNCF の 2025 年年次クラウドネイティブ調査（2025 年 9 月実施、回答 628 名） [2]では、クラウドネイティブ技術の採用が広く進んでおり、成熟度の高い組織ほど CI/CD、GitOps、自動化、頻繁なコードチェックイン、日々リリースを実践している一方で、最大の障害は技術的複雑性よりも開発チームにおける文化的変化に移っていると整理されている。ここからは、ソフトウェア開発の競争力が、技術導入の有無だけでなく、それを支える共通基盤、運用規律、組織文化の成熟度に依存していることが読み取れる。

また、OSS 活用についても、Linux Foundation の調査 [3]では、多くの組織が OSS を重要システムの基盤として利用している一方で、正式なガバナンスや戦略、組織的な受け皿の整備はなお限定的であることが報告されている。OSS 活用の巧拙は、部品を使えるかどうかではなく、それを支えるガバナンスや組織的関与の成熟度に左右されつつある。

また、LeadDev の調査 [4]では、AI の導入効果を測る際に、局所的な速度や出力量そのものではなく、それが事業成果や実際の利用価値に結びついているかを捉えることが、最大の課題として認識されている。AI によってコード生成やデプロイ自体が高速化する時代には、「作る速さ」がボトルネックでなくなる分、むしろ何を作るべきか・作ったものが使われているのかという価値検証の重要性が相対的に高まっている。したがって、速度や出荷量だけでなく、ユーザーの利用・定着に関する指標を技術指標と並べて把握する視点が重要になる。要するに、AI 時代のソフトウェアエンジニアリングでは、何を速く作るかではなく、何を価値に結び付けて継続的に届けるかが重要な論点になりつつあると整理できる。

1.2 本調査が着目する 5 つの課題

本調査は、国内におけるソフトウェア活用の課題認識を出発点としつつ、その課題を解く手がかりを海外の動向に求めるものである。本節では、国内資料（ソフ

トウェアモダナイゼーション委員会報告書 [5]、ソフトウェア開発データ白書 2025 [6]、DX 動向 2025 [7]) が示す国内の課題を整理したうえで、それぞれを、本調査が海外動向を読み解くための論点として設定する。各論点は、第 4 章以降の分析対象と対応している。

課題 1：レガシーと知識の属人化

ソフトウェア開発データ白書 2025（自由記述コメントサマリー）では、OS やハードウェアのサポート切れに伴うシステム更新の負担や、技術・業務の属人化、特定個人に依存した運用体制、ナレッジ継承の困難が、社会全体の共通課題として挙げられている。本調査では、この課題を、レガシー資産への依存と、それを支える知識構造・意思決定のあり方が、システム刷新をどのように制約し、海外ではどのような対応が試みられているかを読み解く論点として設定する（第 4 章）。

課題 2：内製化

DX 動向 2025 [7]では、システム開発等の内製化の状況が調査項目の一つとして扱われている。本調査では、この課題を、開発・運用の主導権と実行能力を組織内部に保持・再構築するために、海外ではどのような条件が整えられているかを読み解く論点として設定する（第 4 章）。

課題 3：俊敏性

DX 動向 2025 [7]では、変化に迅速に適応するためのアプローチであるアジャイルの取り入れ状況について、日本は米国・ドイツに比べて大きく遅れており、「全面的に取り入れている」の回答割合がすべての部門で 1 割以下にとどまることが示されている。本調査では、この課題を、環境変化への迅速な適応と継続的な価値提供を可能にする開発プロセスが、海外ではどのように構築されているかを読み解く論点として設定する（第 4 章、第 6 章）。

課題 4：エコシステムと外部接続

ソフトウェアモダナイゼーション委員会報告書では、社会全体での連携・標準化や共通基盤の整備、OSS をはじめとするコミュニティへの参加が、今後取り組むべき事項として挙げられている。本調査では、この課題を、組織横断・外部連携を支える共通ルール、横断組織、相互運用の仕組みが、海外ではどのように制度化されているかを読み解く論点として設定する（第 4 章）。

課題 5：顧客価値創出への接続

DX 動向 2025 [7]では、国内の DX の目的がコスト削減・業務効率化を中心とし、米国・ドイツが売上・利益の増加といった顧客価値創出を成果の中心に置くのに

対して、国内では顧客価値創出への接続が弱いことが示されている。本調査では、この課題を、ソフトウェアを業務効率化にとどめず、顧客価値創出へ接続するために、海外ではどのような構造が機能しているかを読み解く論点として設定する（第5章、第6章）。

1.3 本調査が分析する3つの観点

ソフトウェアの競争力は、どの技術を導入したかという個別技術の選択そのもの以上に、その技術を取り巻く**環境**（意思決定や能力配分を定める組織の設計、担い手のスキルと育成、開発の進め方）をどのように整えるかによって大きく左右される。同じ技術を導入しても、それを支える組織・人・プロセスが伴わなければ、期待した成果には結びつかない。

そこで本調査では、海外動向を「どの技術が普及しているか」という技術単位ではなく、技術を価値へ結びつける環境の側から捉えることとし、その環境を構成する三つの側面（組織設計、スキル体系・人材育成、開発プロセス）を分析の観点として設定する。

1.2 節で整理した5つの課題も、いずれか単一の技術で解決できる問題ではなく、この三つの側面にまたがる構造的な課題である。レガシーと知識の属人化（課題1）、内製化（課題2）、エコシステムと外部接続（課題4）は、意思決定構造や能力配分といった組織のあり方に根ざす問題であり、主として組織設計の観点にかかわる。俊敏性（課題3）は、迅速な適応を可能にする組織設計と、それを支える開発プロセスの双方にまたがる。顧客価値創出への接続（課題5）は、価値を生み出す担い手のスキル体系・人材育成と、価値を継続的に届ける開発プロセスにかかわる。すなわち、5つの課題を貫く共通の要因は個別技術ではなく、それを取り巻く環境の設計にあり、その環境は組織設計・スキル体系/人材育成・開発プロセスの三つに集約できる。

この見立ては、海外の事例・研究からも確認できる。Green Cargo の事例 [8]では、技術的負債は単なる古いコードの問題ではなく、業務適応、従業員の関与、事業継続、外部委託への依存などが関わる社会技術的問題、つまり問題の所在が技術そのものに限定されず、組織構造や運営にも及ぶとして整理されている。さらに、同事例では、開発を外部に委ねる体制を選んだ結果、内部のIT部門が開発機能を失い、保守機能へ縮退していたことが報告されている。また、公共部門の **backsourcing** 研究 [9]では、内製回帰の成否が、どのツールを使うかではなく、誰にどこまで決めさせるか（意思決定構造）と、どのような能力・体制を組むか（組織設計）に左右されることが示されている。

以上を踏まえ、本報告書ではこの 3 つの観点に沿って、組織設計を第 4 章、スキル体系・人材育成を第 5 章、開発プロセスを第 6 章で、それぞれ海外動向を分析する。

表 1 第 1 章における参考資料の位置づけ

資料	本調査の問題意識形成における役割
ソフトウェアモダナイゼーション委員会 報告書（2026 年 3 月 24 日） [5]	ソフトウェア価値最大化の必要性という課題認識を与える基礎資料
ソフトウェア開発データ白書 2025 グラフ編 [6]	ソフトウェア開発の現状を定量的に把握するための基礎資料
ソフトウェア開発データ白書 2025 コメント編 [6]	上記の定量傾向を、現場実務や背景事情の観点から補足する基礎資料
DX 動向 2025 [7]	成果創出の方向性を「内向き・部分最適」から「外向き・全体最適」へ捉え直す視座を与える基礎資料

第2章 調査の目的

本章では、第1章で整理した国内課題および海外動向を踏まえ、本調査の目的と調査目標を整理する。

2.1 目的

本調査の目的は、海外のソフトウェアエンジニアリングの現状を分析し、AI時代において顧客価値創出を実現している構造を特定することであり、これは最終的に、豊かなデジタル社会の実現に資する知見を得ることを目指すものである。したがって、1.2節で説明した5つの課題を出発点として、その手がかりを海外動向に求める。

2.2 調査目標

本調査の目的、および1.3節で示した3つの観点（組織設計、スキル体系・人材育成、開発プロセス）を踏まえると、各観点は分析の「切り口」とどまらず、AI時代において顧客価値創出を実現している構造を特定するという目的に照らし、それぞれ「何を最大化・最適化する構造か」という調査目標へと具体化できる。すなわち、組織設計についてはレガシーモダナイゼーションと開発アジリティの最大化を、スキル体系・人材育成については生産性と品質の最大化を、開発プロセスについては開発ライフサイクル全体の最適化を、それぞれ実現する構造を明らかにすることが目標となる。したがって本調査では、次の3点を調査目標として設定する。

1. レガシーモダナイゼーションと開発アジリティを最大化する組織設計（経営戦略・組織構造）を明らかにする。
2. 生産性と品質を最大化するスキル体系および育成モデルを特定する。
3. 開発ライフサイクル全体の最適化（品質・コスト・速度）を実現するプロセスモデルを明らかにする。

以降、この三つの調査目標に対応する形で、組織設計を第4章、スキル体系・人材育成を第5章、開発プロセスを第6章で、それぞれ海外動向を分析する。

第3章 調査の方法と対象範囲

3.1 調査手法

本調査は、ソフトウェアエンジニアリングに関する主要カンファレンスおよび主要研究機関を対象とし、第1章で整理した5つの課題、および第2章で示した3つの観点（組織設計、スキル体系・人材育成、開発プロセス）に基づいて、最新動向の机上調査を実施する方法を採る。調査対象の中には、学術会議、研究機関、産業団体、OSSコミュニティ、企業・実務コミュニティが含まれ、論文、調査レポート、ケーススタディ、ガイドブック、実務記事等を横断的に収集・整理する。

本調査において机上調査を採る理由は、第一に、海外の議論は学術論文だけでなく、調査レポートや実務ガイド、コミュニティ発信の中にも重要な論点が存在するためである。第二に、ソフトウェアモダナイゼーション委員会やDX動向2025が示す問題意識を踏まえると、制度・組織・開発現場・外部エコシステムの接続まで含めた広い射程で比較する必要があるためである。第三に、本調査では、単なる事例収集ではなく、比較可能な整理軸を意識することが必要となるためである。

また、本調査では、第2章で示した3つの調査目標、すなわち

- 組織設計（経営戦略・組織構造）
- スキル体系・人材育成
- 開発プロセス

に対応する形で、論点を整理する。

3.2 対象範囲

本調査では、ソフトウェアエンジニアリングに関する主要な研究機関・標準化団体、学術会議・カンファレンス、実務コミュニティ・イベントを幅広く調査の起点とした。調査対象は、次の2つの区分で整理する。

表2 調査対象の整理方法

区分	内容	本報告書での役割
前提資料	ソフトウェアモダナイゼーション委員会 報告書、ソフトウェア開発データ白書 2025、DX 動向 2025	日本側の問題意識と調査設計の基礎
海外文献・記事	論文、調査レポート、ケーススタディ、実務記事	各章の分析対象

このうち、前提資料は日本の現状や問題意識を与える資料であり、そこに記載されていない具体的事実や数値を本文で勝手に補うためのものではない。一方、海外文献・記事は、第 4 章以降で各論点を分析するための直接的な根拠資料である。さらに、対象範囲をこのように分ける理由は、国内の問題意識の整理と海外動向の分析とを混同しないためである。たとえば、DX 動向 2025 は、日本企業と米独企業の DX 成果の差や、部分最適と全体最適の差を示す比較資料であるが、本調査の第 4 章以降では、それを前提にしたうえで、海外文献から組織設計や開発プロセスの構造を分析する、という役割分担をとる。

3.3 分析上の基本方針

本調査では、個別文献の要約をそのまま並べるのではなく、複数資料を比較し、共通点・補完関係を整理することを基本方針とする。具体的には、各論文・記事・調査レポートについて、研究目的、手法、主な結果、そこから読み取れる海外動向、課題との関係、活用できそうな示唆を整理し、そのうえで目標別に再構成する。

3.4 分析上の留意点

本調査には、少なくとも 2 つの留意点がある。

第一に、学術論文、調査レポート、実務記事、ケーススタディでは、方法論の厳密さや射程が異なる。そのため、本報告書では、個々の資料の性格を踏まえつつ、単独資料からの過度な一般化を避ける。

第二に、目標 1・2・3 は相互に関係しているが、本報告書ではまず章ごとに切り分けて分析する。このため、第 3 章時点では、調査の射程を広く持ちつつも、各章の担当範囲を明確に保つことを重視する。

以上の方針に基づき、第 4 章では組織設計、第 5 章ではスキル体系・人材育成、第 6 章では開発プロセスについて、それぞれ海外動向の比較分析を行う。

第4章 組織設計

本章では、第2章で示した3つの調査目標のうち、目標1「組織設計」に対応する海外動向を調査する。具体的には、第1章で整理した5つの課題のうち、レガシーと知識の属人化（課題1）、内製化（課題2）、俊敏性（課題3）、エコシステムと外部接続（課題4）を、組織設計の観点から読み解く。これらの課題は、いずれも個別技術の導入だけでは解決できず、経営戦略と組織構造の設計に深く関わるためである。

本章では、調査結果を「経営戦略」と「組織構造」の二つの観点から整理する。経営戦略の観点では、「レガシーとモダナイゼーション」、「内製化」、「開発プロセス」が主要な論点として確認された。また、組織構造の観点では、「共通基盤」、「OSS」、「体制」、「AI時代を前提とした組織構造」が重要な論点として確認された。

4.1 経営戦略

調査の結果、経営戦略の観点では、「レガシーとモダナイゼーション」「内製化」「開発プロセス」が、相互に独立した論点ではなく、連続した関係の中で捉えられていることが確認された。その典型的な入口の一つがレガシー依存への対応であり、事業要求や制度・環境の変化への対応が難しくなる中で、組織内部に開発実行能力（要求整理・設計・実装・運用改善）をどの程度保持・再構築するかが重要な論点となり、さらに、その能力を継続的に機能させるために開発プロセスの設計が重要になる、という関係である。ただし、この関係の起点は組織の状況によって異なり、必ずしもレガシーの存在を前提とするものではない [1, 2, 8, 9, 10, 11, 12, 13, 14]。

このため、海外で確認された経営戦略上の特徴は、個別の技術導入を進めることよりも、レガシーへの対応、組織的な開発実行能力の保持、継続的な実行の仕組みを一体として設計する点にあると整理できる [8, 9, 10, 12, 13, 14]。

4.1.1 レガシーとモダナイゼーション

レガシーとモダナイゼーションに関する文献を総合すると、レガシーの問題は、単に古いシステムが残っていることではなく、それを前提として固定化した知識構造や意思決定のあり方にあることが示されている。とりわけ、Green Cargo／California Management Review のケース研究 [8]では、技術的負債は、コードやアーキテクチャの老朽化だけでなく、知識の属人化、文書不足、外部依存、既存業務を優先する意思決定と結びついて、変化を困難にする構造的問題として整理されている。

この点は、レガシー資産を新しい仕組みに単純に置き換えれば解決する問題ではないことを示している。同研究 [8]では、旧来の仕組みを一括して置き換えるのではなく、既存資産と新しい仕組みをつなぐ中間的な基盤の整備、段階移行、共通基盤の先行整備を通じて、既存資産への依存を徐々に減らす進め方が確認されている。また、米国国防総省（DoD）のソフトウェアモダナイゼーション実装計画 [12]でも、ソフトウェアモダナイゼーションは一度限りの刷新ではなく、継続的な実装計画として扱われている。これらを合わせると、本研究・資料群では、レガシーモダナイゼーションは一括更改よりも、段階的に依存関係を切り替える形で進められる傾向が確認された。

その一例として、MITRE Corporation の LLM 活用研究論文 [15]では、MUMPS や ALC のようなレガシー言語に対して、レガシーコードに関する説明生成や文書化支援が一定の有効性を持つことが示されている。

ただし、この研究 [15]は、文書化やコード理解の支援可能性を示すものであり、移行期間短縮やコスト削減などのモダナイゼーション成果を直接検証したものではない。したがって、この研究が補っているのは、刷新そのものの成功を示すことではなく、刷新の前提となる知識外在化や移行準備の支援であると位置づけるのが適切である。

以上を総合すると、レガシーとモダナイゼーションに関する海外の知見は、レガシー依存を技術更新だけの問題としてではなく、知識の外在化、段階移行、共通基盤整備、意思決定の見直しを伴う経営戦略上の課題として捉えている点で共通している。

4.1.2 内製化

内製化に関して本調査で参照した文献群を総合すると、内製化は、単に「外部委託を減らすこと」ではなく、ソフトウェアの開発・運用に関する主導権と実行能力を組織内部に取り戻し、それを継続的に維持できる状態を目指す取組として捉えることができる。とりわけ、Lassenius らのバックソーシングに関する実証研究論文 [9]では、バックソーシングは、従来外部に委ねていた活動を組織内部へ戻し、権限と責任を再び内部に置く過程として整理されている。

この点を最も直接的に示しているのが、Lassenius らのバックソーシングに関する実証研究論文 [9]である。同研究では、内製化の主な契機はコスト削減であったが、あわせてシステム開発・運用に対するコントロールを取り戻すことも重要な目的として確認されている。また、当該事例では、各アプリケーション領域への裁量付与、ベンダーとの協調、段階的な引継ぎ、アジャイル開発の並行導入、内部人材の確保と能力構築などが、実際の進め方を構成する重要な要素として示されて

いる。したがって、内製化は、契約形態の変更そのものではなく、開発責任と改善能力を内部へ再配置する組織的取組として理解するのが適切である。

また、米国国防総省（Department of Defense : DoD）関連の 2025 年報告書群は、内製化を直接扱うものではないものの、内部のソフトウェア実行能力を継続的に維持・拡張するために必要な組織条件を示している。たとえば、Carnegie Mellon University Software Engineering Institute（SEI）の DevSecOps 実態調査報告書 [10] では、75 名超のリーダー・実務者に対する調査を通じて、19 のソフトウェア組織・試験組織を対象に実態を把握しており、ソフトウェア・ファクトリーの拡張やモダナイゼーションには、人材・プロセス・技術への投資判断に加え、組織横断の調整や文化面の対応も重要であることが示されている。

同様に、米国国防総省最高情報責任者室（Department of Defense Chief Information Officer : DoD CIO）のソフトウェアモダナイゼーション実装計画 [12] では、ソフトウェア取得経路の採用拡大、ソフトウェア人材群の整備、DoD のサイバー人材フレームワークにおける 8 つのソフトウェア工学職務ロールの位置づけと追跡が示されている。ここからは、ソフトウェア能力の確保が、単なる人員配置ではなく、職務定義、育成、配置、制度、調達方法を含めた継続的な仕組みとして扱われていることが分かる。

さらに、DoD CIO の DevSecOps ガイドブック [13] は、共通の活動や基準を示しつつも、各採用組織が自らの要件や任務上の必要性に応じて内容を調整し、それぞれのプログラムが自らに適したプロセス、活動、ツールを定義すべきであることを明記している。このことは、内部能力の整備が、一律の統制によってではなく、共通部分を明確にしつつ、各領域が自らの条件に応じて適用する前提に立っていることを示している。

また、バックソーシングの実証研究では、3 年間で 100 超のシステムを内製化し、300 人超のアジャイル開発組織を構築した結果として、システムに対する当事者意識の向上、人材の意欲向上、リリース高速化、利用時品質の改善が報告されている一方で、大幅なコスト削減を示す明確な証拠は確認されていない。したがって、内製化の成果は、短期的な費用削減だけでなく、主導権の回復、改善速度、品質、継続的な実行能力の向上という観点から把握することが適切である。

以上から、本調査で参照した文献群からは、内製化を単なる外部委託の縮小ではなく、開発・運用に関する主導権と実行能力を組織内部に再構築し、それを継続的に機能させるための戦略的取組として捉えるのが適切である。また、その実現に当たっては、明確な方針、内部人材の確保と育成、段階的な移行、ベンダーとの協調、共通部分の明確化と各領域における調整の両立が重要な論点になると整理できる。

4.1.3 開発プロセス

開発プロセスに関する文献を総合すると、開発プロセスは、工程を順に並べた手順書としてではなく、変化への対応と継続的な価値提供を支える実行能力として扱われている。この傾向は、DoD 関連の DevSecOps 資料 [10, 13]において、開発から運用までを通じた活動・責任・ツールの組合せとして整理されていることから確認できる。また、DORA の AI 支援開発レポート [1]でも、出力量だけでなく、不安定性や検証負荷も含めて評価する必要があることが示されている。したがって、開発プロセスは、速度のみを追うのではなく、品質、セキュリティ、運用、価値提供を含めた継続的デリバリー能力として扱う方向が確認されたと整理できる。

その際、開発プロセスは、全チームに一律の工程を固定的に適用するものとしてではなく、共通化すべき活動や基準を基盤として設計するものと整理できる。

DoD CIO の DevSecOps ガイドブック [13]では、共通の活動群とベースラインを示しつつも、各プログラムが自らの事情に応じてプロセス、活動、ツールを定義・調整する考え方が示されている。また、Eindhoven University of Technology らの大規模アジャイル組織に関する事例研究論文 [11]では、標準、ルール、評価指標といった公式な統制と、実務者同士が部門横断で知見やノウハウを共有する場などの非公式な調整を組み合わせるものの有効性が示されている。これらを踏まえると、開発プロセスは、活動・責任・基準を明確化したうえで、共通部分を標準化しつつ、各組織・各環境の条件に応じて調整適用する形で設計されると整理できる。

さらに、Cloud Native Computing Foundation (CNCF) の年次調査報告書 [2]は、共通部分を標準化しつつ現場条件に応じて調整適用する開発プロセス設計の差が実践の成熟度の差として表れていることを示している。成熟度の高い組織では、GitOps、CI/CD、自動化、コード変更の共有リポジトリへの頻繁な反映、日次リリースが進んでいる。一方で、成熟度の低い組織では、多様で複雑な技術スタック、長年蓄積した技術的負債、リスク回避的な文化、分散した意思決定が導入の障害として挙げられている。したがって、開発プロセスの差は、単なるツール導入の差ではなく、共通基盤、標準化、文化、統治の差として現れていると考えられる。

以上から、開発プロセスに関する海外の知見は、開発プロセスを個別工程の改善ではなく、共通部分を標準化しつつ、組織の事情に応じて調整適用し、継続的に価値を提供するための実行能力として捉えている点で共通している。

なお、本節では開発プロセスを組織設計・経営戦略の観点から位置づけるにとどめ、AI を組み込んだ開発プロセスの具体的な工程（上流・下流での AI 活用やプロセスの再設計）については、第 6 章で詳しく扱う。

4.2 組織構造

調査の結果、組織構造の観点では、「共通基盤」「OSS」「体制」「AI 時代を前提とした組織構造」が、それぞれ独立した論点として存在しているのではなく、相互に関連しながら整理されていることが確認された。すなわち、組織内部で再利用可能な能力を共通基盤として整備し、そのうえで外部エコシステムとの接続を制度化し、さらにそれらを継続的に機能させるための役割配置や運営体制を設計するとともに、AI の利用を前提として、共通基盤、統治機能、現場の実行組織の役割分担を再設計する、という流れである。

このため、海外で確認された組織構造上の特徴は、個別チームの努力に依存するのではなく、共通化すべき能力、外部と接続すべき機能、横断的に担うべき役割を明確にすることにあると整理できる。

4.2.1 共通基盤

共通基盤に関する文献を総合すると、共通基盤は単なるインフラ共用の仕組みではなく、開発組織全体で再利用可能な能力を安定的に供給するための構造として位置づけられている。

この点を最も具体的に示しているのが、LY Corporation の社内 PaaS 事例記事 [16] である。同事例では、個別チームがそれぞれ仮想マシンや Kubernetes クラスタを管理する状態から、共通基盤を担う組織が基盤機能を集約し、開発者は簡素化されたインタフェースを通じてそれを利用する構造へ移行している。ここでは、共通基盤が単に技術資産をまとめるためではなく、開発者がアプリケーション開発や価値創出に集中できるようにするための組織的仕組みとして扱われている。

この整理は、DORA のプラットフォームエンジニアリング解説記事 [14] や、Platform Engineering Community / Humanitec の調査報告書 [17]、Sodexo の内部開発者プラットフォーム事例記事 [18] とも共通している。これらの資料では、社内の開発者向け共通基盤は、開発者が必要な機能を自ら利用できること、標準的なやり方を繰り返し再現できること、組織として推奨する安全で効率的な実装手順を利用できることを支えるものとして整理されている。つまり、共通基盤は、個別プロジェクトごとの改善を積み上げるのではなく、組織全体に共通する実行能力を先に整えるための仕組みとして理解されている。

さらに、DoD 関連の DevSecOps 資料 [10, 13] を補助的に読むと、ソフトウェア・ファクトリーや DevSecOps 活動を通じて、標準運用、セキュリティ要件、共通的な活動、継続的改善を組織横断で展開する考え方が示されている。したがって、共通基盤は、インフラの一元化というよりも、開発組織全体で共通に保持すべき能力を集約し、安定的に供給する構造として捉えるのが適切である。

以上から、共通基盤に関する海外の知見は、共通基盤を単なる技術基盤ではなく、再利用可能な能力を組織全体へ供給し、個別チームの実行力を底上げするための組織構造として位置づけている点で共通している。

4.2.2 OSS

OSSに関する文献を総合すると、OSSは単なる技術選択やコスト削減手段としてではなく、外部エコシステムとの接続を制度化するための組織的機能と結びついて扱われている。

この点を最も広い視野で示しているのが、Fraunhofer ISST/Data Spaces Support Centre (DSSC) の支援センター資料 [19] と、Data Spaces Support Centre (DSSC) のブループリント資料 [20] である。これらの資料では、複数主体が協調するためには、技術そのものだけでなく、共通ルール、共通語彙、ガバナンス権限、役割分担、相談窓口や支援の仕組みが必要であることが示されている。ここで重要なのは、外部との連携は「使うかどうか」の問題ではなく、誰が何を決め、どのような支援を行うかまで含めて設計されるという点である。

この考え方は、Linux Foundation Research らの OSPO¹ 調査報告書 [21] や Linux Foundation の Global OSS 調査報告書 [3] ともリンクする。OSPO 調査では、OSPO は従来のコンプライアンス管理機能にとどまらず、AI、セキュリティ、OSS 本体への貢献、開発のしやすさまで含めて調整する横断機能として位置づけられている。一方、Global OSS 調査では、OSS 活用そのものは広がっていても、正式なガバナンス、明確な戦略、組織的な受け皿の整備はなお限定的であることが示されている。つまり、OSS の利用が広がるだけでは不十分であり、それを支える横断機能がなければ、外部エコシステムとの継続的な連携は持続しにくい。

さらに、Linux Foundation の OSS 貢献 ROI 調査報告書 [22] では、OSS 本体への貢献やコミュニティへの継続的参加が、セキュリティ課題やバグ報告への応答性、製品開発速度、ロードマップへの影響力、重要変更の事前把握、人材採用・リテンションと結びついて整理されている²。ここから、OSS 参加は本務の周辺的な活動ではなく、競争力や価値創出に関わる投資として扱われ始めていることが分かる。

¹ 企業や組織がオープンソースソフトウェア（OSS）を効率的かつ効果的に活用・管理するために設置される専門部門である。

² 日本でも、独立行政法人情報処理推進機構（IPA）が、企業における OSS 活用推進体制の整備を支援する取組として、The Linux Foundation TODO Group の協力の下、企業向け「OSPO レベル 1 構築ワークショップ」を開催している。

このように、共通基盤が組織内部の再利用能力を整える構造だとすれば、OSSは組織外部との接続を整える構造として位置づけられる。DSSCが協調のためのルールや支援を示し、OSPO調査とGlobal OSS調査が組織的受け皿の必要性を示し、ROI調査がその投資合理性を補っている。以上から、OSSに関する海外の知見は、OSSを単なる利用対象ではなく、外部エコシステムとの接続を継続的に成立させるための組織構造として捉えている点で共通している。

4.2.3 体制

体制に関する近年の海外文献を総合すると、ソフトウェアエンジニアリングにおける体制は、中央集権的に全てを統制する形でも、各チームに全てを委ねる形でもなく、継続的なソフトウェア提供を担う実行単位と、それを組織として支える仕組みとを組み合わせで設計するものとして捉えられている。SEIのDevSecOps実態調査報告書[10]では、DevSecOpsへの移行は、単なるツール導入ではなく、組織文化、スキルセット、プロセス、継続的な開発・運用のための予算措置の仕組み、組織間の関係に及ぶ変革として整理されている[10]。また、ソフトウェア・ファクトリーは、特定の利用者のニーズに応じて継続的に価値を届けるための人材、ツール、プロセスの組合せとして定義されている。これらを踏まえると、体制に関する主要な論点は、個別チームの努力だけで完結させるのではなく、継続的な提供を支える条件を組織としてどのように整えるかにあると整理できる。

また、これらの文献では、体制は抽象的な統治論としてだけではなく、役割構成を伴う実行体制として描かれている[23]。米国国防総省 Office of Systems Engineering and Architectureの継続的デリバリーに関する技術報告書[23]では、開発チームは、コーディング、テスト、サイバーセキュリティなどを含む職能横断的な構成であることが示されている。加えて、DoD Software Factoryにおける職位例として、プロダクトマネージャー、プロダクトオーナー、プロダクトデザイナー、ソフトウェア品質エンジニア、プラットフォームエンジニア、セキュリティエンジニア、データエンジニア、品質・テストエンジニアなどが提示されている[23]。したがって、近年の文献からは、体制を単に部門配置の問題としてではなく、継続的デリバリーに必要な役割を、誰がどのように担うかを組織として設計する傾向が読み取れる。

そのうえで、これらの文献は、一律の標準を固定的に適用することだけを推奨しているわけでもない[13]。DoD CIOのDevSecOpsガイドブック[13]では、各組織は自らの要件や任務上の必要性に応じて内容を調整することが推奨され、活動は「必須」、「推奨」、「必要に応じて」、に区分されている[13]。すなわち、共通に求められる事項を明示しつつも、具体的な適用には人間の判断が残されている[13]。これを踏まえると、近年の文献では、体制は共通に担保すべき事項を組

織として整えつつ、実行単位が条件に応じて調整できる形として理解するのが適切である。

表 3 体制設計における共通機能と実行単位の役割

観点	組織として共通に担保する事項	実行単位が担う事項
方針・基準	開発・運用・セキュリティに関する共通方針、最低限の基準、必須活動の明確化	プロダクトや業務特性に応じた具体的な適用方法の調整
役割	プロダクトマネージャー、品質、セキュリティ、プラットフォーム等の必要役割の定義	各チーム内での役割分担、兼務・専任の判断
プロセス	DevSecOps、継続的デリバリー、レビュー、テスト等の標準的な考え方の整備	チームの成熟度やシステム特性に応じたプロセス運用
人材・スキル	必要スキルの定義、育成、配置、キャリアパスの整備	実務を通じたスキル獲得、チーム内での知識共有
ツール・基盤	共通ツール、共通基盤、標準的な開発環境の提供	必要に応じたツール選択、設定、利用方法の最適化

以上から、体制に関する近年の海外文献からは、ソフトウェアエンジニアリングの体制を、共通基盤の整備の有無だけで捉えるのではなく、継続的な提供を担う実行単位、必要な役割構成、ならびに方針・共通に求める基準・役割・人材を組織として支える仕組みを、どのように組み合わせるかという観点から捉える傾向が確認できる。すなわち、体制の論点は、中央集権か分権かの二者択一ではなく、共通に扱うべき事項と、各実行単位の判断に委ねる事項とをどのように切り分けるかにあると整理できる。

4.2.4 AI 時代を前提とした組織構造

AI 時代を前提とした組織構造は、新たに AI 専任部署を設置することそのものではなく、ソフトウェア開発ライフサイクル全体において、人間と AI の役割分担、横断的な支援機能、統治機能、人材育成のあり方をどのように再設計するかという課題として捉えるのが適切である。Carnegie Mellon University Software Engineering Institute (SEI) の AI 拡張ソフトウェアエンジニアリング解説資料 [24]および Amalfitano らの生成 AI とソフトウェアエンジニアリングに関する研究ロードマップ論文 [25]では、AI は実装工程のみを局所的に支援する技術としてで

はなく、要求、設計、実装、検証、運用、継続的改善までを含むソフトウェア開発プロセス全体に影響を及ぼし得るものとして扱われている。また、生成能力の向上に加えて、監督、検証、トレーサビリティといった論点が重要になっている [24, 25]。

第一に、AI 時代の組織構造では、現場の実行組織に求められる役割が変化すると整理できる。Butler らの生成 AI コーディングツールに関する実証研究論文 [26] および Microsoft Research の生成 AI と高スキル労働に関する実証研究論文 [27] では、AI ツールの利用によって開発者の完了タスク数が増加することや、日々の業務実感に前向きな変化が見られることが報告されている。一方で、AI 生成コードの信頼性に対する見方は同様には向上しておらず、AI の出力に対しては、引き続き慎重な確認と批判的な評価が必要であることが示されている。これらを踏まえると、AI 時代の現場組織では、AI を補助的に活用しながら、人間によるレビュー、品質確認、批判的評価の重要性が引き続き高いと整理できる [26, 27]。

第二に、AI 時代の組織構造では、現場の実行を支える横断的な支援機能が重要になる。ただし、その役割は個別チームの開発を代替することではなく、各チームが安全かつ効率的に AI を活用できるよう、標準的な推奨経路、セルフサービス型の利用環境、標準化されたワークフロー、導入支援を提供することにある。Cloud Native Computing Foundation (CNCF) のプラットフォームエンジニアリング記事 [28] では、プラットフォームエンジニアリングは、標準的な推奨経路、セルフサービス型インフラ、標準化されたワークフローを通じて、複雑性を過度に高めずに開発を拡大するための実践として位置づけられている。また、内部開発者向けプラットフォームは、環境やサービスのセルフサービス提供、観測性、セキュアなワークフロー、テンプレート、チーム間の一貫性を支えるものとして説明されている。したがって、AI 時代の横断機能は、現場の自律的な実行を支える支援機能として設計する方向が重視されていると整理できる [28]。

第三に、AI 時代の組織構造では、統治機能の比重が増す。AI を用いた開発では、生成物の品質だけでなく、データ来歴、トレーサビリティ、セキュリティ、コンプライアンス、OSS や外部供給網への依存といった論点が、個別チームの裁量だけでは十分に扱えなくなる。DoD CIO のソフトウェアモダナイゼーション実装計画 [12] では、部門横断のソフトウェア能力強化の一環として、DevSecOps における AI 導入の枠組み、データ来歴、AI トレーサビリティ要件などが位置づけられている。また、Linux Foundation の Global OSS 調査報告書 [3] では、多くの組織がオープンソースを重要システムの基盤として利用している一方で、それを安全に管理するためのガバナンスやセキュリティの枠組みが十分ではないことが示されている。これらを踏まえると、AI 時代の組織構造では、品質基準、利用方針、セキュリティ、法務、OSS、供給網リスクを横断的に調整する統治機能を、継続的な開発実行を成立させるための重要な機能として位置づける必要がある [3, 12]。

第四に、AI 時代の組織構造では、人材像と育成の考え方も見直す必要がある。**Amalfitano** らの生成 AI とソフトウェアエンジニアリングに関する研究ロードマップ論文 [25]では、生成 AI がソフトウェアプロセスの実行方法や、ソフトウェアの開発・運用・進化に影響を及ぼすことが整理されている。また、**Linux Foundation Research/Linux Foundation Education** の技術人材レポート [29]では、AI や関連技術分野における人材不足が顕在化しており、組織として能力向上を重視する傾向が示されている。したがって、AI 時代の組織構造は、従来の職種区分のまま AI ツールを配布するだけでは不十分であり、AI を活用する実行組織を支える育成の仕組みや学習支援を、組織として計画的に整備する必要がある [25, 29]。

表 4 AI 時代の組織構造における主要機能

機能	主な役割	重視される論点
現場の実行組織	AI を活用しながら、要求整理、設計、実装、検証、運用を行う	AI 出力の確認、人間によるレビュー、品質判断
横断的な支援機能	AI 活用環境、標準ワークフロー、テンプレート、導入支援を提供する	セルフサービス化、標準化、利用しやすさ
統治機能	AI 利用方針、品質基準、セキュリティ、法務、OSS、供給網リスクを調整する	トレーサビリティ、コンプライアンス、リスク管理
人材育成機能	AI を前提としたスキル定義、学習支援、能力向上を設計する	AI 活用能力、批判的評価能力、継続的学習

以上から、AI 時代を前提とした組織構造は、AI ツールの導入有無や専任部署の設置有無のみで捉えるべきではない。重要なのは、現場の実行組織が AI を活用しつつも人間によるレビューと判断を維持すること、横断機能が標準的な利用環境と導入支援を供給すること、統治機能が品質・セキュリティ・法務・OSS・供給網を継続的に調整すること、そしてこれらを支える育成の仕組みを組織として統合的に設計することである。すなわち、AI 時代の組織構造とは、AI ツールを追加することではなく、AI を前提とした開発実行能力を組織全体に埋め込むための役割分担を再設計することと整理できる。

第5章 スキル体系・人材育成

本章では、第2章で示した3つの調査目標のうち、目標2「スキル体系・人材育成」に対応する海外動向を調査する。具体的には、第1章で整理した5つの課題のうち、顧客価値創出への接続（課題5）を、スキル体系・人材育成の観点から読み解く。この課題は、個別技術の導入だけでは解決できず、顧客価値創出を実際に担うソフトウェアエンジニアの役割変化と、それに応じたスキル体系・育成モデルの設計に深く関わるためである。

調査の結果、「上流スキル」「実務埋め込み型の学習」などが主要なトピックとして多く挙げられており、これらは世界的な潮流となっていることが確認された。

5.1 役割の変化

海外の実証研究・調査に共通するのは、生成AIやAIエージェントの普及によって、ソフトウェアエンジニアの価値の中心が「コードを直接書く量」から「適切な文脈を与え、出力を評価し、組織の制約に沿って統合すること」へ移っている点である。

例えば、Fraunhofer IESE における LLM 支援開発に関する研究 [30] では、「通常の形で記述された要件は、そのままでは LLM への入力として抽象的すぎる」ため、開発者がまず要件を実装可能なタスクへと分解し、さらに設計判断やアーキテクチャ上の制約を付加してからプロンプトに落とし込む必要があると指摘している。これは、AI 時代において重要なのが単純なコーディング速度ではなく、要件・設計意図・制約条件を構造化し、AI が利用可能な形に変換する上流的な能力であることを示している。

実務調査でも同様の傾向が確認されている。LeadDev の調査 [4] では、AI 時代に最も需要が高まる能力として「critical thinking (43%)」と「architectural design (34%)」が挙げられた一方で、「writing code」は 8%にとどまった。これは、ソフトウェアエンジニアに期待される中心的価値が、コードを書く作業そのものから、AI の出力を吟味し、設計判断を下し、全体最適の観点から成果物を統合する役割へ移行していることを示唆している。

Microsoft Research 等による現場実験 [27] では、生成 AI は開発者の生産性を改善し得る一方で、その効果は一様ではなく、経験差や業務特性によってばらつくことが示されている。経験の浅い層で相対的な改善が大きいという示唆はあるが、それは「熟練が不要になる」ことを意味しない。むしろ、複雑・曖昧・制約の多い実務では、熟練者ほど問題を分解し、設計意図や制約を明示し、出力を検証・修正することで AI の効果を引き出している。

また、Microsoft Research 開発者と AI エージェントの協働研究 [31]では、現実のソフトウェア工学タスクでは人間が最終判断者であり続けることが確認されている。これは複雑で曖昧な実タスクにおいて、AI エージェントで一度に解決するより、ソフトウェアエンジニアが AI エージェントと議論を交わしながら段階的に課題を解決する方が課題解決の成功率が高かったことから確認されている。特に、要件が曖昧で複数の解法があり、既存システムとの整合性が重要な場面では、エンジニアは「実装者」よりも、問題設定者、コンテキスト設計者、検証者、意思決定者としての役割を強く担う。

これらを総括して、役割変化の本質は、自動化そのものではなく、人間の責務がより上位の判断・統合・説明へシフトする点にあると整理する。

表 5 ソフトウェアエンジニアの役割の変化

項番	従来	AI 時代	実務上の含意
1	実装（コーディング）	課題分解・文脈設計・出力統合	仕様、制約、判断根拠を明示して AI に渡す能力が重要になる
2	個人作業中心	人間と AI の協働設計中心	AI との役割境界、AI 出力のレビュー手順を設計する必要がある
3	後工程での品質確認	AI 生成直後からの検証	テスト、レビュー、セキュリティ確認を前倒しで組み込む必要がある
4	暗黙知での対応	説明可能な判断・記録	なぜ AI がその出力を採用したかを説明し、再現可能にする必要がある

5.2 スキル体系

AI 時代のソフトウェアエンジニアに求められるスキルは、単なる「プロンプト能力」のみでは不十分のようである。調査の結果、1.上流で AI に適切な問題を与える能力、2.下流で生成物を検証し運用につなげる能力、3.全体を貫く AI 活用基礎技能・倫理・説明責任・協働能力、の三つが実務上有効であることが確認された。以下では、この三層構造に沿って調査結果を詳細に整理する。

5.2.1 上流スキル

第一に重要なのは、問題設定・要求分析・仕様化・アーキテクチャ判断といった上流スキルである。これは 5.1 節でも言及したとおりである。先に述べた Fraunhofer IESE の研究 [30]では、開発者は要件を実装可能なタスクへ分解し、設計判断、非機能要件、アーキテクチャ制約、既存システムとの整合条件を補ってはじめて、AI を有効に活用できるとしている。

したがって、上流スキルの中核は、1.問題を定義する能力、2.曖昧さを減らす能力、3.設計上の制約を言語化する能力、4.評価観点を置いてタスクを分解する能力、5.AI の出力を評価して組織の制約に沿って統合する能力、の五点と整理できる。特に、3、4、5 の能力は、AI を安全かつ再現可能に活用するための入力・評価・検証の枠組みを設計する力であり、ハーネスエンジニアリング力ともいえる。これは従来の上流工程スキルの延長線上にあるが、AI 時代には「人が自分のために考える仕様」ではなく、「人と AI が共有可能な作業可能表現へ変換する能力」として再定義される。

5.2.2 下流スキル

AI 時代においては、AI によって生成されたコード・設計案・テスト案を採用するかどうかの判断と、その後の品質保証といった下流スキルも重要となる。

IBM の企業内調査 [32]では、AI コードアシスタントは生産性や開発体験を改善し得る一方、生成物の所有権や責任所在に関する懸念が示されている。Carnegie Mellon 大学の SEI の保守研究 [33]では LLM 利用時に学生が正確で再現可能な解を安定して得られず、かつ AI を過信する傾向が確認されており、AI 活用には厳格な監督が必要だとされている。さらに、Gothenburg の企業・大学共同でのコードレビュー研究 [34]では LLM 利用には懸念（偽陽性や信頼性の問題）が伴い、その有効性はレビューアのコードベース理解や変更の重大度に依存すると報告されている。加えて北京大学のセキュアコードレビュー研究 [35]ではセキュリティレビューにおける LLM ベースの既存手法の課題とハルシネーション対策、評価指標の必要性を指摘しており、Meta の LLM ベースのテスト生成における研究 [36]でも生成されたテストの採否やプライバシー関連性は最終的にエンジニアが判断している。これらに共通するのは、AI だけでは品質や安全性は担保されず、人間がレビュー観点を設定し、例外やリスクを見抜き、証拠に基づいて採否を判断することが不可欠だという点である。

したがって、下流スキルの中核は、1.出力コード等の妥当性検証能力、2.再現性確認能力、3.セキュリティ評価能力、4.本番適用可否の判定、5.運用時のフィードバック収集活用能力、の五点と整理できる。これらを踏まえて、AI 時代には、テストやレビューの専門職だけでなく、AI 利用者全員が最低限の検証責任を負う前提であることを念頭に置く必要がある。

5.2.3 横断的スキル

AI 活用を実務に埋め込むうえでは、技術的技能に加えて、批判的思考、倫理判断、説明責任といった横断的スキルが不可欠である。

まず、AI 活用基礎技能として、これまで述べてきたものを含め、適切なコンテキスト付与、段階的なプロンプト設計、小さなバッチでの反復、内部文書・コード・データの参照可能性を踏まえた作業設計が必要である。DORA が示す能力群 [37] は個人能力だけでなく組織能力を含むが、個人レベルでは「AI に何を、どの粒度で、どの根拠とともに渡すか」を設計する能力として読み替えることができる。

Calgary 大学の研究 [38] では、好奇心、批判的思考、共感、倫理的責任が、AI を含む開発において重要な基盤になることが示されている。

また、Gothenburg の CSE (Department of Computer Science and Engineering) の AI ソフトウェア工学の研究 [39] では、AI の出力及びその背後にある意思決定プロセスと根拠に対する透明性と説明責任を支えるには、倫理と技術をまたぐ H-shaped competencies (倫理・技術の複合能力) と、それを支える組織文化が必要とされている。

したがって、横断的スキルの中核は、1.AI と人間の役割境界と協働を意識した AI 活用基礎技能、2.AI 出力を鵜呑みにしない批判的思考、3.利用者・顧客・社会への影響を考慮する倫理判断、4.判断過程を記録し説明する能力、の四点と整理できる。これらは補助的能力ではなく、AI 時代における品質と信頼の基盤として位置付けられる。

表 6 AI 時代のスキル体系

項番	体系区分	主な内容	人材育成上の示唆
1	上流スキル	問題定義、要求分析、仕様化、制約言語化、アーキテクチャ判断、タスク分解、コンテキスト設計、出力評価、出力統合	従来の上流工程スキルの延長として、AI への INPUT 情報の構造化能力も育てる
2	下流スキル	コード評価、レビュー観点抽出、テスト戦略、再現可能性の確保、セキュリティ評価、本番適用可否判断、運用/監視/フィードバック	AI 利用者全員に最低限の検証責任を持たせる
3	横断的スキル	AI 活用基礎技能、批判的思考、倫理判断、説明責任	評価制度と日常運用に組み込み、付随要素ではなく中核能力として扱う

5.3 育成モデル

AI 時代のソフトウェアエンジニアの育成方法について、単発のツール研修では不十分であり、実務に埋め込まれた継続学習への転換が求められている。

George Mason 大学及び Notre Dame 大学の共同研究 [40]は、LLM が学習支援や個別化に有効である一方、従来型カリキュラムでは不十分になりつつあり、評価方法・授業設計・課題設計の見直しが必要であることを示している。Adobe Systems も同様に、従来のコンピュータサイエンス教育の課題を見直し、暗記や単独実装に偏った教育から、AI 時代により適した高次の実践能力を重視するカリキュラムへの転換を提案している [41]。

特に重要なのは、知識注入型の講義よりも、実際の業務や演習の中で、問題分解、AI との対話、生成物の検証、振り返りを反復する学習モデルである。California 大学、Sevilla 大学及び Aalto 大学のチームスキル教育に生成 AI を足場として活用する研究 [42]は、コミュニケーション、要件確認、対人調整といった従来教えにくかった能力の訓練可能性を示している。

これらと 5.2 節のスキル体系の整理も踏まえ、育成モデルは大きく四段階で設計することが望ましいと考えられる。

1. 全員共通の基礎訓練として、AI 利用方針、情報管理、基本的なプロンプト設計、検証手順、記録の取り方の学習
2. 職種別・役割別に、上流、実装、品質、運用、マネジメントの各能力の育成
3. 実案件やそれに近い演習で、人間と AI の協働を前提にしたレビュー・振り返り・改善ループ
4. 評価制度や昇格基準とリンクしたスキル体系の継続的な更新

これは企業内育成にもそのまま当てはまり、短期的にはツール習熟、長期的には思考様式と仕事の進め方の転換も検討する必要があると考えられる。

表 7 AI 時代の育成モデルにおける重要観点

項番	観点	内容	実現イメージ (例)
1	実務埋め込み型継続学習	日常業務の中で AI 活用・検証・振り返りを回す	ペア作業、レビュー会
2	役割別学習	上流・下流・横断能力を職種・等級別に分ける	アーキテクト向け制約記述訓練、QA 向け検証演習
3	シナリオベース学習	曖昧な要件、品質事故、セキュリティ懸念を題材に学ぶ	ケース演習、模擬障害対応
4	評価制度の刷新	新しいスキル体系を人事評価	スキルマップ更新、評価制

項番	観点	内容	実現イメージ（例）
	新	や昇格基準に結び付ける	度更新

5.4 個人外要素

個人のスキル定義と育成だけでは、AI 時代の開発成果は安定しない。5.3 節でも言及したとおり、組織内の評価の見直しも含め、個人外要素として、少なくとも組織戦略、組織基盤、開発プロセスの三つを一体で設計する必要がある。加えて、OSS コミュニティへの貢献や標準化動向への追従も、持続的な能力形成に影響する。

5.4.1 組織戦略

AI 導入の目的が曖昧なままでは、現場は「とりあえず使う」状態に陥りやすい。LeadDev の調査 [4]では、AI 導入は予算、評価、採用、人材パイプライン、プロセス設計にまで影響しており、経営・事業・技術の統合が必要であるとしている。したがって、どの領域で AI を活用し、どの価値指標で測るかを組織として定め、その上で定義したスキル体系と育成・評価をリンクする必要がある。

5.4.2 組織基盤

個人が優れていても、参照可能な文書やデータ、標準的な開発環境、個人を取り巻く文化、が整っていないければ、効果的な AI 活用は再現しない。

DORA は、AI 導入の主たる効果を **amplifier**（増幅器）として捉え、AI は組織の強みも弱みも拡大すると整理している [1]。したがって、個人の生産性向上のみを論じるのではなく、組織能力と結び付けて理解する必要がある。特に重要な組織能力として、明確で浸透した AI ポリシー、健全なデータエコシステム、AI が参照可能な内部データ、強力なバージョン管理、高品質な内部プラットフォーム等が示されている [37]。

また、DoD の DevSecOps 研究は、効果的なリーダーシップと、イノベーション、協力、継続的学習を優先する文化は、DevSecOps の能力を大規模に提供できる人材を育成するために不可欠であるとしている [10]。先に紹介した Gothenburg の CSE の AI ソフトウェア工学の研究 [39]では、透明性と説明責任を支えるための組織文化も必要としている。これは個人の善意に依存する話ではなく、誰が何を決め、どの根拠で採用し、どのリスクを受容したかを説明できる状態を組織として設計するという意味である。

これらが整った組織基盤の実現例として、European Commission では「AI Gov Tech エコシステムの強化」「技能投資、行政内イノベーションチームの設立等に

よる政府の能力構築」「調達プロセスの整備と小規模企業の参加促進」「人間中心かつ倫理的な AI 活用の徹底と、透明性・説明責任・公平性の確保」を提言し、2025 年 10 月に Apply AI Strategy を開始している [43, 44]。

5.4.3 開発プロセス

ソフトウェアエンジニアの役割変化を踏まえ、AI 利用を前提にした開発プロセスの各工程で、どこまでを AI に任せ、どこで人が確認・判断し、どの証跡を残すかを設計する必要がある。また、先に紹介した Microsoft Research 等による現場実験 [27]や Carnegie Mellon 大学の SEI の保守研究 [33]が示したように単に AI 利用をするだけでは特に若手において基礎的理解の空洞化も生じうることと、5.3 節で整理した実務埋め込み型継続学習が育成モデル上の重要観点であることを踏まえ、開発プロセスと一体で育成モデルを考える必要がある。

5.4.4 OSS・標準化

AI 時代のスキル形成は、社内だけで完結しない。OSS コミュニティへの参加・貢献、セキュアオープンソースの実践、標準化活動、外部の評価フレームワークの活用を通じて、品質・セキュリティ・説明責任の共通言語を持つことができる。OpenSSF の 2025 年 セキュアオープンソースソフトウェアビジョン概要 [45]では、OSS のセキュリティ確保には、脆弱性対応の個別最適ではなく、コミュニティ横断の協働、共通フレームワーク、実践知の共有が必要であることが示されており、これが重要なのは、OSS の安全性が今や一組織の内部管理を超えて、広範な社会基盤の安定性と説明責任に直結しているためである。組織としても外部標準やコミュニティ知見を取り込み、教育内容と運用ルールに反映させることが、属人的な AI 活用を避けるうえで有効であると考えられる。

第 6 章 開発プロセス変革

本章では、第 2 章で示した 3 つの調査目標のうち、目標 3「開発プロセス」に対応する海外動向を調査する。具体的には、第 1 章で整理した 5 つの課題のうち、俊敏性（課題 3）と顧客価値創出への接続（課題 5）を、開発プロセスの観点から読み解く。これらの課題は、いずれも個別ツールの導入だけでは解決できず、AI を前提とした開発ライフサイクル全体の再設計に深く関わるためである。

調査の結果、「AI ツールの導入」よりも「AI を前提に工程を再設計すること」が主要なトピックとして多く挙げられており、特に小さな反復、上流 QA の前倒し、下流の自動化範囲と人間の最終判断範囲の再整理、LLM 出力の高精度化が世界的な潮流となっていることが確認された。

6.1 開発プロセスの見直し

海外の主要研究・実務報告に共通するのは、AI 導入を単なるコーディング支援として扱う段階から、開発ライフサイクル全体を再設計する段階へ移行している点である。第 5 章で述べたとおり、AI の効果は組織能力の「増幅器」として現れ、内部プラットフォームやデータ基盤、バージョン管理といった基盤の有無が成果を左右する。実際、LeadDev の調査でも、先進企業は AI 投資の中心を個人ツールよりも内部エンジニアリングプロセスに置いており、導入効果の測定が最大の課題として認識している [4]。

さらに、CNCf、DoD、Carnegie Mellon 大学の SEI の報告は、生成 AI を実運用に載せるには、モデル単体ではなく、トレーサビリティ、テスト、セキュリティ、運用監視、責任分界を含む基盤的プロセスが不可欠であると指摘する [10, 12, 46, 47]。Fraunhofer IESE も、AI 活用の論点が「コード補完」から「AI が関与する SDLC³全体の制御と品質保証」へ移っていると総括している [48]。

6.1.1 反復プロセス

この変化の中核は、工程を大きな一回限りの受け渡しで進めるのではなく、要件の補足、生成、検証、修正、再生成を短いサイクルで回す反復プロセスへの移行である。Princeton 大学、MIT 等は、実務観察に基づいて、ソフトウェアエンジニアの AI 利用を「プロンプトする→出力を点検する→修正する→必要なら手作業へ戻る」という反復ワークフローとして定式化している [27]。ここで重要なのは、

³ SDLC（Software Development Life Cycle）：要件定義・設計・実装・テスト・リリース・運用・保守に至る、ソフトウェア開発の一連の工程（ソフトウェア開発ライフサイクル）を指す。

AI 利用が単発の問い合わせではなく、生成結果を評価しながら逐次的に軌道修正する活動として捉えられている点である。同様の観点で、DORA は小さなバッチと安全網の整備を重視し、Microsoft Research の研究でも、複雑で曖昧な実タスクでは一括生成よりも、人間が段階的に文脈（コンテキスト）を補いながら AI と対話することで成功率が高くなることを示している [31, 37]。

したがって、AI 時代の反復プロセスは、アジャイルの単純な延長ではなく、AI の生成と人間の評価・修正を往復させながら成果物の精度を高めていく循環的な開発プロセスとして理解する必要がある。

6.1.2 導入効果

開発ライフサイクル全体への AI 導入は、主として三つの効果をもたらしている。

1. 生産性の改善

第一は、生産性、とりわけコーディング速度の改善で、これは比較的一貫して報告されている。Microsoft 系の 3 件の現場実験では、生成 AI 利用により開発者の生産性が向上し、タスク完了数において平均で約 26% の改善が観測された。また、改善幅は経験の浅い層で相対的に大きかった [27]。IBM の企業内調査でも、AI コードアシスタントは開発体験や作業速度の改善に寄与する一方、責任所在や生成物の扱いに関する懸念が残ることが示されている [32]。

ただし、DORA と LeadDev が共通して指摘するように、局所的なコーディング速度の向上は、そのまま品質向上や事業成果につながるとは限らない [1, 4, 37]。要件の曖昧さやレビュー不足が残れば、後工程での手戻りが増え、総コストは逆に上がりうる。したがって、導入効果を評価する際は、コーディング速度だけでなく、レビュー滞留時間、バグ発生率、リリース失敗率、再作業時間といったプロセスメトリクスを同時に追跡することが重要である [49]。

2. 品質確保の前倒し

第二は、品質確保の前倒しである。反復単位が小さくなることで、レビュー、テスト、リスク判定、リリースゲート（制御）を後工程での一括検査ではなく、変更の直後に組み込めるようになる。Meta のリリース展開研究では、コード差分単位で重大障害候補を推定し、危険度に応じてリリースを段階的に制御する運用が示されている [50]。DORA もまた、小さなバッチは AI 利用時の不安定な大規模変更を抑え、速度を製品成果へ結び付ける規律であると位置付けている [37]。すなわち、反復の短縮は単なるスピード向上策ではなく、AI 起因の品質変動を管理するための制御方式でもある。

3. 学習速度の向上

第三は、学習速度の向上である。AI 導入時には、何を AI に任せ、何を人が確認すべきかが組織ごとに異なるため、仮説と検証を繰り返しながら最適な使い方を見つける必要がある。DORA は、この探索を価値流（value stream）と結び付けて行うことを推奨しており [37]、LeadDev の調査でも、実務では評価制度、予算、人材育成まで含めて試行錯誤が進んでいる [4]。言い換えれば、反復プロセスの効果は「AI が速い」ことに由来するのではなく、「小さく試し、小さく直し、小さく出す」ことを継続できる組織能力から生じている。

ただし、導入効果は一律ではない。DORA は、AI が高業績組織の強みを増幅し、低業績組織では摩擦や不安定性も増幅すると報告しており [1, 37]、Microsoft Research 等の現場実験でも、効果は開発者の経験や業務特性によってばらつくことが示されている [27]。また、ブラジルの大規模メディア企業の事例研究では、主にコード集約的な作業では有用だが、創造的・協働的な作業では効果が限定的としている [49]。したがって、日本企業が海外動向を参照する際は、AI ツールの採用率や表面的な成功事例だけを追うのではなく、小さな変更単位、頻繁な統合、短い検証ループ、明確な責任分界といった基礎的な運用様式まで参照し、自社のプロセスを見直すとよい。

6.2 上流工程での AI 活用

上流工程での AI 活用については、中国を中心として、AI を用いて要件の曖昧さを減らし、設計意図を明示し、下流で検証可能な形に変換することに重心が移っている。すなわち、AI は上流を省略するための道具ではなく、上流成果物の形式化・構造化によって品質を高めるための道具として使われ始めている。

6.2.1 品質保証（QA）の高度化

QA においては、下流のテスト専用活動ではなく、以下に示す研究事例のように上流で要件の明確化と検証可能性を高める活動へと再定義されつつある。

- 中国の航空宇宙ソフトウェア向け要件 QA 研究は、LLM を用いて要件に対する質問応答を行い、ゼロショット、few-shot、RAG⁴を比較している。結果は、RAG などの補助的手法により性能は改善されるものの、航空宇宙という専門領域における要求仕様 QA では依然としてハルシネーションが顕著であり、信頼性確保のためには評価指標の整備とともに体系的な分析が不可欠であることを示している [51]。

⁴ RAG（Retrieval-Augmented Generation）：外部の文書やデータベースから関連情報を検索し、その内容を踏まえて回答・生成を行う手法（検索拡張生成）を指す。

- 北京大学とロンドンの UCL の研究は、曖昧な要件記述そのものを修復対象と捉え、テストとプログラム修復を併用して曖昧な要件記述を最小限にすることで、後続のコード生成精度を改善できることを示した [52]。
- 南京大学とシンガポールの大学の研究は、対話的生成とミューテーション⁵に基づく選別を通じて、形式仕様を生成し、要件分析・検証・テストの基盤となる機械可読な成果物を前倒しで得られることを示している [53]。

6.2.2 要件分析の高度化

6.2.1 項でも一部触れているとおり、要件分析そのものの高度化も進んでいる。海外では、LLM を「完成した仕様を書く装置」として使うのではなく、「要件を再利用可能で検証可能な構造へ変換する補助者」として使う傾向が強い。

- Fraunhofer IESE 等の調査では、通常の実要件記述はそのままでは抽象的すぎて LLM 入力として不十分であり、開発者は実装可能なタスクへの分解、設計判断、アーキテクチャ制約、既存システムとの整合条件の付加を行って始めて有効活用できると報告している [30]。
- 北京の大学・研究所等の宇宙産業向けの研究では、形式が統一されていない分散された要件成果物を LLM で統合し、再利用可能な要件コンポーネントとして整理する手法が提示されている [54]。
- Monash 大学等の宇宙産業向けの RAG 研究では、規格・ミッション文書・インタフェース文書から関連コンテキストを抽出しつつ要件草案を生成する手法を示し、手作業の負荷軽減と要件カバレッジ向上の可能性を報告している [55]。

6.2.3 設計の高度化

要件分析だけでなく設計での AI 活用も進んでいる。海外では、LLM を完成形のアーキテクチャを自動的に決定する装置としてではなく、設計候補の生成・比較、既存システムの構造再把握、ソフトウェアモデル⁶の進化支援を行う補助者として使う傾向が強い。すなわち、AI は設計者の代替ではなく、設計意図と制約を明示

⁵ ミューテーションテストの文脈で使われる用語。ミューテーションとは、元のプログラムに小さな変更を人工的に加えることで、既存のテストが本当にバグを検出できるか確かめることを目的として行われる。

⁶ ソフトウェアモデルとは、UML や SysML など記述される、ソフトウェアの構造や振る舞いを抽象的に表現した設計モデルを指す。

し、候補比較とトレードオフ分析を前倒しするための道具として位置付けられ始めている。

- **Carnegie Mellon** 大学の SEI のソフトウェアアーキテクチャ研究は、生成 AI の有効性は設計作業の種類によって異なると整理している。すなわち、既存情報の整理、設計候補の探索、アーキテクチャ再構成のようにコンテキストを収集・要約・比較する作業では有望である一方、利害調整や品質属性間のトレードオフ判断のように説明責任を伴う作業では、人間の最終判断が不可欠であることを示している [56]。
- **Oulu** 大学等のソフトウェアアーキテクチャ分野のマルチボーカルレビュー (MLR) では、生成 AI の適用先は主としてアーキテクチャ決定の支援とアーキテクチャの再構築に集中しており、**Requirements-to-Architecture** や **Architecture-to-Code** のような上流寄りの工程で活用が進んでいることが報告されている。他方で、出力の厳密な評価や設計固有データセットはなお不足しており、設計で AI を使う場合も、設計意図・前提制約・評価観点を成果物として残す運用が必要であるとしている [57]。
- **Saarland** 大学等の研究では、ソフトウェアモデルの履歴と RAG を組み合わせることで、LLM が既存のソフトウェアモデルの進化を支援できることが示されている。実験では、産業データで意味的に正しい補完が 62.30%、型として正しい補完が最大 86.19%に達しており、設計成果物をコード生成の前段で更新・補完する用途に実用可能性があることを示している [58]。
- **Stuttgart** 大学の研究では要件から複数のアーキテクチャ候補を半自動生成し、候補間の比較とトレードオフ分析を支援する構想も提案している。ここで重要なのは、AI が単一の「正解設計」を返すのではなく、複数候補とその判断根拠を提示し、人間が選択・修正する前提で設計プロセスを組み直している点である [59]。

表 8 上流工程での AI 活用

項番	活用対象	主な内容・効果	実務上の留意点
1	要件 QA	要件文書に対する質問応答、整合確認、標準文書との照合を前倒しで実施する。	ハルシネーション抑制のため RAG と人間レビューを併用する。
2	曖昧性修復	あいまいな問題記述を最小限にし、後続のコード生成誤りを低減する。	自動修復結果を設計者が承認する運用が必要となる。

項番	活用対象	主な内容・効果	実務上の留意点
3	形式仕様化	自然言語の要件から検証可能な仕様を生成し、解析・テストに繋げる。	適用範囲を限定し、仕様の妥当性を評価する。
4	再利用・標準準拠	過去の成果物や標準文書を横断的に参照し、統合・整理して再利用可能な要件として再構成する。	ナレッジの更新と出典追跡を維持する必要がある。
5	設計支援・再構成	設計候補の生成・比較、既存のソフトウェアモデル・アーキテクチャの補完や再構成を支援する。	AI 案を比較案として扱い、品質属性・制約・責任分界を人間が明示的に確認する。

6.3 下流工程での AI 活用

下流工程では、AI の役割は実装コードを単に量産することではなく、レビュー、テスト、リリース判定の各工程で高リスク箇所を早く見つけ、人的レビューの焦点を絞り込むことにある。海外では、AI を使った「コードレビュー前処理」「対応優先順位付け」「テスト実行自動化」が進みつつある一方、最終的な採否判断は人間が保持する設計が主流である。

6.3.1 コードレビュー

コードレビュー工程の変革は、AI による一括代替ではなく、レビューの入口を整理し、重要箇所へ注力させる方向で進んでいる。Gothenburg の企業・大学共同でのコードレビュー研究では、LLM はコードレビューのコンテキストの要約や初期指摘に有望であり、レビューの初期段階を支援しうる一方で、偽陽性、コンテキスト理解の不十分さ、出力の信頼性の懸念が残ることが示されており、その有効性はレビューアのコードベース理解や変更の重大度に大きく依存することが示されている [34]。

実務面では、例えば、中国の BitsAI-CR のように、LLM にレビューを全面的に委ねるのではなく、コードの問題候補の抽出と妥当性の検証を分離し、段階的に実施することで誤検知を抑えつつ、運用データのフィードバックによる継続的な改善もすることで、開発者が本当に確認すべきコードの変更点へ注意を向けられる実運用型の自動コードレビュー手法が示されている [60]。北京大学のセキュアコードレビュー研究では LLM にレビュー全体を代行させるのではなく、セキュリティ観点に即した追加学習と外部知識の参照により、開発者が重点的に確認すべき脆弱性候補と修正方針を提示する、セキュアコードレビュー支援の枠組みを示している [35]。

総じて、コードレビュー工程は、(1) AI による要約・初期指摘生成、(2) 人間による採否判断と重大度評価、(3) フィードバックを使ったルール・モデルの継続改善、という三層構造へ変化していつている。この構造では、人間の役割は減るのではなく、より高次の判断へ移る。レビューのボトルネックを減らしつつ品質を保つには、AI 出力をレビューの代替とみなさず、レビュー過程をデータ駆動で改善する契機として扱うことが必要である。

6.3.2 テスト

テスト工程でも、海外では、AI の役割は単なるテストの自動生成を超え、テストの生成・移植・実行・リスク判定までを含むように向かっている。

- テスト生成

Meta の研究では、従来のように大量の疑似バグで網羅的にテストの弱さを測るのではなく、既存テストがまだ見逃している重要な不具合パターンに絞ってテストを自動生成している。実際、プライバシー問題を例に、約 1 万 800 の Android Kotlin クラスから約 570 件のテストを生成しており、Messenger と WhatsApp のテストイベントでは、生成されたテストの 73% が受け入れられている [36]。重要なのは、AI が大量の候補を出すだけでなく、関心の高い故障種別にフォーカスし、人間が採用判断を行う設計になっている点である。すなわち、テスト工程でも「大量生成→人手で整理」ではなく、「リスク仮説に基づく重点生成→採否判断」という方向へ変化している。また、南京大学等の研究では、ユニットテスト生成を含む複数のユニットテスト関連タスクにおいて、適切にファインチューニングされた LLM が既存の最先端手法に匹敵、あるいはそれを上回る性能を示すことを、大規模な実証評価によって明らかにしている [61]。

- テスト移植

California 大学の Android アプリ間のテスト移植研究では、既存の UI テスト資産を LLM により自動再構成・再適用することで、テストの新規作成を大きく代替できる可能性を示している。実際、5 カテゴリ・120 件の移植実験で 97.5% の成功率と 91.1% の手作業削減率を達成しており、既存手法を上回る性能によって、AI がテスト資産再利用の自動化と保守負荷の大幅軽減に寄与しうることを実証している [62]。

- テスト実行

Stuttgart 大学の研究では、多様な言語・ビルド系を持つ任意プロジェクトのテスト実行自体を自動化し、実行環境構築という従来は属人的であった作業に対する AI 活用の可能性を示している。実際、異なる言語・ビルドシステム・テストフレームワークをもつ任意のプロジェクトに対し、ビルドとテスト実行のスクリプト

を LLM エージェントが自律的に準備・改善し、50 プロジェクト中 33 件でテストスイート実行に成功している [63]。

- リスク判定

Meta のリリース・デプロイの研究では、LLM でコード差分単位にリスク予測をすることで、従来モデルより多くの重大障害候補を捕捉できることを示しており、AI が本番前のゲーティングにも入り始めている [50]。

このように、下流工程の AI 活用は「テストを書く工程」の効率化にとどまらず、「何を優先的に試験し、どの変更を止めるか」まで含むリスクベース運用へ広がっている。

表 9 下流工程での AI 活用

項番	活用対象	主な内容・効果	実務上の留意点
1	コードレビュー前処理	変更の要約、論点抽出、既知ルール照合によりレビュー入口を整える。	偽陽性を抑えるためコンテキスト付与と優先度制御が必要となる。
2	セキュリティレビュー	専用モデルや評価指標で脆弱性観点を補強し、危険なコード差分を早期に発見する。	汎用 LLM の回答をそのまま採用しない。
3	テスト生成	ミューテーションやファインチューニングにより高価値テストを生成する。	カバレッジだけでなく欠陥検出力で評価する。
4	テスト資産再利用・実行	既存テストの移植、異種環境での自動実行を支援する。	環境差異と依存関係の管理が必要となる。
5	リリース判断	リスクスコアに基づき危険度の高い差分のリリースを制限する。	最終判断は運用責任者が行う。

6.4 LLM のアウトプットの高精度化

AI を組み込んだソフトウェア開発プロセスにおいて、AI のアウトプットそのものを高精度にしていくことも重要となる。AI、特に LLM を高精度化するうえで重要なのは、より高性能な基盤モデルを待つことだけではなく、開発プロセス全体で精度を作り込むことである。これまで整理した内容も合わせて、海外の研究動向を総合すると、その中核は大きく以下の四つに整理できる。

1. 入力の意味記述の低減

前述した北京大学とロンドンの UCL の研究は、問題記述の意味さを自動修復することでコード生成精度を改善しており、精度低下の原因がモデル能力だけでなく入力仕様の不明確さにもあることを示している [52]。また、中国の国防科技大学の研究では、コードスニペットへの適応では単発指示よりも対話的な追加質問やコンテキスト補完が有効であり、既存コードや制約を伴う実務では、タスク分解して入力を整えながら進めるプロセスが不可欠であることを示している [64]。

2. 内部知識への接続

前述した要件 QA、RAG を用いた要件生成、セキュアコードレビュー支援などの研究では、内部文書、標準、過去のレビューや履歴データを参照させることで、LLM の出力を組織固有のコンテキストに接地し、根拠の薄い生成を抑えている [30, 51, 55]。DORA も、AI が内部データへアクセスできることを重要な能力として位置付けており、高精度化には「大きいモデルを使うこと」だけでなく、「モデルが参照できる組織知を整備すること」でも重要となる [37]。

3. タスク分解と多段階生成・評価

前述した中国の BitsAI-CR によるコードレビューや Manchester 大学等の要件定義書生成の研究では、複雑なタスクを複数段階に分け、それぞれに異なる評価基準や確認工程を設けることで、一括生成よりも精度を高められることを示している [60, 65]。これは、AI を工程の一部に差し込むだけでなく、上流で要件を分解・構造化し、下流でレビューやテスト、リリース判定を前倒して組み込むような、工程全体の再設計が必要であることを意味する。

4. 誤りを前提とした検知・検証・保守的運用

King's College London、Calgary 大学、York 大学の共同研究では、メタモルフィック関係を用いてハルシネーションを自己完結的に検知する手法を示している [66]。また、前述した南京大学とシンガポールの大学の研究での対話的生成とミューテーションに基づく選別を通じた形式仕様生成手法や、Meta でのミューテーションを活用したテスト生成と検査器評価手法でも「出力の妥当性を事後にふるいにかける」方向性を示している [36, 53]。

さらに、California 大学と IBM 研究所の LLM の自信度を校正する研究や、Meta 等のリスクスコアリングを組み込んだリリース運用の研究もあり、これらは生成結果をそのまま採用するのではなく、信頼度やリスクに応じて判断する仕組みの重要性を示している [50, 67]。

一方で、Delft 工科大学の研究では既製 LLM を制御なしにプログラミングへ適用すると有害な出力が生じうることも報告されており、実務では人間の監督、継続的監査、最終判断が不可欠であると総括できる [27, 68]。

表 10 LLM のアウトプット高精度化の観点

項 番	観点	主要施策	実務上の要点
1	コンテキ スト	内部文書・コード・標準仕様・レビュー履歴を参照させ、出力を組織コンテキストに接地する。	コンテキスト不足による誤答を減らすため、更新された知識基盤を維持する。
2	分解・対 話	タスク分解、追加質問、段階的プロンプトで曖昧さを減らす。	一括生成より反復生成を基本とする。
3	検証	形式仕様、テスト、ハルシネーション検知、セキュリティ検査で出力をふるいにかける。	出力を証拠付きで採否判断する。
4	校正・ゲ ーティン グ	信頼度推定、リスクスコア、リリース制御で意思決定に結びつける。	モデルの自信と実正解のずれを継続監視する。
5	ガバナン ス	ログ、トレーサビリティ、責任分界、監査を設計する。	速度向上と説明責任を両立させる。

第7章 結論と今後の展望

本章では、第2章で示した3つの調査目標に対する本調査の結論を述べ、あわせて今後の展望を示す。

7.1 調査目標に対する結論

目標1（組織設計）については、海外では、レガシー対応・内製化・開発プロセスを個別の施策としてではなく、連続した一体の経営戦略として設計している構造が確認された。多くの事例ではレガシー依存の克服がその起点となるが、出発点は組織の状況によって異なり、いずれの場合も、開発の主導権と実行能力を組織内部に保持・再構築し（内製化）、それを継続的に機能させる開発プロセスへとつなげている点が共通している。組織構造の面でも、再利用可能な能力を共通基盤として整備し、OSS等の外部エコシステムとの接続を制度化したうえで、AIの利用を前提に共通基盤・統治機能・現場の実行組織の役割分担を再設計している。ここから、顧客価値創出を支える組織設計の中核は、個別技術の導入よりも、組織的な開発実行能力を一体として設計する点にあると整理できる。

目標2（スキル体系・人材育成）については、海外では、生成AIやAIエージェントの普及により、ソフトウェアエンジニアの価値の中心が「コードを書く量」から、要件を分解し設計意図・制約を明示してAIに渡し、出力を検証・統合する上位の判断へ移っていることが確認された。求められるスキルは、上流（問題設定・仕様化・制約の言語化）、下流（検証・再現性・セキュリティ・採否判断）、横断（批判的思考・倫理・説明責任）の三層として再定義されている。育成は、単発のツール研修では不十分であり、実務に埋め込んだ継続学習への転換と、評価・昇格基準との連動が求められる。さらに、個人のスキルだけでは成果は安定せず、組織戦略・組織基盤・開発プロセスを一体で整えることが、生産性と品質を最大化する条件となる。

目標3（開発プロセス）については、海外では、AI導入を単なるコーディング支援として扱う段階から、AIを前提に開発ライフサイクル全体を再設計する段階へ移行していることが確認された。その中核は、生成と人間の評価・修正を短いサイクルで往復させる反復プロセスであり、上流ではAIによって要件の曖昧さを減らし検証可能な形へ構造化し、下流ではレビュー・テスト・リリース判定で高リスク箇所を早期に絞り込むリスクベース運用へ広げている。その際、最終的な採否判断は人間が保持し、AIの出力は、曖昧記述の低減・内部知識への接続・多段階の生成と検証を通じて、工程全体で高精度化される。ここから、開発ライフサイクル全体の最適化（品質・コスト・速度）は、個別ツールの導入ではなく、AIを前提とした工程の再設計によって実現されると整理できる。

以上の結論の基礎となった、各章・各観点における海外動向の要点を、表 11 に整理する。

表 11 各調査目標における海外動向の要点

調査目標	観点（節）	海外動向の要点
目標 1 組織設計	経営戦略 (4.1)	レガシー対応・内製化・開発プロセスを個別施策としてではなく、連続した一体の経営戦略として設計する。レガシー依存の克服を起点の一つとしつつ、開発の主導権と実行能力を組織内部に保持・再構築し、それを継続的に機能させる仕組みまで一体で設計する点に特徴がある。
	組織構造 (4.2)	個別チームの努力に依存せず、共通化すべき能力・外部と接続すべき機能・横断的に担うべき役割を組織として明確化する。再利用可能な能力を共通基盤として整備し、OSS 等の外部接続を制度化したうえで、AI 利用を前提に共通基盤・統治機能・現場組織の役割分担を再設計する。
目標 2 スキル 体系・ 人材育 成	役割の変化 (5.1)	価値の中心が「コードを書く量」から、要件のタスク分解、設計意図・制約の明示、AI 出力の検証・統合といった上位の判断へ移る。複雑・曖昧なタスクほど、人間が最終判断者として問題設定・文脈設計・検証を担う。
	スキル体系 (5.2)	求められるスキルを、上流（問題設定・仕様化・制約の言語化）、下流（検証・再現性・セキュリティ・採否判断）、横断（批判的思考・倫理・説明責任）の三層として再定義する。テスト等の専門職に限らず、AI を利用する全員が最低限の検証責任を負う前提に立つ。
	育成モデル (5.3)	単発のツール研修では不十分であり、実務に埋め込んだ継続学習への転換が求められる。あわせて、評価・昇格基準と連動させながら、スキル体系を継続的に更新することが重要となる。
	個人外要素 (5.4)	個人のスキルだけでは成果が安定せず、組織戦略・組織基盤・開発プロセスを一体で整えることが、生産性と品質を最大化する条件となる。OSS への貢献や標準化を通じて、品質・セキュリティ・説明責任の共通言語を持つことも有効である。
目標 3	開発プロセ	単なるコーディング支援から、生成と人間の評価・修正

調査目標	観点（節）	海外動向の要点
開発プロセス	スの見直し (6.1)	を短いサイクルで往復させる反復プロセスへ移り、開発ライフサイクル全体を再設計する。局所的な速度向上は品質・事業成果に直結しないため、速度だけでなく品質・手戻り・リリース安定性を含めて評価することが重視される。小さな反復は、AI 起因の品質変動を管理する制御方式としても位置づけられる。
	上流工程の AI 活用 (6.2)	AI を上流工程の省略ではなく、上流成果物の形式化・構造化によって品質を高める道具として用いる。要件の曖昧さ低減、検証可能な仕様への変換、設計候補の生成・比較とトレーサビリティの前倒しなど、下流で検証可能な形に整える活用が進む。
	下流工程での AI 活用 (6.3)	実装コードの量産ではなく、レビュー・テスト・リリース判定で高リスク箇所を早期に絞り込むリスクベース運用へ広がる。AI は要約・初期指摘やリスク推定を担い、重大度の判断と最終的な採否は人間が保持する構造が主流である。
	LLM のアウトプットの 高精度化 (6.4)	高精度化を基盤モデル任せにせず、工程全体で作り込む。中核となるのは、入力 of 曖昧記述の低減、内部知識への接続、タスク分解と多段階の生成・評価、誤りを前提とした検知・検証・保守的運用の四点である。

7.2 今後の展望

本調査は海外動向の把握に主眼を置いたものであり、対象領域は変化が速い。したがって、今後は本調査で整理した論点を定期的に更新・追跡し、動向の変化を継続的にモニタリングしていくことが求められる。

また、ソフトウェアエンジニアリングは、システム全体を対象とするシステムズエンジニアリング（SE）に包含される一領域であり、本調査が組織設計・人材育成・開発プロセスの観点で確認した論点の多くは、それを包含する SE のレベルでも同様に議論されている。SE の直近動向については別報告〔システムズエンジニアリング動向調査報告書〕で整理しており、あわせて参照されたい。

参考文献

- [1] Derek DeBellis, Kevin M. Storer, Nathen Harvey, Matt Beane, Rob Edwards, Edward Fraser, Ben Good, Eirini Kalliamvakou, Gene Kim, Eric Maxwell, Sarah D' Angelo, Sarah Inman, Ambar Murillo, and Daniella Villalba, “State of AI-assisted Software Development 2025,” <https://research.google/pubs/dora-2025-state-of-ai-assisted-software-development-report/>, 2025.
- [2] Cloud Native Computing Foundation, “CNCF Annual Cloud Native Survey: The Infrastructure of AI's Future,” <https://www.cncf.io/reports/the-cncf-annual-cloud-native-survey/>, 2026.
- [3] Linux Foundation, “The State of Global Open Source 2025,” <https://www.linuxfoundation.org/research/world-of-open-source-global-2025>, 2025.
- [4] LeadDev, “The AI Impact Report 2025,” <https://leaddev.com/the-ai-impact-report-2025>, 2025.
- [5] 情報処理推進機構（IPA）ソフトウェアモダナイゼーション委員会, “ソフトウェアモダナイゼーション委員会 報告書,” <https://www.ipa.go.jp/disc/committee/eid2eo00000036bq-att/software-modernization-committee-20260324-report.pdf>, 2026.
- [6] 情報処理推進機構（IPA）, “ソフトウェア開発データ白書 2025,” <https://www.ipa.go.jp/digital/software-survey/software-engineering/software2025.html>, 2025.
- [7] 情報処理推進機構（IPA）, “DX 動向 2025,” <https://www.ipa.go.jp/digital/chousa/dx-trend/tbl5kb0000001mn2-att/dx-trend-2025.pdf>, 2025.
- [8] Aleksandre Asatiani, Tapani Rinta-Kahila, Jacob Torell, and Johan Magnusson, “Technical Debt Is Killing Digital Transformation, But There Is a Way Out,” <https://doi.org/10.1177/00081256251370795>, 2025.
- [9] Casper Lassenius, Parastoo Mohagheghi, and Jefferson Seide Molléri, “Bringing it home: successful backourcing of software development in the public sector,” <https://link.springer.com/article/10.1007/s10664-025-10722-1>, 2025.

- [10] DoD Chief Information Officer, “The State of DevSecOps within the Department of Defense,”
<https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsStateOf.pdf>, 2025.
- [11] Bart Daemen, Konstantinos Tsilionis, and Oktay Turetken, “Balancing organizational alignment, team autonomy, and control in large-scale agile organizations,” <https://doi.org/10.1016/j.infsof.2025.107998>, 2025.
- [12] DoD Chief Information Officer, “Software Modernization Implementation Plan FY25-26,” <https://dodcio.defense.gov/Portals/0/Documents/Library/SW-Mod-I-Plan25-26.pdf>, 2025.
- [13] DoD Chief Information Officer, “DoD Enterprise DevSecOps Activities & Tools Guidebook,”
<https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebook.pdf>, 2025.
- [14] DORA, “Platform engineering,” <https://dora.dev/capabilities/platform-engineering/>, 2026.
- [15] Colin Diggs, Michael Doyle, Amit Madan, Siggy Scott, Emily Escamilla, Jacob Zimmer, Naveed Nekoo, Paul Ursino, Michael Bartholf, Zachary Robin, Anand Patel, Chris Glasz, William Macke, Paul Kirk, Jasper Phillips, Arun Sridharan, Doug Wendt, Scott Rosen, “Leveraging LLMs for Legacy Code Modernization: Evaluation of LLM-Generated Documentation,”
<https://doi.org/10.1109/LLM4Code66737.2025.00027>, 2025.
- [16] LY Corporation, “How LY Corporation is leveraging Kubernetes as the foundation of its large-scale internal PaaS,” <https://www.cncf.io/case-studies/ly-corporation/>, 2025.
- [17] Platform Engineering Community / Humanitec, “State of Platform Engineering Report Volume 3,” <https://humanitec.com/whitepapers/state-of-platform-engineering-report-volume-3>, 2025.
- [18] Sodexo, “Scaling digital and AI product development with an internal developer platform,” <https://2025.platformcon.com/sessions/scaling-digital-and-ai-product-development-with-an-internal-developer-platform>, 2025.
- [19] Fraunhofer ISST / Data Spaces Support Centre, “DSSC - Data Spaces Support

Centre,” <https://dssc.eu/>, 2025.

- [20] Data Spaces Support Centre, “The Blueprint for Data Spaces v2.0,” <https://dssc.eu/space/BVE2/1071251457/>, 2025.
- [21] Linux Foundation Research, “The 2025 State of OSPOs and Open Source Management,” <https://www.linuxfoundation.org/research/ospo-2025>, 2025.
- [22] Linux Foundation, “ROI for Open Source Software Contribution,” <https://www.linuxfoundation.org/research/contribution-roi>, 2026.
- [23] Office of Systems Engineering and Architecture, Office of the Under Secretary of Defense for Research and Engineering, “Software Engineering for Continuous Delivery of Warfighting Capability,” <https://www.cto.mil/wp-content/uploads/2025/08/SWE-Guide-July2025-secured-1.pdf>, 2025.
- [24] Software Engineering Institute, “AI-Augmented Software Engineering,” <https://www.sei.cmu.edu/ai-augmented-software-engineering/>, 2025.
- [25] Domenico Amalfitano, Andreas Metzger, Marco Autili, Tommaso Fulcini, Tobias Hey, Jan Keim, Patrizio Pelliccione, Vincenzo Scotti, Anne Kozirolek, Raffaella Mirandola, and Andreas Vogelsang, “A Research Roadmap for Augmenting Software Engineering Processes and Software Products with Generative AI,” <https://arxiv.org/abs/2510.26275>, 2025.
- [26] Jenna Butler, Jina Suh, Sankeerti Haniyur, and Constance Hadley, “Dear Diary: A Randomized Controlled Trial of Generative AI Coding Tools in the Workplace,” <https://doi.org/10.1109/ICSE-SEIP66354.2025.00034>, 2025.
- [27] Kevin Zheyuan Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz, “The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers,” <https://doi.org/10.2139/ssrn.4945566>, 2025.
- [28] Prithvi Raj, “From YAML to Intelligence: The Evolution of Platform Engineering,” <https://www.cncf.io/blog/2025/07/22/from-yaml-to-intelligence-the-evolution-of-platform-engineering/>, 2025.
- [29] Marco Gerosa and Adrienn Lawson, “2025 State of Tech Talent Report,” <https://www.linuxfoundation.org/research/open-source-jobs-report-2025>, 2025.

- [30] Jonathan Ullrich, Matthias Koch, and Andreas Vogelsang, “From Requirements to Code: Understanding Developer Practices in LLM-Assisted Software Engineering,” <https://arxiv.org/abs/2507.07548>, 2025.
- [31] Aayush Kumar, Yasharth Bajpai, Sumit Gulwani, Gustavo Soares, and Emerson R. Murphy-Hill, “Why AI Agents Still Need You: Findings from Developer-Agent Collaborations in the Wild,” <https://doi.org/10.1109/ASE63991.2025.00043>, 2025.
- [32] Justin D. Weisz, Shraddha Vijay Kumar, Michael Muller, Karen-Ellen Browne, Arielle Goldberg, Katrin Ellice Heintze, and Shagun Bajpai, “Examining the Use and Impact of an AI Code Assistant on Developer Productivity and Experience in the Enterprise,” <https://doi.org/10.1145/3706599.3706670>, 2025.
- [33] Software Engineering Institute, “Efficacy of Human Teaming with Generative AI for Software Maintenance,” <https://sei.cmu.edu/annual-reviews/2025-year-in-review/efficacy-of-human-teaming-with-generative-ai-for-software-maintenance/>, 2026.
- [34] Fannar Steinn Aðalsteinsson, Björn Borgar Magnússon, Mislav Milicevic, Adam Nirving Davidsson, and Chih-Hong Cheng, “Rethinking Code Review Workflows with LLM Assistance: An Empirical Study,” <https://doi.org/10.1109/ESEM64174.2025.00013>, 2025.
- [35] Fang Liu, Simiao Liu, Yinghao Zhu, Xiaoli Lian, and Li Zhang, “SecureReviewer: Enhancing Large Language Models for Secure Code Review through Secure-Aware Fine-Tuning,” <https://doi.org/10.48550/arXiv.2510.26457>, 2025.
- [36] Christopher Foster, Abhishek Gulati, Mark Harman, Inna Harper, Ke Mao, Jillian Ritchey, Hervé Robert, and Shubho Sengupta, “Mutation-Guided LLM-based Test Generation at Meta,” <https://doi.org/10.1145/3696630.3728544>, 2025.
- [37] DORA, “DORA AI Capabilities Model,” <https://dora.dev/research/2025/ai-capabilities-model/>, 2025.
- [38] Matheus de Moraes Leça and Ronnie de Souza Santos, “Curious, Critical Thinker, Empathetic, and Ethically Responsible: Essential Soft Skills for Data Scientists in Software Engineering,” <https://doi.org/10.1109/ICSE-SEIS66351.2025.00021>, 2025.
- [39] Lekshmi Murali Rani, Faezeh Mohammadi, Robert Feldt, and Richard Berntsson-

Svensson, “An Empirical Study on Decision-Making Aspects in Responsible Software Engineering for AI,” <https://doi.org/10.1109/ICSE-SEIP66354.2025.00056>, 2025.

- [40] Nishat Raihan, Mohammed Latif Siddiq, Joanna C. S. Santos, and Marcos Zampieri, “Large Language Models in Computer Science Education: A Systematic Literature Review,” <https://doi.org/10.1145/3641554.3701863>, 2025.
- [41] Titus Winters, “CS and SE Education, Post-AI,” <https://doi.org/10.1145/3770762.3773025>, 2025.
- [42] Armando Fox, Pablo Fernandez, Juho Leinonen, and José Antonio Parejo Maestre, “Using Generative AI to Scaffold the Teaching of Software Engineering Team Skills,” <https://doi.org/10.1145/3641555.3705107>, 2025.
- [43] European Commission, “AI Adoption in the EU’ s Public Sector: An Opportunity to Better Serve Citizens and Support Startups,” <https://digital-strategy.ec.europa.eu/en/library/ai-adoption-eus-public-sector-opportunity-better-serve-citizens-and-support-startups>, 2025.
- [44] European Commission, “Apply AI Strategy,” <https://digital-strategy.ec.europa.eu/en/policies/apply-ai>, 2025.
- [45] Open Source Security Foundation (OpenSSF), “Secure Open Source Software Vision Brief 2025,” <https://openssf.org/resources/publications/secure-open-source-software-vision-brief-2025/>, 2025.
- [46] Cloud Native Computing Foundation, “The Platform Under the Model: How Cloud Native Powers AI Engineering in Production,” <https://www.cncf.io/blog/2026/03/26/the-platform-under-the-model-how-cloud-native-powers-ai-engineering-in-production/>, 2026.
- [47] Carnegie Mellon University Software Engineering Institute, “Perspectives on Generative AI in Software Engineering and Acquisition,” <https://sei.cmu.edu/blog/perspectives-on-generative-ai-in-software-engineering-and-acquisition/>, 2025.
- [48] Fraunhofer IESE, “Generative AI in Software Engineering: A Year in Retrospective,” <https://www.iese.fraunhofer.de/blog/generative-ai-in-se-and-ai-orchestrated-sdlc-retrospective/>, 2025.

- [49] Guilherme Vaz Pereira, Victoria Jackson, Rafael Prikladnicki, André van der Hoek, Luciane Fortes, Carolina Araújo, André Coelho, Ligia Chelli, and Diego Ramos, “Exploring GenAI in Software Development: Insights from a Case Study in a Large Brazilian Company,” <https://doi.org/10.1109/ICSE-SEIP66354.2025.00035>, 2025.
- [50] Rui Abreu, Vijayaraghavan Murali, Peter C Rigby, Chandra Maddila, Weiyan Sun, Jun Ge, Kaavya Chinniah, Audris Mockus, Megh Mehta, and Nachiappan Nagappan, “Moving Faster and Reducing Risk: Using LLMs in Release Deployment,” <https://conf.researchr.org/details/icse-2025/icse-2025-software-engineering-in-practice/18/Moving-Faster-and-Reducing-Risk-Using-LLMs-in-Release-Deployment>, 2025.
- [51] Longxing Yang, Yixing Luo, Hao Gao, Yingshuang Fan, Jingru Zhang, Xiaofeng Li, Xiaogang Dong, Bin Gu, Zhi Jin, and Mengfei Yang, “Evaluating Large Language Models for Requirements Question Answering in Industrial Aerospace Software,” <https://doi.org/10.1145/3696630.3728560>, 2025.
- [52] Haoxiang Jia, Robbie Morris, He Ye, Federica Sarro, and Sergey Mechtaev, “Automated Repair of Ambiguous Problem Descriptions for LLM-Based Code Generation,” <https://conf.researchr.org/details/ase-2025/ase-2025-papers/129/Automated-Repair-of-Ambiguous-Problem-Descriptions-for-LLM-Based-Code-Generation>, 2025.
- [53] Lezhi Ma, Shangqing Liu, Yi Li, Xiaofei Xie, and Lei Bu, “SpecGen: Automated Generation of Formal Program Specifications via Large Language Models,” <https://conf.researchr.org/details/icse-2025/icse-2025-research-track/173/SpecGen-Automated-Generation-of-Formal-Program-Specifications-via-Large-Language-Mod>, 2025.
- [54] Yixing Luo, Yiping Wang, Xiaofeng Li, Bin Gu, and Zhi Jin, “Leveraging Large Language Models for Reusable Requirements Management in Aerospace Software,” <https://conf.researchr.org/details/RE-2025/RE-2025-industrial-innovation-papers/3/Leveraging-Large-Language-Models-for-Reusable-Requirements-Management-in-Aerospace-So>, 2025.
- [55] Chetan Arora, Fanyu Wang, Kla Tantithamthavorn, Aldeida Aleti, and Shaun Kenyon, “From Domain Documents to Requirements: AI-Powered Retrieval-Augmented Generation in the Space Industry,” <https://arxiv.org/abs/2507.07689>, 2025.

- [56] James Ivers and Ipek Ozkaya, “Will Generative AI Fill the Automation Gap in Software Architecting?,” <https://doi.org/10.1109/ICSA-C65153.2025.00014>, 2025.
- [57] Matteo Esposito, Xiaozhou Li, Sergio Moreschini, Noman Ahmad, Tomas Cerny, Karthik Vaidhyanathan, Valentina Lenarduzzi, and Davide Taibi, “Generative AI for software architecture: Applications, challenges, and future directions,” <https://doi.org/10.1016/j.jss.2025.112607>, 2025.
- [58] Christof Tinnes, Alisa Carla Welter, and Sven Apel, “Software Model Evolution with Large Language Models: Experiments on Simulated, Public, and Industrial Datasets,” <https://www.se.cs.uni-saarland.de/publications/docs/TWA+25.pdf>, 2025.
- [59] Tobias Eisenreich, Sandro Speth, and Stefan Wagner, “From Requirements to Architecture: An AI-Based Journey to Semi-Automatically Generate Software Architectures,” <https://doi.org/10.1145/3643660.3643942>, 2024.
- [60] Tao Sun, Jian Xu, Yuanpeng Li, Zhao Yan, Ge Zhang, Lintao Xie, Lu Geng, Zheng Wang, Yueyan Chen, Qin Lin, Wenbo Duan, Kaixin Sui, and Yuanshuo Zhu, “BitsAI-CR: Automated Code Review via LLM in Practice,” <https://conf.researchr.org/details/fse-2025/fse-2025-industry-papers/24/BitsAI-CR-Automated-Code-Review-via-LLM-in-Practice>, 2025.
- [61] Ye Shang, Quanjun Zhang, Chunrong Fang, Siqi Gu, Jianyi Zhou, and Zhenyu Chen, “A Large-scale Empirical Study on Fine-tuning Large Language Models for Unit Testing,” <https://doi.org/10.1145/3728951>, 2025.
- [62] Benyamin Beyzaei, Saghar Talebipour, Ghazal Rafiei, Nenad Medvidovic, and Sam Malek, “Automated Test Transfer Across Android Apps Using Large Language Models,” <https://doi.org/10.1145/3728975>, 2025.
- [63] Islem Bouzenia and Michael Pradel, “You Name It, I Run It: An LLM Agent to Execute Tests of Arbitrary Projects,” <https://arxiv.org/abs/2412.10133>, 2025.
- [64] Tanghaoran Zhang, Yue Yu, Xinjun Mao, Shangwen Wang, Kang Yang, Yao Lu, Zhang Zhang, and Yuxin Zhao, “Instruct or Interact? Exploring and Eliciting LLMs’ Capability in Code Snippet Adaptation Through Prompt Engineering,” <https://arxiv.org/abs/2411.15501>, 2025.
- [65] Taohong Zhu, Lucas Cordeiro, and Youcheng Sun, “ReqInOne: A Large

Language Model-Based Agent for Software Requirements Specification Generation,” <https://conf.researchr.org/details/RE-2025/RE-2025-re-next-papers/6/ReqInOne-A-Large-Language-Model-Based-Agent-for-Software-Requirements-Specification->, 2025.

- [66] Borui Yang, Md Afif Al Mamun, Jie M. Zhang, and Gias Uddin, “Hallucination Detection in Large Language Models with Metamorphic Relations,” <https://doi.org/10.1145/3715735>, 2025.
- [67] Yuvraj Singh Virk, Premkumar Devanbu, and Toufique Ahmed, “Calibration of Large Language Models on Code Summarization,” <https://doi.org/10.1145/3729400>, 2025.
- [68] Ali Al-Kaswan, Sebastian Deatc, Begüm Koç, Arie van Deursen, and Maliheh Izadi, “Code Red! On the Harmfulness of Applying Off-the-shelf Large Language Models to Programming Tasks,” <https://doi.org/10.1145/3729380>, 2025.