アーキテクチャ決定レコードを活用して、ソフトウェア開発プロジェクトの技術的意思決定を効率化する

AWS 規範ガイダンス

AWS 規範ガイダンス: アーキテクチャ決定レコードを活用して、ソフトウェア開発プロジェクトの技術的意思決定を効率化する

Table of Contents

アーキテクチャ決定レコードを活用して、ソフトウェア開発プロジェクトの技術的意思決定を効率化する

Darius Kunce、Dominik Goby、Amazon Web Services (AWS)

2022 年 3 月 (ドキュメント履歴)

このガイドでは、ソフトウェアエンジニアリングプロジェクトのアーキテクチャ決定レコード (ADR) プロセスを紹介します。ADR はチームの連携を支援し、プロジェクトや製品の戦略的方向性を文書化し、繰り返し発生する時間のかかる意思決定作業を減らします。

ソフトウェアエンジニアリングチームは、プロジェクトや製品の開発中に、目標を達成するためにアーキテクチャ上の決定を下す必要があります。これらの決定は、コマンドクエリ責任分離 (CQRS) パターンを使用するなどの技術的なものである場合もあれば、GitFlow ワークフローを使用してソースコードを管理するなどのプロセス関連のものである場合もあります。このような決定を行うのは時間がかかり、難しいプロセスです。チームはこれらの決定について正当性を示し、文書化し、関連する利害関係者に伝える必要があります。

アーキテクチャ上の決定を行う際には、次の 3 つの大きなアンチパターンがよく起こります。

- 間違った選択をすることを恐れて、何の決定も行なわれない。
- 何の正当性もなく決定が行なわれ、なぜその決定が行なわれたのか理解されない。その結果、同じトピックが複数回議論されることになる。
- 決定がアーキテクチャ上決定リポジトリに記録されないため、チームメンバーは決定が下されたことを忘れたり、知らなかったりする。

こうしたアンチパターンには、製品やプロジェクトの開発プロセスで取り組むことが特に重要です。

意思決定、背景、決定に至った考慮事項を ADR という形で把握することで、現在および将来の利害関係者は、行なわれた決定や各決定の背後にある思考プロセスに関する情報を収集できます。これにより、ソフトウェア開発時間が短縮され、将来のチームにより適切なドキュメントが提供されます。

ターゲットを絞ったビジネス成果

ADR は次の 3 つのビジネス成果を目標としています。

- 現在のチームメンバーと将来のチームメンバーを連携させる。

- プロジェクトや製品の戦略的方向性を設定する。
- アーキテクチャの決定を適切に文書化して伝達するプロセスを定義することで、決定のアンチパターンを回避する。

ADR は決定の背景を把握して、将来の利害関係者に情報を提供します。ADR のコレクションは、経験の引き継ぎと参考資料になります。チームやプロジェクトメンバーは、フォローアッププロジェクトや製品機能の計画に ADR コレクションを使用します。ADR を参照できるようになれば、開発、レビュー、アーキテクチャの決定に必要な時間を短縮できます。ADR により、他のチームが他のプロジェクトチームや製品開発チームが検討した検討事項から学び、洞察を得られるようになります。

ADR プロセス

アーキテクチャ決定レコード (ADR) は、構築計画中のソフトウェアアーキテクチャの重要な側面に関してチームが取った選択について説明する文書のことです。各 ADR には、アーキテクチャの決定、その背景、およびその結果が記載されています。ADR には状態があるため、ライフサイクルに従います。ADR の例については、「付録」を参照してください。

ADR プロセスはアーキテクチャ決定レコードのコレクションを出力します。このコレクションが決定ログを作成します。決定ログには、プロジェクトのコンテキストだけでなく、実装や設計に関する詳細な情報も記録されます。プロジェクトメンバーは各 ADR の見出しにざっと目を通して、プロジェクトコンテキストの概要を把握します。また ADR を読み、プロジェクトの実装と設計上の選択について深く掘り下げます。

チームが ADR を受け入れると、その ADR はイミュータブルになります。新しいインサイトに別の決定が必要な場合、チームは新しい ADR を提案します。チームがその新しい ADR を受け入れると、新しい ADR が前の ADR に取って代わります。

ADR プロセスの範囲

プロジェクトメンバーは、ソフトウェアプロジェクトまたは製品に影響を及ぼすアーキテクチャ上の重要な決定があるたびに ADR を作成する必要があります (Richards and Ford 2020)。この重要な決定には以下を含みます。

- 構造 (マイクロサービスなどのパターンなど)
- 非機能的要件 (セキュリティ、高可用性、耐障害性)
- 依存関係 (コンポーネントの結合)
- インターフェイス (API と公開済みの契約)
- 構築技術 (ライブラリ、フレームワーク、ツール、プロセス)

ADR プロセスでは、機能的要件と非機能的要件が最も一般的な入力項目です。

ADR の内容

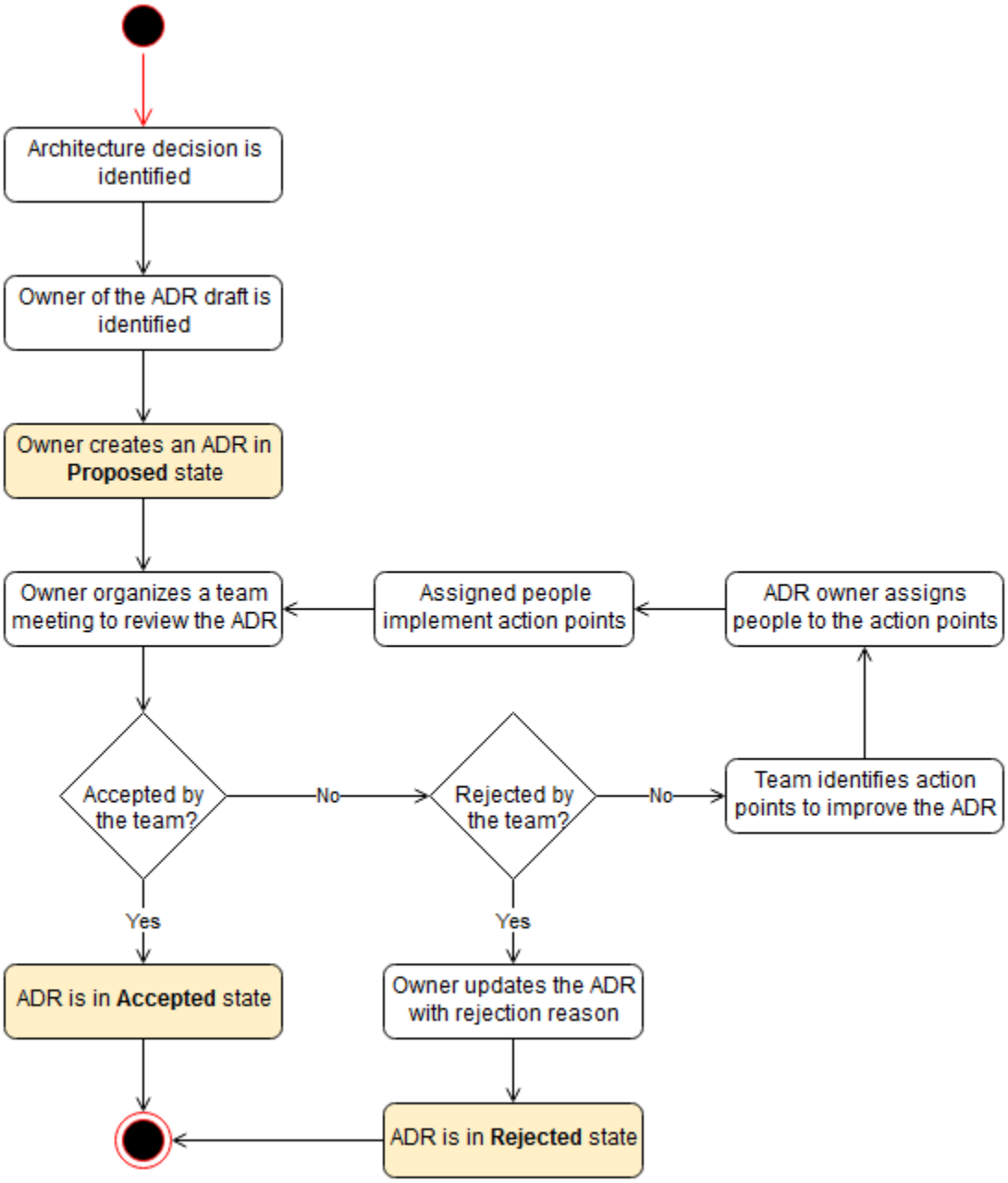
チームが ADR の必要性を判断すると、チームメンバーはプロジェクト全体のテンプレートに基づいて ADR の作成を開始します。(テンプレートの例については、「ADR GitHub organization」を参照

してください。) テンプレートを使うと ADR の作成が簡単になり、ADR がすべての関連情報を確実にキャプチャできるようになります。各 ADR には、少なくとも、決定の背景、決定自体、および決定がプロジェクトとその成果物にもたらす結果を定義する必要があります。(これらのセクションの例については、「付録」を参照してください。) ADR 構造の最も強力な点の 1 つは、チームがどのように決定を実行したかではなく、決定の理由に焦点を当てていることです。チームが決定を下した理由を理解することで、他のチームメンバーがその決定を採用しやすくなり、意思決定プロセスに関与しなかった他のアーキテクトが将来その決定を覆すのを防ぐことができます。

ADR の導入プロセス

すべてのチームメンバーが ADR を作成できますが、チームで ADR の所有権の定義を確立する必要があります。ADR の所有者である各作成者は、ADR の内容を積極的に管理し、伝える必要があります。この所有権を明確にするために、このガイドの以下のセクションでは ADR 作成者を ADR 所有者と呼びます。他のチームメンバーはいつでも ADR に寄稿できます。チームが ADR を受け入れる前に ADR の内容が変更された場合、所有者はその変更を承認する必要があります。

次の図は、ADR の作成、所有、採用のプロセスを示したものです。



チームでアーキテクチャの決定事項とその所有者を決定したら、ADR 所有者はプロセス開始時に ADR を提案済みの状態にします。提案済み状態の ADR はレビューの準備ができています。

その後、ADR 所有者は ADR のレビュープロセスを開始します。ADR レビュープロセスの目標は、チームが ADR を受け入れるか、やり直す必要があると判断するか、ADR を拒否するかを決定することです。所有者を含むプロジェクトチームで ADR をレビューします。レビューミーティングは、ADR を読むためだけの時間を設けることから始めます。平均して 10～15 分で十分です。この

時間で、各チームメンバーは文書を読み、不明瞭なトピックを報告するコメントや質問を付け足します。レビューフェーズの後、ADR 所有者は各コメントを読み上げてチームで話し合います。

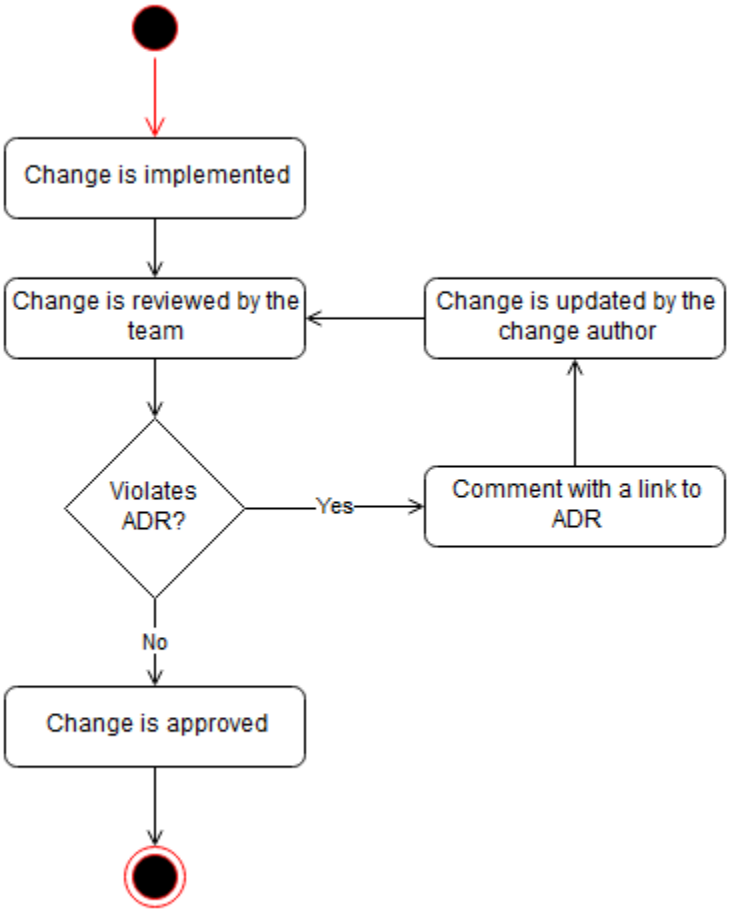
チームが ADR を改善するためのアクションポイントを見つけても、ADR の状態は提案済みのまま変わりません。ADR 所有者はアクションを策定し、チームと協力して各アクションに担当者を付けます。これで各チームメンバーは寄稿することができ、アクションポイントを解決することができます。レビュープロセスのスケジュールを変更するのは ADR 所有者の責任です。

また、チームは ADR を拒否する決定を行うこともできます。この場合、ADR 所有者は、将来同じトピックに関する議論が起こるのを防ぐため、拒否の理由を追加します。所有者は ADR の状態を拒否済みに変更します。

チームが ADR を承認すると、所有者はタイムスタンプ、バージョン、利害関係者のリストを追加します。その後、所有者は状態を承諾済みに更新します。

ADR と ADR が作成する決定ログは、チームが下した決定を表し、すべての決定の履歴になります。チームは可能な限り、コードやアーキテクチャのレビューの際に ADR を参考として使用します。チームメンバーは、コードレビュー、設計作業、実装作業を行うだけでなく、製品の戦略的決定について ADR を参照する必要があります。

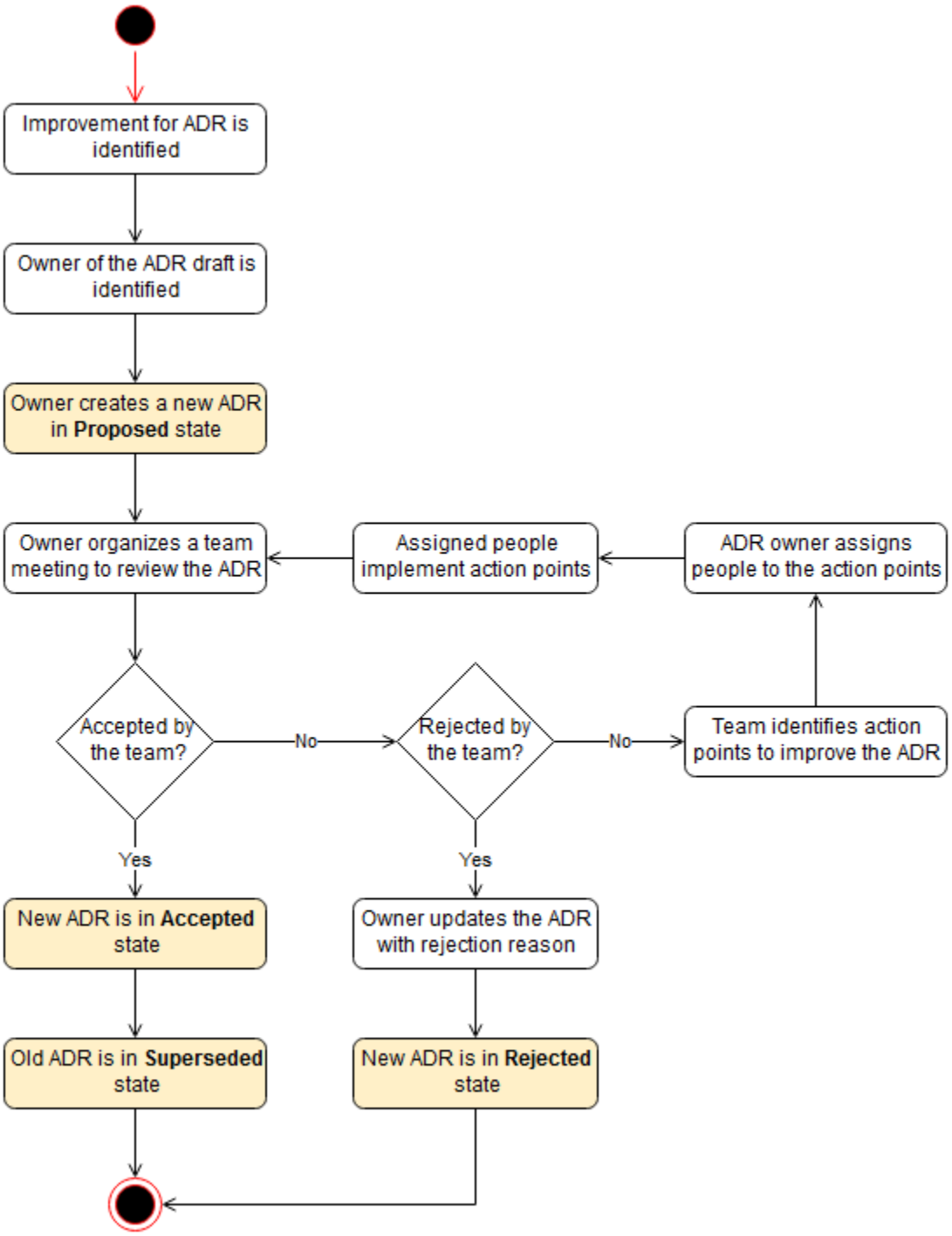
次の図は、ADR を適用して、ソフトウェアコンポーネントの変更が、合意された決定に準拠しているかどうかを検証するプロセスを示しています。



良い習慣として、各ソフトウェアの変更はピアレビューを受け、少なくとも 1 回の承認を得る必要があります。コードレビュー中に、コードレビュー担当者は ADR に違反する変更を 1 つ以上発見することがあります。この場合、レビュー担当者はコード変更の作成者にコードの更新を依頼し、ADR へのリンクを共有します。作成者がコードを更新すると、そのコードはピアレビュー担当者によって承認され、メインのコードベースにマージされます。

ADR のレビュープロセス

ADR をチームが承認または拒否した後は、ADR を変更不可能な文書として扱う必要があります。既存の ADR を変更する場合は、新しい ADR を作成し、新しい ADR のレビュープロセスを確立し、ADR を承認する必要があります。チームが新しい ADR を承認した場合、所有者は古い ADR の状態を置き換え済みに変更する必要があります。以下の図は、その更新プロセスを示したものです。


Improvement for ADR is identified
Owner of the ADR draft is identified
Owner creates a new ADR in **Proposed** state
Owner organizes a team meeting to review the ADR
Assigned people implement action points
ADR owner assigns people to the action points
Accepted by the team?
No
Rejected by the team?
No
Team identifies action points to improve the ADR
Yes
Yes
New ADR is in **Accepted** state
Owner updates the ADR with rejection reason
Old ADR is in **Superseded** state
New ADR is in **Rejected** state

ベストプラクティス

所有権を昇格させる。プロジェクトチームの各メンバーには、ADR を作成して所有する権限を与えられる必要があります。このプラクティスにより、アーキテクチャの研究作業がチームメンバー間で分散され、ソリューションアーキテクトやチームリーダーからの作業がオフロードされます。また、意思決定プロセスにおける当事者意識も育まれます。これにより、チームはそれらの決定を組織の上位レベルから押し付けられた決定として扱うのではなく、より迅速に受け入れられるようになります。

ADR 履歴を保存する。ADR には変更履歴が必要であり、各変更には所有者が設定されている必要があります。ADR 所有者が ADR を更新したら、古い ADR のステータスを置き換え済みに変更する必要があります。変更内容を新しい ADR の変更履歴に記録し、古い ADR は決定ログに残しておきます。

定期的なレビュー会議を予定する。新しい (未開発の) プロジェクトに取り組んでいる場合、ADR プロセスは最初はかなり厳しいものになる可能性があります。毎日の簡単なミーティングの前後に、ADR に関するディスカッションとレビューミーティングを定期的に行うことをお勧めします。このアプローチでは、定義した ADR は 2～3 回のスプリントで安定し、少ないミーティングで強固な基盤を構築できます。

ADR を一元的な場所に集約する。各プロジェクトメンバーは ADR のコレクションにアクセスできる必要があります。ADR は 1 か所に保存し、プロジェクトドキュメントのメインページで参照することをお勧めします。ADR の保存には次の 2 つの方法がよく使用されます。

- ADR のバージョン管理が容易になる Git リポジトリ
- チームメンバー全員が ADR にアクセスできるようになる Wiki ページ

非準拠コードに対処する。ADR プロセスでは、準拠していないレガシーコードの問題は解決されません。確立された ADR をサポートしていないレガシーコードがある場合は、新しい変更を加えながら古いコードベースやアーティファクトを徐々に更新することも、チームが技術的負債タスクを作成してコードを明示的にリファクタリングすることもできます。

よくある質問

ADR プロセスを作成するメリットは何ですか?

プロジェクトチームは ADR プロセスを作成して、アーキテクチャに関する意思決定を効率化し、同じアーキテクチャのトピックについての繰り返しの議論を回避し、アーキテクチャに関する決定を効果的に伝達すべきです。

プロジェクトチームはいつ ADR を作成する必要がありますか?

プロジェクトチームは、構造 (マイクロサービスなどのパターン)、機能以外の要件 (セキュリティ、高可用性、耐障害性)、依存関係 (コンポーネントの結合)、インターフェイス (API と公開済みの契約)、建設技術 (ライブラリ、フレームワーク、ツール、プロセス) に影響するソフトウェアのあらゆる側面について ADR を作成する必要があります。

プロジェクトチームはどのくらいの頻度で ADR をレビューすべきですか?

プロジェクトチームは ADR を受け入れる前に少なくとも 1 回はレビューする必要があります。

ADR を作成するのは誰ですか?

すべてのチームメンバーが ADR を作成できます。ADR の所有権の概念を広めることをお勧めします。ADR を所有する作成者は、ADR の内容を積極的に管理し、伝える必要があります。他のチームメンバーはいつでも ADR に寄稿できます。ADR の所有者は ADR への変更を承認する必要があります。

ADR にはどのような情報を含める必要がありますか?

少なくとも、各 ADR は、決定の背景、決定自体、決定がプロジェクトとその成果物に及ぼす影響について定義する必要があります。コンテキストには、チームが検討した可能な解決策を記載する必要があります。また、プロジェクト、顧客、または技術スタックに関連する情報も含める必要があります。決定には、チームが採用することを決定したソリューションを強制的な型で明確に記載する必要があります。「すべき」などの言葉は使わず、各決定を「私たちは...を使用します」または「チー

ムは...を使用する必要があります」と表現します。結果のセクションでは、決定を下す際の既知のトレードオフをすべて記載する必要があります。各 ADR には、ステータスと、変更日と変更の責任者を記載した変更ログが必要です。

ADR テンプレートはどこにありますか?

ADR テンプレートには複数のバージョンとバリエーションがあります。一般的に使用される ADR テンプレートの公開コレクションについては、ADR GitHub リポジトリを参照してください。

次のステップとリソース

小規模から始めて、ADR がチームにもたらすメリットを確認することをお勧めします。進行中のプロジェクトに取り組んでいる場合は、次のアーキテクチャ変更を特定し、提案された ADR プロセスを適用して最初の ADR を作成します。

もう 1 つの開始点は、ADR を使用してソフトウェア開発プロセス全体を文書化することです。開発プロセスは、チームがどの文書にも書かれていない暗黙の知識に基づいていることがよくあります。このプロセスを文書化することで、チームの新メンバーがよりスムーズに作業できるようになります。

未開発のプロジェクトに取り組んでいる場合は、ADR プロセスを適用して、すべての決定事項を最初から数文にまとめます。その後、それらの ADR を繰り返し処理して、新しい情報を補足することができます。ADR を確立したら、コードレビュープロセスの参考として ADR の使用を開始できます。

リソース

- アーキテクチャ決定レコード。https://adr.github.io/。
- Mark Richards、Neal Ford。2021 年。Fundamentals of Software Architecture。セバストポリ: O'Reilly Media

付録: ADR の例

タイトル

この決定は、ABC アプリケーション開発におけるソフトウェア開発ライフサイクルのアプローチを定義しています。

ステータス

承諾

日付

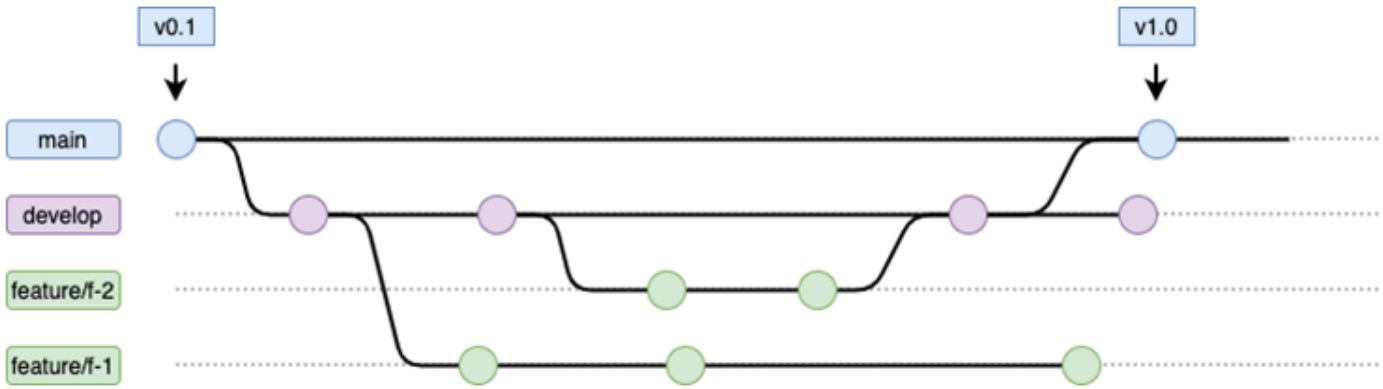
2022-03-11

Context

ABC アプリケーションはパッケージ化されたソリューションで、デプロイパッケージを使用してお客様の環境にデプロイされます。制御可能な機能、修正プログラム、リリースパイプラインを実現できる開発プロセスが必要です。

決定

GitFlow ワークフローを改造したバージョンを使用して、ABC アプリケーションを開発します。



わかりやすくするため、`hotfix/*` ブランチと `release/*` ブランチは使用しません。ABC アプリケーションは特定の環境にデプロイされる代わりにパッケージ化されるからです。このため、複雑にして、本番リリースのバグ修正に迅速に対応できなくなったり、別の環境でのリリースのテストができなくなったりする必要はありません。

合意されたブランチ戦略は次のとおりです。

- 各リポジトリには、リリースのタグ付けに使用される保護された `main` ブランチが必要です。
- 各リポジトリには、進行中の開発作業のすべてに対して、保護された `develop` ブランチが必要です。

結果

ポジティブ:

- 適合した GitFlow プロセスにより、ABC アプリケーションのリリースバージョンを管理できるようになります。

ネガティブ:

- GitFlow はトランクベースの開発や GitHub フローよりも複雑で、オーバーヘッドも大きくなります。

コンプライアンス

- 各リポジトリの `main` ブランチと `develop` ブランチには `Protected` のマークを付ける必要があります。
- `main` ブランチと `develop` ブランチへの変更はマージリクエストを使用して伝達する必要があります。
- マージリクエストごとに少なくとも 1 つの承認が必要です。

Notes (メモ)

- 著者: Jane Doe
- バージョン: 0.1
- 変更ログ:
  - 0.1: 初回提案バージョン

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
初版発行	—	2022 年 3 月 16 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。