R-SDDプラットフォーム 業務利用マニュアル

> 対象読者: プロジェクト責任者、PM/PMO、業務設計者、開発者、テスト担当者、レビュー担当者、運用担当者
> 本書は特定プロジェクトに依存しない共通マニュアルである。個別プロジェクトのパス、URL、モジュール一覧、認証情報は導入時設定およびプロジェクト別運用補足に記載する。 更新日: 2026年10月10日

1. 製品の目的

R-SDDプラットフォーム は、企画・設計・実装・テスト・受入・リリースに分散する成果物を一つの追跡モデルで可視化し、工程の完了を客観的に判定する開発ガバナンス基盤である。

本製品が提供する価値は次の4点である。

1. **可視化**: 現在の工程、成果物、未達項目、テスト結果を同じ画面で確認する。
2. **追跡可能性**: 業務要求から設計、コード、API、テスト、リリースまでの関係を追跡する。
3. **品質ゲート**: ファイルの存在だけでなく、レビュー確認と実行結果を組み合わせて出門を判定する。
4. **監査性**: 誰が、いつ、何を確認またはスキップしたかをプロジェクト単位で記録する。

R-SDDプラットフォームは、利用者の明示的な操作により仕様書の草案生成、リポジトリ解析、AI深層分析を行う。ただし、正式成果物としての採用、コミット、Push、リリースおよび業務判断を自動で代行するものではない。各ツールが生成した一次情報を読み取り、事実に基づいて状態を表示し、最終判断は権限を持つ利用者が行う。

1.1 本製品の中心は「表示」ではなく「門控」である

本製品における工程ステータスは、単なる進捗自己申告ではない。各工程の成果物、客観証跡、SEまたは指定責任者の確認を揃えて初めて、次工程へ進む資格が成立する。

```
成果物あり + 客観証跡PASS + SE門控確認
                    ↓
              次工程へ進行可能
```

いずれか一つでも不足する場合、その工程は未完了である。後続作業を技術的に着手できる場合でも、プロジェクト管理上は次工程へ正式移行してはならない。この強制力により、「作成したつもり」「テストしたつもり」「レビューしたつもり」を排除し、伝統的なウォーターフォール開発における工程責任を可視化する。

2. 利用者と責任分担

役割	主な責任	プラットフォーム上の操作
プロジェクト責任者	工程、対象範囲、例外、リリース可否の決定	全体進捗確認、スキップ承認、出門判定
業務責任者	業務要求、業務ルール、受入条件の確定	業務設計・UAT成果物の確認
アーキテクト	システム構成、共通方式、命名体系の統制	開発規約・基本設計の確認
開発担当	詳細設計、実装、単体・結合テスト	成果物作成、テスト結果登録、是正
テスト担当	システムテスト計画・実行・証跡管理	シナリオ結果確認、欠陥追跡
レビュー担当	成果物と出門条件の独立確認	個別確認、一括確認、差戻し
運用担当	配備、監視、障害・変更管理	本番チェックリストと運用証跡確認

同一人物が複数役割を兼務できる。ただし、重要な品質ゲートでは作成者と確認者を分離する運用を推奨する。

3. 標準ライフサイクル

```
開発規約 → 基本設計 → 詳細設計 → 単体テスト → 結合テスト
        → システムテスト → UAT → 本番リリース
```

各工程は次の3要素で管理する。

- **成果物**: 規定の場所と名前で存在する文書・コード・結果ファイル
- **客観証跡**: テスト結果、レビュー記録、承認記録、リリース記録
- **出門判断**: 責任者が条件充足を確認した事実

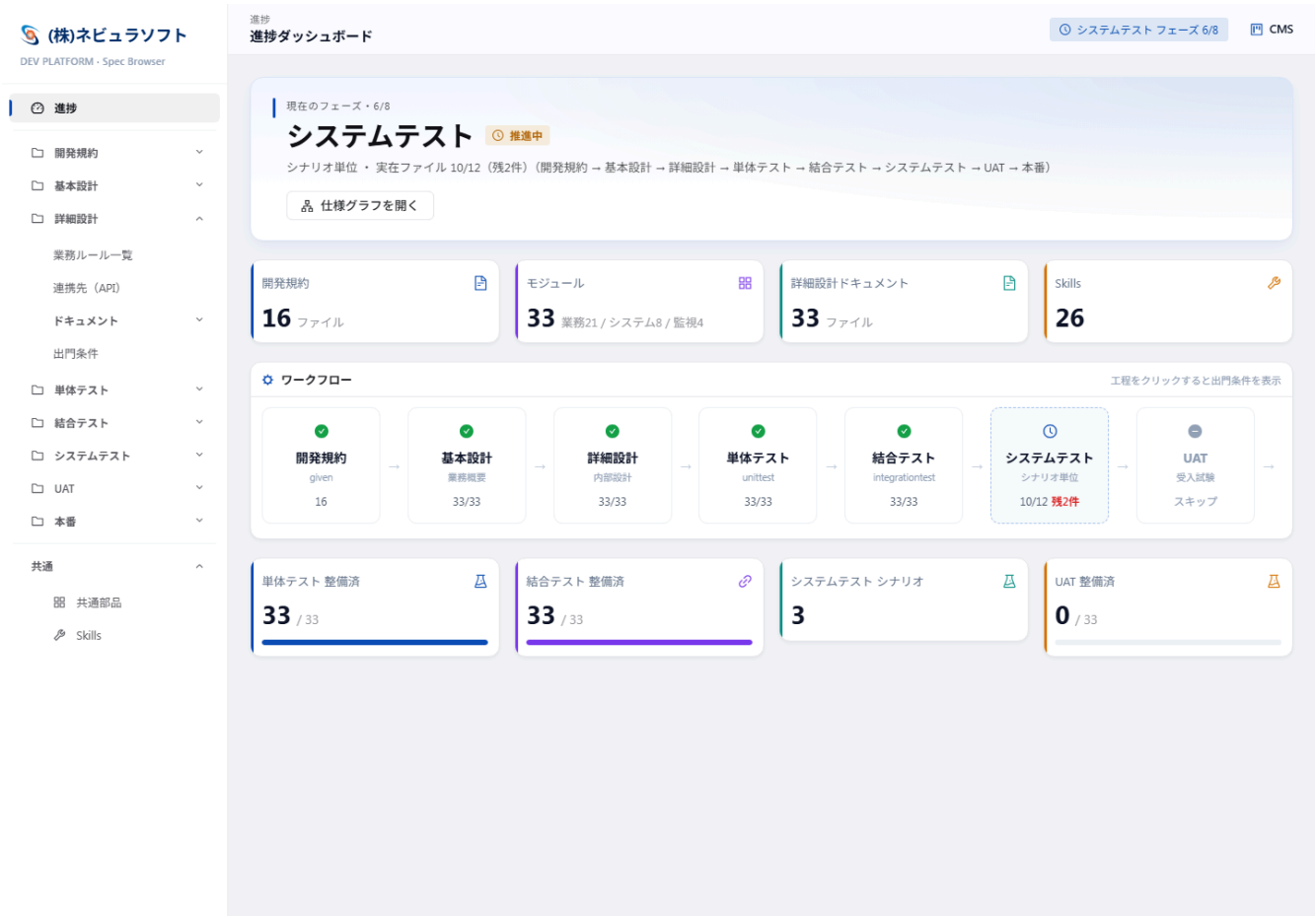


図1: 進捗ダッシュボード。成果物数だけでなく、工程の門控状態を可視化する。

3.1 規約関係図（SRG）で規約の依存と整合を確認する

3種類の関係図

関係図は用途に応じて次の3種類を切り替える。SRGの正式な台帳データ（全規約、全規則、uses、直接依存）は共通であり、概要図で辺や複合ノードを省略しても台帳自体は変更されない。

表示	用途	表示内容
概要図	全体構造の把握と移動	複合ノードと推移簡約を使用する。省略した直接参照数を画面に表示する。正式な影響判定には使用しない
直接参照図	監査、定義確認、依存関係の精査	規約を統合せず、現在の表示範囲にある全ての直接参照を表示する。正式SRG台帳の投影である
規則フォーカス	仕様変更の影響確認	定義規約 → 選択した規則 → 直接確認対象 → 関連影響候補 の順で表示する。規則IDを選択すると利用できる

「直接確認対象」は選択規則を uses に記載している規約である。「関連影響候補」はその先の下流規約であり、自動的に変更対象または確認待ちと断定するものではない。

規則本文区間とハッシュ

規則単位で変更を識別するため、定義元本文では規則ごとに開始・終了標記を付ける。

Markdown:

```
<!-- rule: naming.db-identifier-snake-case -->
ここに当該規則の本文を記載する。
<!-- /rule -->
```

YAML:

```
# rule: naming.db-identifier-snake-case
# ここに当該規則の説明を記載する。
# /rule
```

完整性確認ではこの区間を正規化して規則本文ハッシュを保存する。定義規約の別の規則だけが変わった場合、その規則を利用していない規約は確認待ちにしない。利用中の規則本文が変わった場合だけ、直接利用する規約を確認待ちにする。終了標記のない従来文書も読み取れるが、規則範囲が次の標記またはファイル末尾まで広がるため、新規・更新時は終了標記を付ける。

サイドメニューの **「規約関係図（SRG）」** では、開発規約・基本設計などの「規約」文書が、互いにどの規則を使っているかを確認できる。進捗ダッシュボードが工程ごとの達成状況を示すのに対し、規約関係図は **規則がどの規約で定義され、どの規約が使っているか** を示す。

この図は R-SDD（Relational Spec-Driven Development）の三つの原則に基づく。

原則	内容	画面での確認
AI執行	規則の追加・変更・解釈は人が決め、AIは規則に従って実行する	完整性確認と編集は人が行う（確認者が記録される）
有向非巡回	規約を点、「規約Bは規約Aの規則を直接使う」を矢印A→Bとし、循環させない	検証結果（循環があるとエラー）
SSOT	一つの規則は一つの規約でだけ定義する	検証結果（二重定義があるとエラー）

関係は `docs/srg.yaml` に記録する。各規約が定義する規則（`defines`）、使う規則（`uses`）、直接依存する規約（`dependsOn`）を人が確認して確定したものが唯一の定義元であり、本文からAIが推測して追加することはない。



図2: 規約関係図。「規約を選ぶ」で上流・下流だけを表示し、ノードをクリックすると直接つながる規約と矢印を強調し、ほかは淡色にする。

画面の構成

- **ヘッダー**: タイトル「規約関係図（SRG)」の右に検証結果（正常時は「有向非巡回、SSOT」）を表示する。警告・エラーがあるときは、その内容と色に変わる。右隣の **「バグ解析」** ボタンから障害影響分析（第12章）を開く。
- **件数**: 規約（ノード)、依存（エッジ)、規則の数を表示する。エラー・警告は、1件以上あるときだけ表示する。
- **タブ**: 関係図／規約一覧／検出事項／完整性確認／編集。

関係図の見方と操作

- **基盤／全体**: 「基盤」は開発規約と基本設計（共通・基盤・データベース）の文書を1つずつ描く。「全体」はフォルダ単位に集約して依存の本数を表示する。
- **規約の絞り込み**: 規約を選ぶと、その規約の上流と下流だけを表示し、矢印のラベルは使っている規則の数を表す。矢印は「A → B」で、BがAの規則を使う（BはAに依存する)。
- **強調表示**: ノードをクリックすると、そのノードと直接つながる規約・矢印を強調し、ほかを淡色にする。もう一度同じノードをクリックするか、図の空白をクリックすると解除する。
- **拡大縮小**: 図の右上の ＋／－ で拡大・縮小し、「全体を表示」ボタンで幅に収まる大きさへ戻す。Ctrl（Mac は ⌘）＋ホイールでも拡大縮小できる。大きな図は縮小せず、スクロールして確認する。

規約一覧・検出事項・編集

- **規約一覧**: 規約ごとの定義数、直接の依存先を一覧し、行を開くと定義する規則と使う規則を確認できる。
- **検出事項**: 検証で見つかった問題。主な内容は次のとおりである。

コード	内容と対処
CYCLE（エラー）	依存が循環している。規約の責務を分割・移動して循環を解消する
DUP_RULE（エラー）	規則が2つ以上の規約で定義されている。定義元を一つにする

コード	内容と対処
UNDEFINED_RULE / UNKNOWN_DEP（エラー）	使う規則の定義元、または依存先の規約が存在しない
MISSING_DIRECT_EDGE（エラー）	規則を使っているのに、定義元の規約へ直接依存していない（間接依存は不可）
RULE_MARKER_UNKNOWN / FOREIGN / DUP（エラー）	本文の規則ID標記が、未定義の規則、他の規約が定義する規則、または重複している
RULE_MARKER_MISSING（警告）	定義した規則の標記が本文に無い（生成物ではエラー）。本文のどこで述べているか辿れない
UNUSED_RULE（警告）	定義した規則をどの規約も使っていない。使う側の記載漏れか、不要な規則
UNUSED_EDGE（警告）	依存があるのに、その根拠となる規則の利用が無い

- **編集**: `srg.yaml` を画面で編集できる。保存前に検証を実行でき、検証エラーがあっても途中状態として保存はできるが、結果に表示される。

規則IDの標記

定義元の規約本文には、その規則を述べる箇所の直前に規則IDの標記を置く（Markdownでは `<!-- rule: tech.jdk-version -->`、YAMLでは `# rule: tech.jdk-version`）。Markdownの標記は表示では見えない。YAMLの標記はソース上では見える（データ構造には影響しない）。標記により、`srg.yaml` の定義と本文を突き合わせ、「規則がどこで述べられているか」を追跡できる。本文では、他の規約が定義する規則を転記せず、規則または定義元の規約を参照する。

完整性確認

規約が変更されたら、その規約と、依存で下流にあたる規約すべてを、確認を終えるまで「生成の根拠として使わない」。**「完整性確認」タブ**で、変更の影響範囲を確認する。

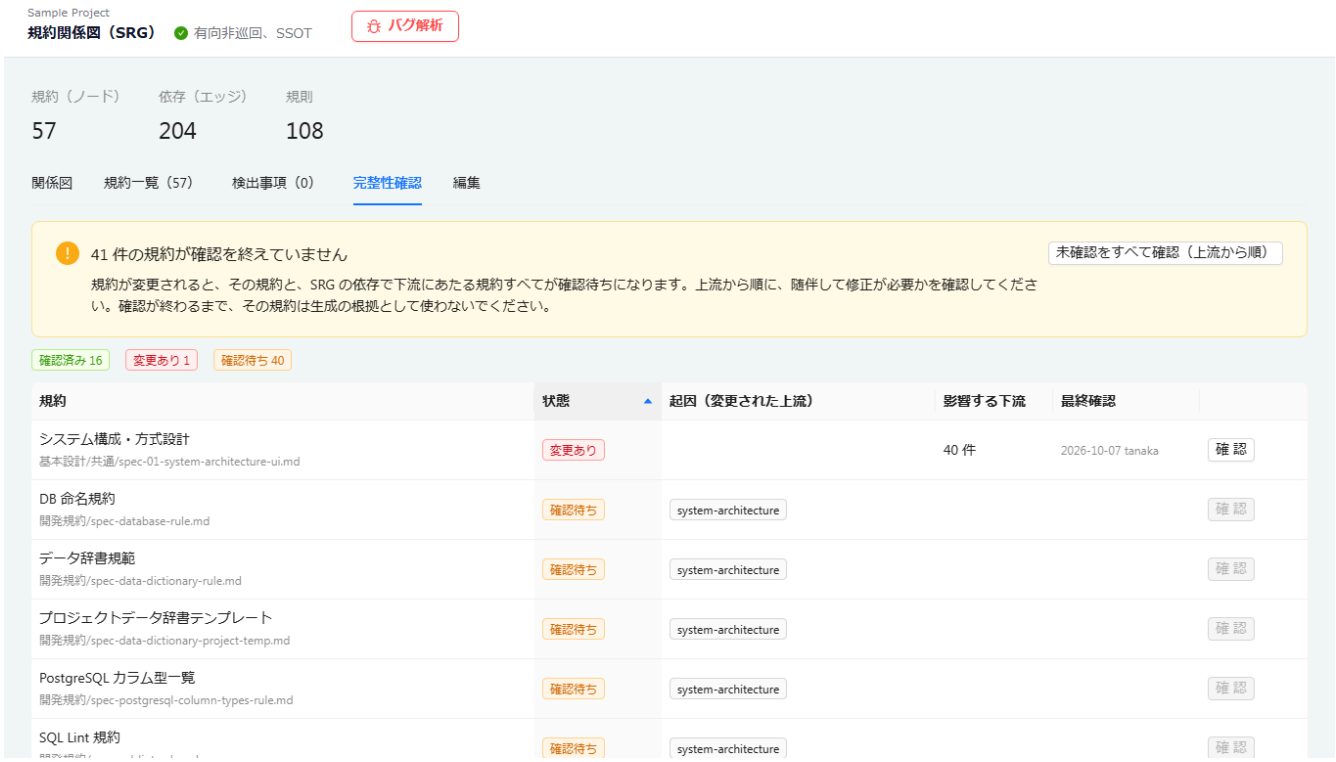


図3: 完整性確認。上流の規約が変更され、下流の規約が「確認待ち」になっている例。

状態	意味
確認済み	確認後に変更が無く、上流も変わっていない。根拠として使える
変更あり	確認後に、その規約の本文が変更された
確認待ち	規約自身は変更されていないが、上流の規約が変更されたため、随伴して修正が必要か確認が要る
未確認	一度も確認されていない
ファイルなし	規約のファイルが見つからない

1. 規約を変更したら、まず変更した規約を確認する。その下流が「確認待ち」になる。
2. 各規約について、上流の変更に合わせて修正が必要かを確かめ、**上流から順に**「確認」する。上流が未確認の規約は、確認できない（先に確認すべき上流がツールチップに表示される）。
3. 「未確認をすべて確認（上流から順）」で、一括して順に確認できる。変更内容と影響を確認した上で実行する。
4. 確認者と確認日は記録される。すべて「確認済み」になるまで、確認待ちの規約は生成や設計判断の根拠にしない。

なお、改行コードの違い（CRLF/LF）だけでは変更とみなさない。規約関係図に検証エラー（循環、二重定義など）がある間、または `srg.yaml` が下書き（`draft`）の間は、完整性確認は使えない。先に検出事項を解消する。

規約関係図の業務価値

利用場面	確認できること	得られる効果
設計レビュー	規則がどの規約で定義され、どこで使われているか	規則の二重定義・食い違いの早期発見
変更影響分析	変更した規約から、影響を受ける下流の規約を逆引きできるか	改修漏れ・修正漏れの削減
門控確認	上流の規約の変更が、下流へ確認済みで反映されているか	形式的な確認の防止
生成の根拠管理	確認を終えていない規約を根拠に生成していないか	AI生成物の信頼性の確保
引継ぎ・監査	規則の出所と、誰がいつ確認したかを第三者が追跡できるか	属人化の抑制と説明責任の確保

SEの門控確認での使い方

SEは各工程の「確認」を行う前に、進捗数値に加えて規約関係図も確認する。

1. ヘッダーの検証結果に、エラー・警告が出ていないことを確認する。
2. 対象の規約を選び、上流の規約へさかのぼれること、下流の規約へ接続されていることを確認する。
3. 「完整性確認」に「変更あり」「確認待ち」の規約が残っていないことを確認する。
4. 変更案件では、影響先として表示された規約が、すべて確認済みになっていることを確認する。
5. 規約間の関係が不足しているときは、門控を通過させず、規約または `srg.yaml` を是正する。

規約関係図は参考図ではなく、規約間の整合と変更影響を確認するための実務上の証跡である。

工程を実施しない場合は、成果物を偽装せず、理由・承認者・日時を伴う「スキップ」として記録する。

3.2 工程門控の運用原則

1. **工程順序を強制する。** 前工程の出門条件が満たされていない間、後工程を確認済みにできない。画面、個別確認、一括確認、APIのいずれから操作しても同じ判定を適用し、SEまたは工程責任者の確認を移行条件とする。
2. **確認は成果物の存在だけで行わない。** テスト工程では実行結果がPASSであることを必須とする。
3. **確認者を明確にする。** 匿名・共有アカウントによる確認は監査証跡として扱わない。
4. **確認をGitリビジョンに結び付ける。** 確認時点の取込済みGitコミットを確認履歴に記録する。リビジョンは監査証跡であり、それだけで確認の有効・無効を判定しない。
5. **上流変更を下流へ伝播する。** 上流文書または利用中の規則が変わった場合、影響を受ける下流成果物の確認状態を「要再確認」として再計算する。既に確認済みであっても、変更前の確認をそのまま有効扱いにしない。
6. **差戻しを正常な統制として扱う。** 後続変更でテストがFAIL/MISSINGになった場合、確認状態を解除し、再確認する。
7. **例外を無記録の迂回路にしない。** 「対象外」または「条件付き通過」を登録できるのはプロジェクト管理者だけとし、理由を必須にする。操作者、日時、対象、理由を監査履歴へ残し、記録を伴わない回避手段を設けない。

8. **例外着手と工程通過を区別する。** 納期等により後工程の技術作業を先行着手する場合も、前工程の出門条件を満たすまでは後工程を確認済みにできない。正式な例外が必要な場合は、プロジェクト管理者が条件付き通過として未達項目、理由、責任者および解消条件を記録する。

これらの原則は、進捗画面の表示上の注意ではなく、サーバー側の判定にも適用される。画面を経由しない操作で工程順序や権限を回避してはならない。

3.3 SEによる門控確認

SEは「確認」ボタンを押す前に、次を確認する。

- 対象工程の必須成果物が全て存在する
- 成果物が前工程の承認済み内容と整合する
- 必須レビュー指摘が解消されている
- テスト工程では対象結果が全てPASSである
- 未解決事項、既知制約、残存リスクが明記されている
- 後続工程の入力として利用できる品質である

門控確認は形式的な押印ではない。次工程の担当者に対して「この成果物を正式な入力として使用してよい」と保証する業務行為である。



図4: 工程門控のプレビュー。未達件数を確認してから詳細へ進む。



図5: モジュール単位の門控確認。成果物・実行結果・確認状態を同時に確認する。

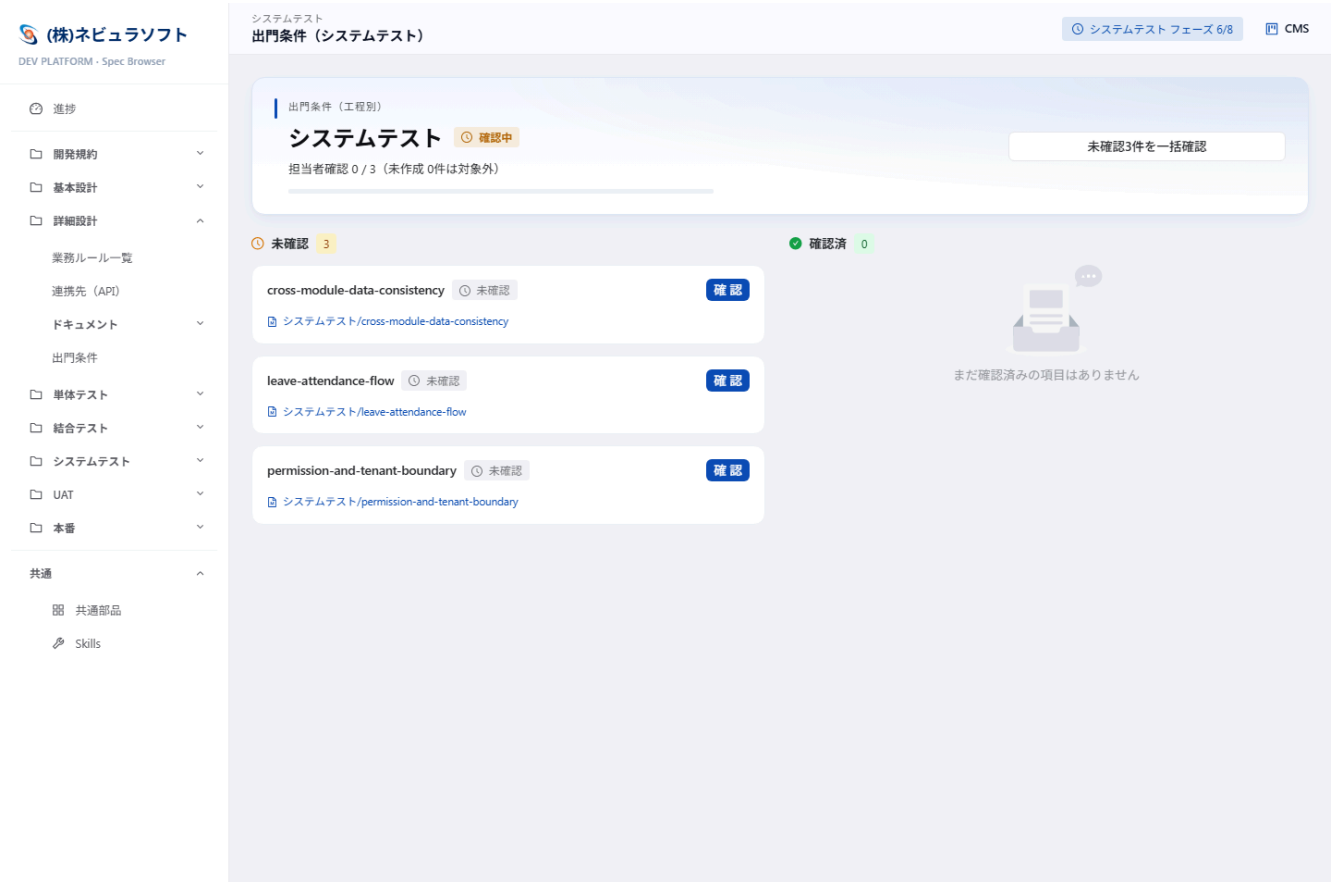


図6: シナリオ単位の門控確認。必須項目にFAIL/MISSINGが残る場合は通過できない。



図7: 工程を対象外とする場合も、理由を入力して承認可能な証跡として残す。

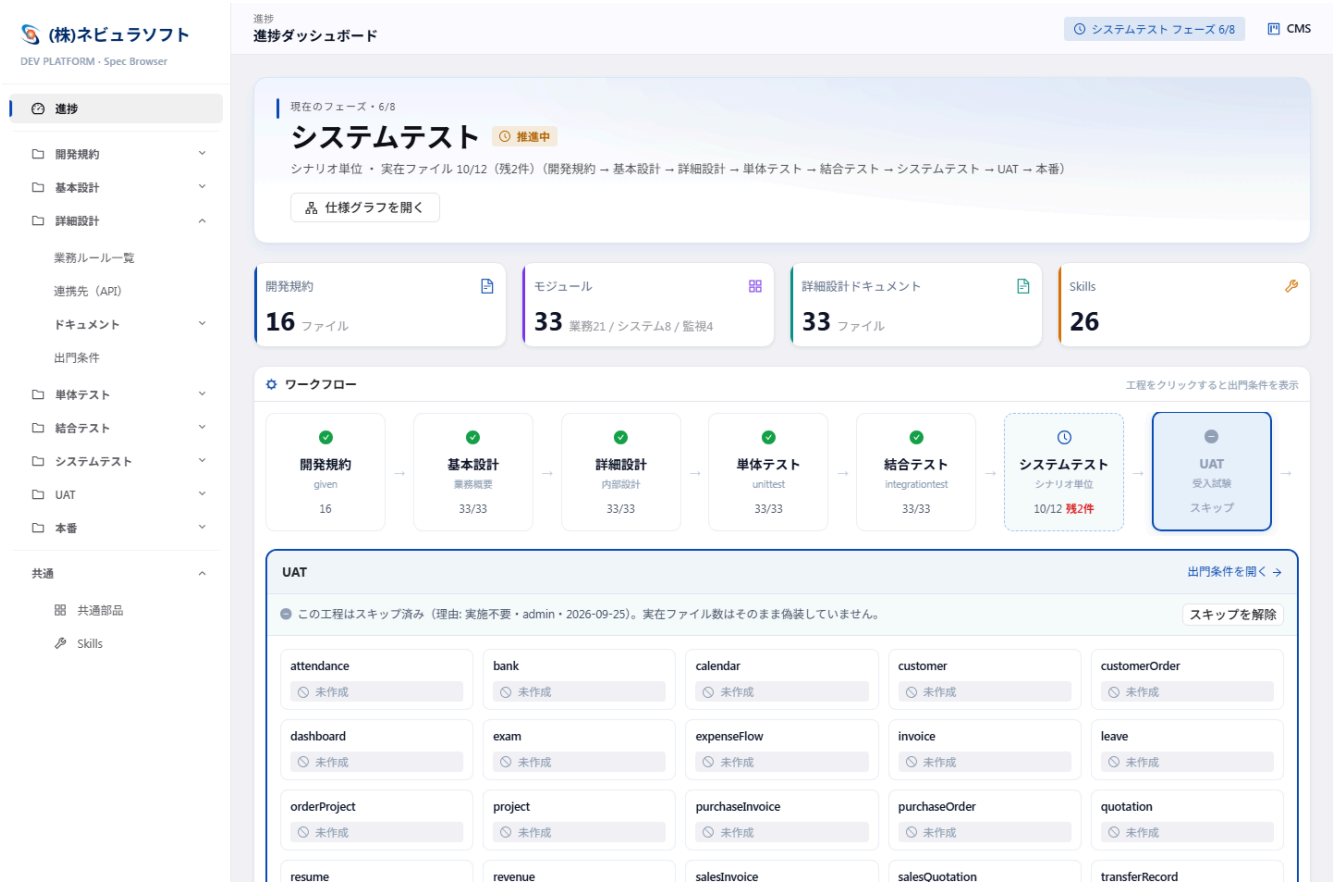


図8: スキップは完了と区別して表示し、実在成果物数を偽装しない。

3.4 管理者が見るべき指標

指標	意味	管理上の判断
成果物作成率	必要な文書・コードが存在する割合	作業量の進捗
証跡PASS率	テスト・検証が成功した割合	技術的な品質
門控確認率	SE/責任者が確認した割合	正式な工程進捗
差戻し数	確認後に無効化された件数	変更影響と品質変動
スキップ数	対象外として承認された件数	スコープ・リスク管理

成果物作成率が100%でも、門控確認率が100%でなければ工程完了ではない。この差を明示できることが、一般的な進捗表とR-SDDプラットフォームの本質的な違いである。

4. 導入前に準備する文書

4.1 必須文書

文書	決める内容	完了条件	主担当
プロジェクト定義	プロジェクトキー、表示名、対象リポジトリ、環境	一意なキーと参照先が確定	PM/PMO
用語集	業務用語、略語、同義語、禁止語	主要画面・API・DBで同じ語を利用	業務責任者
モジュール台帳	モジュールID、名称、分類、責任者、境界	全機能が一つの責任領域に所属	アーキテクト
システム関連図	フロント、API、サービス、DB、外部連携	接続方向、責任境界、データ所有者が明確	アーキテクト
成果物マップ	工程ごとの配置先、命名、必須/任意	プラットフォーム設定と一致	PMO
品質ゲート定義	工程別の確認条件、PASS基準、例外手順	判定が人によって変わらない	品質責任者
テスト結果契約	結果JSONのキー、状態、集計単位	実行ツールから自動生成可能	テスト責任者
権限・監査方針	利用者、ロール、確認権限、保管期間	最小権限と証跡要件が確定	運用責任者

4.2 導入時チェックリスト

- ☐ プロジェクトキーが他プロジェクトと重複していない
- ☐ モジュール台帳と実際のリポジトリ構成が一致している
- ☐ 同一モジュール名を全工程で使用している
- ☐ 外部システム、共有DB、共通部品の所有者が明記されている

- [ ] テスト結果の生成元と更新責任者が決まっている
- [ ] スキップ可能な工程と承認者が決まっている
- [ ] 本番URL、認証情報、接続制限がプロジェクト別運用資料に分離されている

5. 命名規約

5.1 識別子の階層

対象	形式	例	用途
プロジェクトキー	システムが小文字英数字とハイフンへ正規化して生成	`sales-core`	DB・監査記録で使用する内部識別子。通常、利用者は直接指定しない
モジュールID	lowerCamelCase	`purchaseInvoice`	全工程を横断する名寄せキー
シナリオID	kebab-case	`order-to-cash`	複数モジュール横断テスト
APIリソース	kebab-case またはプロジェクト標準	`/purchase-invoices`	外部インタフェース
DBオブジェクト	snake_case	`purchase_invoice`	永続化層
UIコンポーネント	PascalCase	`PurchaseInvoiceForm`	フロントエンド
テストケースID	`<level>-<scope>-<number>`	`ST-order-to-cash-001`	証跡・欠陥との関連付け

5.2 命名の原則

1. モジュールIDは基本設計で確定し、以後の工程で変更しない。
2. 表示名は翻訳できるが、識別子は言語ごとに変えない。
3. 略語は用語集に登録されたものだけを使う。
4. 技術名ではなく、責任を持つ業務能力またはデータ所有単位でモジュールを分ける。
5. 同義語が発生した場合は正規名を一つ決め、旧名は別名として用語集に残す。
6. 改名時は文書だけでなく、結果JSON、設定、リンク、監査キーの移行計画を作る。

5.3 成果物ファイル名

```
基本設計/<category>/<module>/spec-<module>-business-ui.md
詳細設計/<category>/<module>/spec-<module>-detail-sd.md
単体テスト/<category>/<module>/spec-<module>-unittest-sd.md
結合テスト/<category>/<module>/spec-<module>-integrationtest-sd.md
システムテスト/<scenario>/spec-<scenario>-systemtest-sd.md
UAT/<category>/<module>/spec-<module>-uat-sd.md
```

`<module>` と `<scenario>` はフォルダ名、ファイル名、結果ファイル内のキーで完全一致させる。大文字小文字、単数・複数、略称の揺れも別名として扱われるため禁止する。

6. 工程・成果物・コードの関連

```
業務要求 / 用語集
  └ 基本設計（モジュール境界・画面・業務フロー）
      ├ 詳細設計（画面/API/データ/例外/権限）
      │   ├ Frontend（画面・状態・APIクライアント）
      │   ├ Backend（Controller・Service・Repository）
      │   └ Database（テーブル・制約・マイグレーション）
      ├ 単体テスト（局所ロジック）
      └ 結合テスト（同一モジュール内の接続）
          └ システムテスト（複数モジュール/外部連携）
              └ UAT（業務受入）
                  └ 本番判定（配備・監視・ロールバック）
```

6.1 追跡マトリクス

各モジュールは最低限、次のキーで関連付ける。

追跡元	追跡先	必須キー
業務要求	基本設計	requirement ID、module ID
基本設計	詳細設計	module ID、screen/API ID
詳細設計	Frontend	route、component、API client
詳細設計	Backend	endpoint、controller、service
詳細設計	Database	entity、table、migration
設計	テスト	requirement/module/scenario ID
テスト	欠陥	test case ID、defect ID

追跡元	追跡先	必須キー
リリース	成果物	version、commit、artifact、migration

「文書がある」だけでなく、リンク先が存在し、同じ識別子で相互参照できる状態を完成とする。



図9: 共通のモジュールIDを用いて、基本設計・詳細設計・テストを横断する。

7. 品質ゲート

工程	最低条件
開発規約	必須規約が存在し、適用範囲と例外手順が明記されている
基本設計	対象モジュールが網羅され、業務責任者が確認している
詳細設計	UI/API/データ/権限/例外の整合性が確認されている
単体テスト	仕様書、コード、実行結果が一致し、対象結果がPASS
結合テスト	実接続を含む主要フローがPASS
システムテスト	シナリオ配下の必須項目が全てPASS
UAT	受入条件と業務責任者の判定が記録されている
本番	配備、監視、移行、ロールバック、承認が揃っている

テスト結果が後からFAILまたは未実施へ変わった場合、過去の確認状態を有効なまま保持しない。現在の事実を優先する。



図10: 実行結果のPASS/FAIL/MISSINGを表示し、門控確認の可否へ直接連動させる。

8. 日常の利用手順

1. 進捗ダッシュボードで現在工程と未達数を確認する。
2. 対象工程を開き、不足成果物とFAIL/MISSINGを特定する。
3. 原本リポジトリで成果物またはテストを修正する。
4. テストを実行し、所定の結果ファイルを更新する。
5. R-SDDプラットフォームを再読込し、一次情報との一致を確認する。
6. レビュー担当が出門条件を確認する。
7. 例外がある場合は、理由・影響・期限・承認者を記録する。
8. 障害や変更要求が発生した場合は「バグ解析」で影響候補を抽出し、根拠と信頼度を確認する。
9. 必要な場合だけLLM深層分析を有効にし、送信範囲と利用上限を確認して実行する。

9. 商用運用で必要な管理

9.1 セキュリティ

- 認証情報を文書、Git、テスト結果へ記録しない。

- LLM提供元のAPI Keyは「個人 LLM 設定」だけで登録し、画面キャプチャ、文書、Gitへ記録しない。
- 利用者ごとに識別可能なアカウントを発行する。
- 公開範囲はネットワーク制御とアプリケーション認証の両方で制限する。
- 確認、スキップ、解除、設定変更を監査対象とする。

9.2 可用性とバックアップ

- 確認台帳DBを定期バックアップする。
- プロジェクト設定、成果物リポジトリ、結果ファイルの復旧手順を用意する。
- ヘルスチェック、ログ監視、容量監視、証明書期限監視を行う。

9.3 変更管理

- 設定変更と製品更新を分離する。
- 更新前にバックアップ、互換性、DBマイグレーション、ロールバックを確認する。
- 更新後はログイン、プロジェクト読込、成果物数、テスト結果、確認登録をスモークテストする。

10. プロジェクト別設定として分離する情報

次の情報は共通マニュアル本文へ固定値として書かず、プロジェクト別運用補足または環境設定で管理する。

- URL、IPアドレス、ポート、公開範囲
- 認証方式、利用者、パスワード
- リポジトリの絶対パス
- モジュール数とモジュール一覧
- DB接続先と資格情報
- プロジェクト固有の工程名、例外、スキップ方針
- 外部システム名、担当組織、連絡先

11. トラブルシューティング

状況	確認事項
成果物が0件	設定パス、命名、拡張子、実行ユーザーの読取権限
同じモジュールが二重表示	module IDの大文字小文字、単複、略称の揺れ
テストがMISSING	結果ファイルの配置、category/module/scenarioキー
確認できない	必須成果物、PASS条件、確認権限
確認済みが解除された	最新テスト結果がFAIL/MISSINGへ変化していないか
工程間リンクが切れる	追跡キー、改名時の移行漏れ、対象リポジトリ設定

状況	確認事項
完整性確認が使えない	`srg.yaml` が下書き（draft）、または検証エラー（循環・二重定義など）がある。検出事項を解消する
バグ解析が0件	現在のワークスペース、解析済み状態、障害内容の具体性、ファイル除外条件を確認する
LLM深層分析を有効にできない	個人LLM設定でKeyを登録し、接続確認、既定提供元、Token上限を確認する
ワークスペース解析が中断表示	サーバー再起動等でジョブが中断された可能性がある。対象ワークスペースから再分析する

12. 障害影響分析（バグ解析）

規約関係図（SRG）画面のヘッダーにある「バグ解析」ボタン（`/srg/analyze`）から、障害内容、エラー、脆弱性名、画面名、API名、ファイル名などを入力し、仕様・コード・テストへの影響候補を抽出する。解析結果は自動的な修正指示ではなく、調査開始点として使用する。

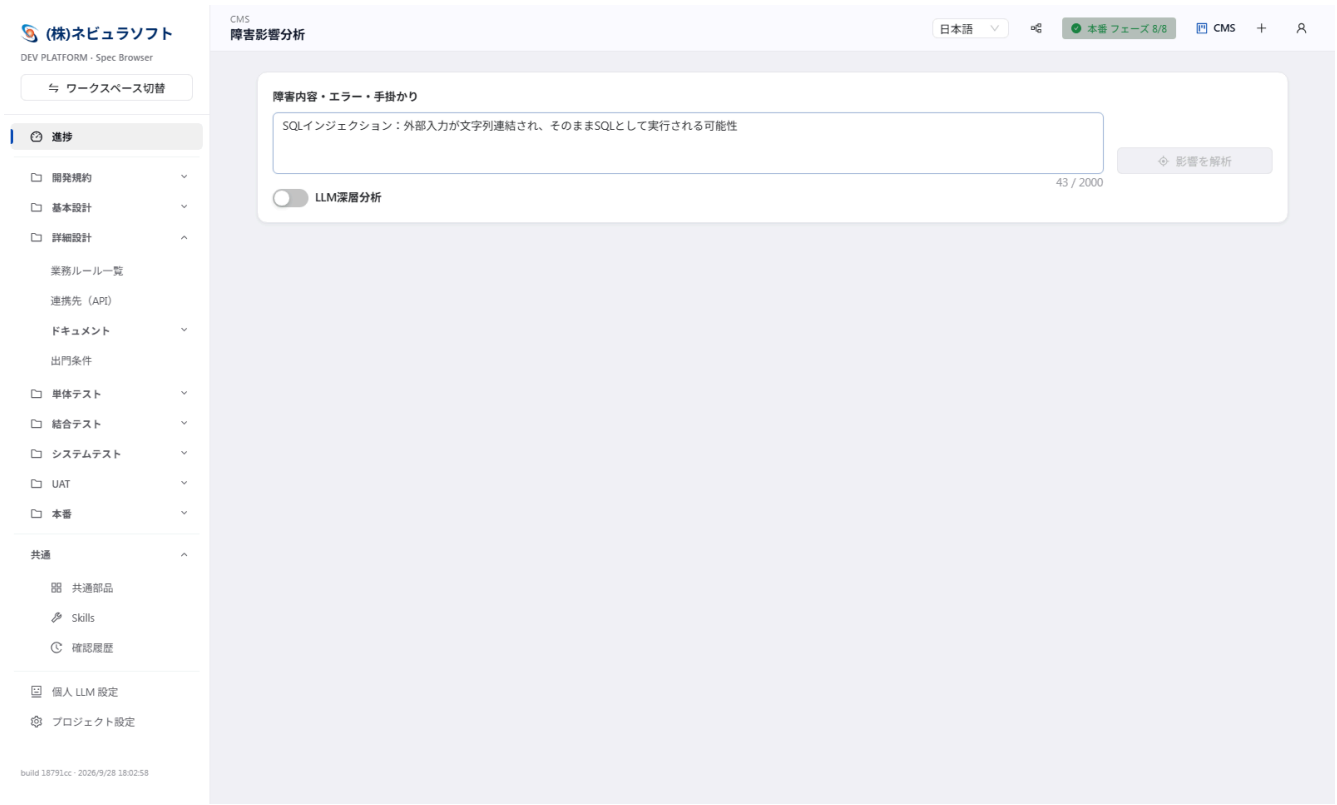


図11: 障害内容を入力する。LLMを使わない場合も、仕様文書、コード、呼出関係、DB依存関係を横断して検索する。

12.1 静的解析の基本手順

1. 現象だけでなく、対象機能、識別子、例外文、セキュリティ観点を具体的に入力する。
2. 「影響を解析」を選択する。
3. 候補ファイルの一致理由、変更可能性、根拠信頼度を確認する。

4. 「関連コードと呼出関係」を展開し、入口から下流処理、SQL・DBオブジェクトまで追跡する。
5. 間接影響候補と推奨回帰テストも確認する。

同じファイル内の複数の入力点は、メソッドまたは処理単位へ集約して表示される。件数が多い場合は初期表示を限定し、ノードを必要な範囲だけ展開する。



図12: 解析結果例。候補が得られても高確度の変更箇所が0件の場合があるため、候補文書、根拠、回帰テストを併せて判断する。

12.2 LLM深層分析

「LLM深層分析」を有効にすると、静的解析結果に加えて、選択した提供元がコード抜粋と関係情報を解釈する。利用前に次を確認する。

- API Keyが現在のユーザーに設定されていること。
- 送信対象のファイル数、概算文字数、機密候補の除外・マスク件数。
- 利用する提供元、モデル、分析深度、Token上限。
- リポジトリの情報を外部LLMへ送信できる契約・社内ルールであること。

LLMの結果には「LLM推定」の出所を付け、静的根拠と分離して保存する。双方が同じ関係を確認した場合は「双方確認」として表示する。LLMジョブはバックグラウンド実行され、画面からキャンセルできる。

13. 個人 LLM 設定

サイドバーの「個人 LLM 設定」で、OpenAI / CodexまたはAnthropic / ClaudeのAPI Keyと分析既定値を管理する。Keyと設定はユーザー単位で分離され、他の利用者および他プロジェクトへ共有されない。

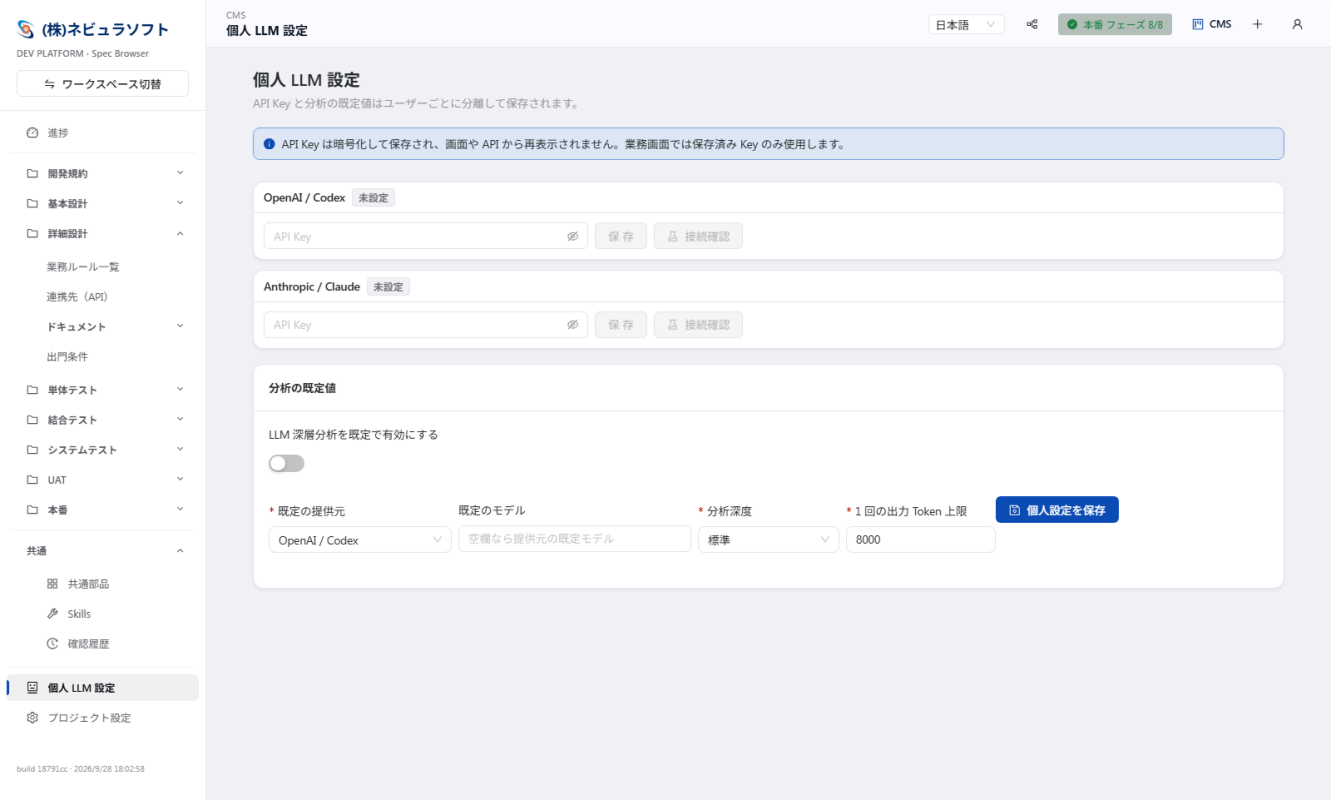


図13: 提供元ごとのKey状態と、既定提供元、モデル、分析深度、出力Token上限を管理する。

- 保存後のKeyは暗号化され、画面/APIから再表示されない。画面には設定状態と末尾のみを表示する。
- 「接続確認」で提供元への疎通とKeyの有効性を確認する。
- Keyを変更する場合は新しいKeyを入力して「置換」する。不要になったKeyは「削除」する。
- 業務画面ではKeyを直接入力せず、ここで保存したKeyだけを使用する。
- 「LLM深層分析を既定で有効にする」は個人の初期値であり、実行画面で変更できる。

14. プロジェクトとワークスペース

14.1 プロジェクト

一般ユーザーは、ユーザー登録時にはプロジェクトを指定しない。メール確認後に初めてログインし、所属プロジェクトがない場合は「新しいプロジェクトを作成」するか、プロジェクト管理者から届いた招待メールのリンクから参加する。新規作成した利用者は、そのプロジェクトのプロジェクト管理者になる。既存プロジェクトへの参加は招待が必須であり、プロジェクト名の入力だけでは参加できない（プラットフォーム管理者が `DEVTOOL_ALLOW_JOIN_BY_NAME=true` を設定した環境に限り、プロジェクト名を正確に入力して参加できる）。プロジェクトIDとプロジェクトキーはシステムが内部的に生成するため、連番や表示名を識別子として扱わない。

一度選択したプロジェクトは利用者ごとに記録され、次回ログイン時は原則として前回のプロジェクトを開く。複数プロジェクトへ所属している場合は、ヘッダーのプロジェクト切替から変更する。

本プラットフォームでは、**1プロジェクト ＝ 1ワークスペース ＝ 1 GitHubリポジトリ**として管理する。別のリポジトリを管理するときは、新しいプロジェクトを作成する。

確認履歴、個人設定、解析状態、キャッシュはプロジェクトおよびユーザーのコンテキストで分離される。画面上部のプロジェクトを切り替えても、別プロジェクトの確認状態を上書きしない。

14.2 GitHubワークスペースの取込と解析

「プロジェクトリポジトリ」では、GitHubリポジトリURLを指定してサーバーへ取り込み、仕様・コード・DBオブジェクト・呼出関係を解析する。公開リポジトリは認証不要であり、非公開リポジトリではPersonal Access Tokenを入力する。新規プロジェクトでまだリポジトリを登録していない場合は、初回取込時にGitHub URLを入力する。既存リポジトリの再取込で認証が失敗した場合は、Personal Access Tokenを再入力して実行する。

Personal Access Tokenはプロジェクト情報として永続保存されない。取込処理中のみ暗号化されたタスクデータとして保持され、処理の完了、失敗、取消または再試行終了後に削除される。

取込と再取込は、担当者が開始する**人工のレビュー批次**である。プラットフォームはプロジェクトリポジトリを定期的に自動取込せず、GitHub上の変更を自動的に工程確認へ反映しない。担当者は、ローカル作業をコミット・プッシュした後、確認対象とするコミットを確かめてから再取込を開始する。取込完了後、プラットフォームは新しいスナップショットに基づいて成果物、規約完整性、上流・下流の影響、テスト結果および工程状態を再計算する。

この運用により、一回の取込を一つのレビュー境界として扱う。取込後にGitHubへ追加された変更は、次の再取込を実施するまでプラットフォームの判定対象にならない。確認履歴に表示されるGitリビジョンを使用し、「どのコミットに対する確認か」を監査できる。

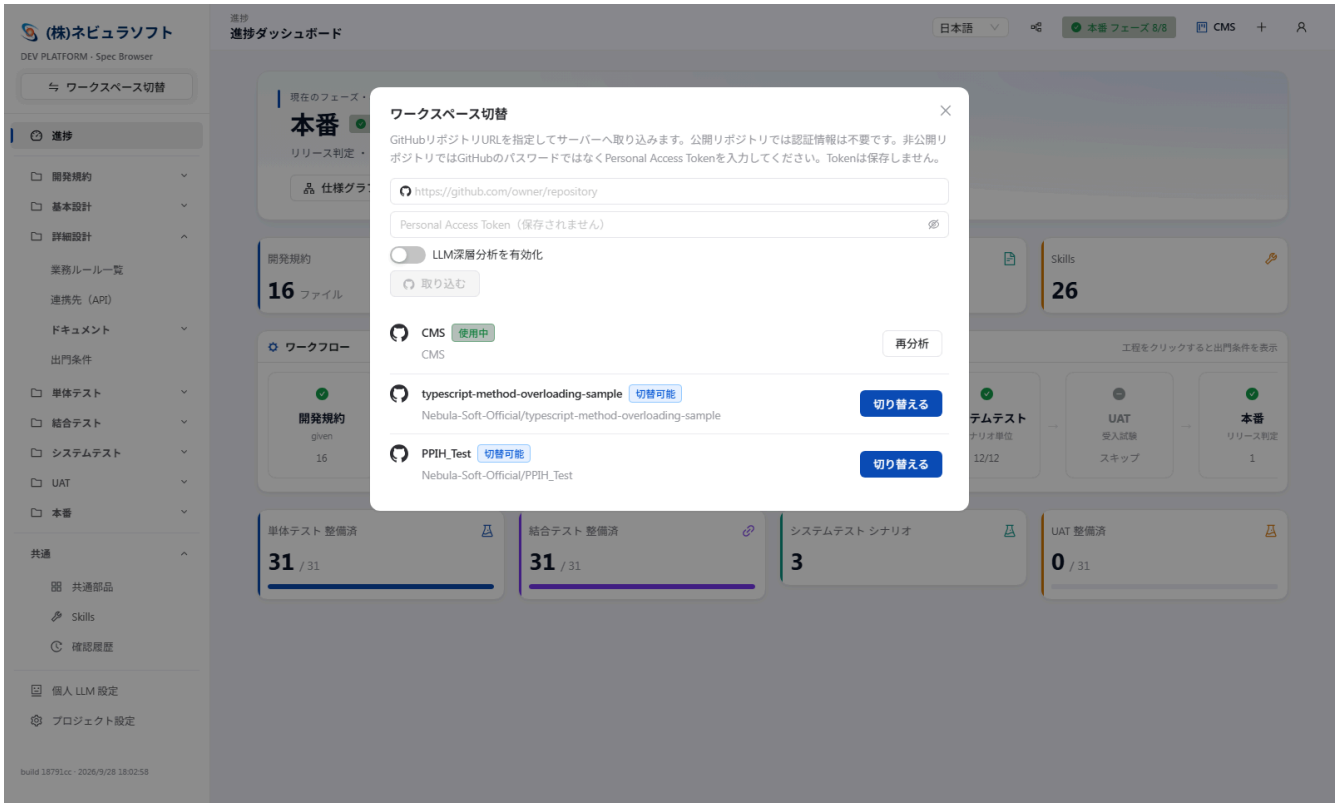


図14: 現在のワークスペース、切替可能なワークスペース、再分析を一つの画面で管理する。

1. リポジトリURLを入力する。
2. 非公開リポジトリの場合だけTokenを入力する。

3. 必要な場合はLLM深層分析を有効にする。
4. 「取り込む」を選択し、解析完了まで待つ。
5. 既存ワークスペースは「切り替える」、現在のワークスペースは「再分析」を使用する。

解析中にサーバーが再起動したジョブは中断として扱われる。表示された状態を確認し、同じワークスペースで再分析する。

14.3 Spec（文書）の生成と共有（ローカルで生成し、GitHub 経由で共有する）

規約・基本設計などの文書のひな形は、**自分の PC のフォルダへ生成**する。サーバーには書き込まない。サーバーが内容を受け取る手段は、GitHub から取込（再取込）したときの **スナップショット**（その時点の写し）だけである。

```
自分の PC で生成 → 自分でコミット・プッシュ → GitHub
                                                ├ 他の人が取込 → そのまま利用できる
                                                └ 再取込 → サーバーのスナップショットが最新になり、分析・表示に反映
```

1. プロジェクト開始時に、「ワークスペース切替」で対象のワークスペースの **「初版Spec生成」** を選ぶ。この操作は公開済み Template から初版を作るためのものであり、業務に合わせて編集を始めた後の日常的な再生成には使用しない。
2. 書き込み先の自分の PC のフォルダ（GitHub リポジトリのルート）を選ぶ。Chrome / Edge で、https のアドレスから開いた場合に使える。
3. 技術スタックとプロジェクト名などを確認し、「プレビュー」で内容を確認してから「ローカルに書き込む」を選ぶ。手で編集したファイルは上書きしない。
4. 生成されたファイルをコミット・プッシュする。共有しない場合はプッシュしなくてよい。

すでに独自の文書構成（`devtool.config.yaml`）を持つ既存プロジェクトのフォルダ（例: 本番稼働中のプロジェクト）には、書き込めない。その場合は別のフォルダへ生成する。

スナップショットと再取込

- ワークスペース切替の一覧と、バグ解析の画面には、サーバー上のスナップショットが **どのコミットか**（コミット、日時、取込日時）を表示する。分析や表示は、このスナップショットに対して行う。
- GitHub にプッシュした内容を反映するには、**「再取込」** を選ぶ。GitHub の最新の内容でスナップショットを更新する。非公開のリポジトリでは、Token の入力が求められる（保存しない）。
- ローカルで生成・修正してもプッシュしていない内容は、スナップショットには含まれない。バグ解析などの結果に反映したいときは、先にプッシュして再取込する。
- GitHub のリポジトリにまだコミットが無い場合は、再取込の際にその旨が表示される。
- リポジトリ名を GitHub 上で変更した場合は、新しい名前で取込み直す（別のワークスペースとして登録される）。古いワークスペースは使わない。

15. AI協働による影響分析

障害影響分析の 「**LLM深層分析**」 を有効にし、「**AI協働**」 を選択すると、異なる2つのAIを順番に実行できる。メインAgentが対象ワークスペースを読み取り専用で分析し、サブAgentがその結論を再確認して、見落とし、誤り、リスク、改善案を補足する。AIの回答は判断材料であり、成果物の修正、工程確認、リリース承認を自動で行うものではない。



図15: AI協働では利用方式、メインAgent、サブAgentを設定し、2つの出力を並べて確認する。

15.1 利用方式を選ぶ

利用方式	事前準備	実行場所	主な用途
企業アカウント	Windows用Runnerの導入、Runnerの起動、Claude CodeとCodexへのログイン	利用者PC上のRunner	企業契約済みのCLIアカウントを利用する場合
企業API Key	「個人 LLM 設定」でOpenAIとAnthropicのAPI Keyを登録	R-SDDプラットフォームサーバー	企業管理のAPI Keyでサーバー実行する場合

どちらの方式でもメインAgentとサブAgentには異なるAIを指定する。たとえば、メインAgentをClaude、サブAgentをCodexとした場合、Claudeの分析完了後にCodexが同じタスクとメインAgentの結論を再確認する。



図16: 企業API Key方式ではOpenAIとAnthropicの両方が設定済みであることを確認する。

15.2 Agent Runnerとは何か

Agent Runnerは、**R-SDDプラットフォームから依頼されたAI協働タスクを利用者のWindows端末で受け取り、端末上でログイン済みのClaude CodeとCodexを実行する中継プログラム**である。企業アカウント方式でのみ使用し、企業API Key方式では使用しない。

```
R-SDDプラットフォーム
  │  タスク登録・実行状況表示
  ▼
Agent Runner（利用者のWindows端末）
  ├─ 対象ワークスペースの読み取り専用副本を取得・キャッシュ
  ├─ メインAgent（Claude CodeまたはCodex）を実行
  ├─ メインAgentの結果をサブAgentへ渡して再確認
  └─ 状態・ログ・最終結果をR-SDDプラットフォームへ返送
```

Agent Runnerを使用する理由

- **企業契約のCLIログインを利用するため**: Claude CodeとCodexのログイン情報をR-SDDプラットフォームサーバーへ登録せず、ログイン済みの端末上で各CLIを実行できる。
- **認証情報を端末外へ出さないため**: Claude Code / Codexのセッション、パスワード、TokenをR-SDDプラットフォームへアップロードしない。
- **2つのAgentを順番どおり実行するため**: メインAgentの完了結果をサブAgentへ安全に引き渡し、独立した再確認を実施する。

- **実行場所を管理するため**: どの端末がオンラインで、どのAIが利用可能かをR-SDDプラットフォームへ通知し、対象ユーザー・プロジェクトのタスクだけを取得する。
- **大きなワークスペースの転送を減らすため**: 初回に取得した副本を端末内へキャッシュし、同一ワークスペースでは再利用する。

Agent Runnerが行うこと／行わないこと

行うこと	行わないこと
R-SDDプラットフォームへのペアリングと定期的なオンライン通知	Claude Code / Codexの認証情報をR-SDDプラットフォームへ送信すること
自分に割り当てられたAI協働タスクの取得	他ユーザー・他プロジェクトのタスクを取得すること
ワークスペース副本の取得、保存、再利用	原本ワークスペースを直接変更すること
Claude Code / Codexの読み取り専用分析	ファイル変更、コミット、Push、リリースを行うこと
分析状態、ログ、結果の返送	AIの結論を承認済みの業務判断として登録すること

Runnerは15秒間隔でオンライン状態を通知し、タスクを定期取得する。画面の 「**Runnerオンライン**」 は、直近の通知とClaude Code / Codexの利用可能状態を基に表示される。PCのスリープ、ネットワーク切断、Runner終了、CLIのログアウトが発生すると実行できない。

> Agent Runnerは常駐型のAIそのものではない。R-SDDプラットフォームと端末上のClaude Code / Codexをつなぎ、タスク順序、読み取り専用ワークスペース、ログ返送を管理する実行窓口である。

15.3 企業アカウント方式の初回準備

1. 「**Runnerを取得**」 を選択し、Windows用インストーラーをダウンロードする。
2. ダウンロードしたコマンドファイルを実行する。画面に表示された短時間有効なペアリングコードにより、Runnerが現在のユーザーとプロジェクトへ関連付けられる。
3. Runner端末でClaude CodeとCodexへ、それぞれ会社で許可されたアカウントを使ってログインする。
4. Runnerを起動し、AI協働設定に 「**Runnerオンライン**」 と表示されることを確認する。
5. Runner更新の案内が出た場合は、新しいインストーラーを取得して更新してから実行する。

Runnerは対象ワークスペースの読み取り専用副本を利用する。同じワークスペースはローカルキャッシュを再利用するが、Agentにはファイル変更、コミット、Pushを行わせない。Runner端末を共有する場合も、ログインアカウント、端末管理、作業ログの取扱いを社内規程に合わせる。

15.4 企業API Key方式の初回準備

1. サイドバーの 「**個人 LLM 設定**」 を開く。
2. OpenAIとAnthropicの企業API Keyをそれぞれ登録し、接続確認を行う。
3. 障害影響分析へ戻り、利用方式で 「**企業API Key**」 を選ぶ。
4. 「**企業API Keyを利用できます**」 と表示されることを確認する。

API Keyはユーザー単位で分離される。画面キャプチャ、文書、Git、Agentへの指示文へKeyを記載しない。利用枠、送信可能なコード範囲、外部LLM利用条件は、実行前に所属組織のルールを確認する。

15.5 分析を実行する

1. 障害内容、エラー、対象機能、識別子、再現条件などを具体的に入力する。
2. 「**LLM深層分析**」 を有効にし、「**AI協働**」 タブを選択する。
3. 利用方式、メインAgent、サブAgentを選び、「**設定を保存**」 を選択する。
4. 準備状態に警告がないことを確認し、「**影響を解析**」 を選択する。
5. メインAgentの分析完了後、サブAgentの再確認結果を読む。結論が異なる場合は、対象ファイルと静的解析根拠へ戻って確認する。
6. 実行履歴から過去のタスク、状態、使用したAgentの順序を確認する。実行中の履歴は削除できない。

表示状態は **待機中、実行中、完了、失敗** の4種類である。失敗時は該当するAgent側に理由が表示される。利用枠不足、ログイン切れ、タイムアウト、回答未完了などを確認し、原因を解消してから新しい分析を実行する。

15.6 結果の読み方と業務上の注意

- **メインAgent**: 最初の調査結果、根拠、影響候補、推奨確認事項を提示する。
- **サブAgent**: メインAgentの結論を独立して再確認し、漏れ、誤り、追加リスク、より安全な案を提示する。
- **静的解析との照合**: AI出力だけで影響範囲を確定せず、ファイル、呼出関係、DB依存、テスト結果、動的仕様グラフと照合する。
- **機密情報**: パスワード、Token、秘密鍵、個人情報、未承認の顧客データをタスクへ入力しない。
- **最終責任**: 修正方針、テスト範囲、工程通過、リリース可否は担当者および承認者が決定する。

15.7 AI協働のトラブルシューティング

状況	確認事項
AI協働を開始できない	障害内容が入力済みか、異なる2つのAgentが選択されているか
「企業API Keyを設定してください」	個人 LLM 設定でOpenAIとAnthropicの両方を登録し、接続確認する
「Runnerの起動とログインが必要」	Runnerプロセス、オンライン表示、Claude CodeとCodexのログイン状態を確認する
Runner更新が必要	「Runnerを取得」から最新版を再導入し、Runnerを再起動する
利用枠不足	対象AIの企業アカウントまたはAPI契約の利用枠を管理者へ確認する
片方のAgentだけ失敗	失敗側パネルの理由を確認する。成功側の結果だけで確定せず、復旧後に再実行する
実行が終わらない	Runnerの接続、ネットワーク、提供元の稼働状況、30分の実行上限を確認する

16. 初回利用、ログインおよびアカウント管理

16.1 ログイン方式

ログイン画面では、利用者の所属に応じて次の方式を選択する。

利用者	ログイン方式	アカウントの作成
本社社員・社内利用者	Microsoft企業アカウント	Microsoft Entra IDで初めてログインしたときに、プラットフォーム内の連携アカウントを自動作成する
一般ユーザー・外部ユーザー	ユーザー名とパスワード	ログイン画面の「ユーザー登録」から登録する

企業アカウントは本プラットフォーム用に設定された単一のEntraテナントだけを受け付ける。他社のMicrosoftアカウントは、そのまま企業アカウントとしてログインできない。企業ユーザーのパスワードとMFAはMicrosoft Entra側で管理する。

一般ユーザーは、ユーザー名、メールアドレス、パスワード、画面に表示された検証コードを入力し、利用規約とプライバシーポリシーを確認して登録する。登録画面にプロジェクトの入力欄はない。

16.2 メール確認とパスワード再設定

一般ユーザーは、登録後に届く確認メールのリンクを開いてメールアドレスを確認する。確認が完了するまでは通常ログインできない。メールが届かない場合は、ログイン画面の「確認メール再送」を利用する。連続送信を防ぐため、再送には待機時間が設けられている。

パスワードを忘れた場合は「パスワードを忘れた」を選択し、登録メールアドレスへ届く期限付きリンクから新しいパスワードを設定する。パスワード再設定用のリンクやコードを第三者へ転送してはならない。

16.3 一般ユーザーのMFA設定

一般ユーザーは、メール確認後7日以内にMFAを設定する。期限までに設定しなかったアカウントはロックされる。

1. 右上のユーザーメニューから **設定** > **アカウント** を開く。
2. MFA欄の「MFA設定」を選択する。
3. Microsoft AuthenticatorなどのTOTP対応アプリでQRコードを読み取る。
4. 認証アプリに表示された6桁コードを入力する。
5. コードが更新された後、次の6桁コードをもう一度入力する。
6. 表示された復旧コードを安全な場所に保管する。

MFA設定後は、ユーザー名とパスワードの確認に成功した後、別のMFA確認画面が表示される。認証アプリの6桁コード、または未使用の復旧コードを入力してログインする。同じ6桁コードを繰り返し使用することはできない。

MFAを再設定するときは、アカウント画面の「MFA設定変更」から現在のMFAコードを入力して再登録する。端末を紛失し、現在のコードも復旧コードも利用できない場合は、プラットフォーム管理者へMFAリセットを依頼する。管理者によるリセット時は全セッションが無効化され、利用者には新たに7日間の設定期限が付与される。

> Microsoft企業アカウントでは、プラットフォーム内のMFA設定を使用しない。Microsoft Authenticatorの通知や数字の一致確認を含む認証方法は、所属組織のEntraポリシーに従う。

16.4 アカウント設定

右上のユーザーメニューは次の構成とする。

- **設定 > アカウント**: ユーザー名、メールアドレス、ログイン方式、状態、メール変更、パスワード変更、MFA、個人データ出力、アカウント退会、セッション情報
- **設定 > LLM設定**: 個人API Key、利用方式、既定モデルおよび分析パラメータ
- **言語**: 表示言語の変更
- **ダークモードへ**: 表示テーマの変更
- **ログアウト**: 現在のセッションを終了

パスワード変更は一般ユーザーだけに表示される。企業ユーザーのパスワードはMicrosoft Entraで管理する。プラットフォーム管理者は、管理不能を防ぐため自分自身のアカウントを画面から退会できない。

16.5 セッション

- 一般ユーザーのセッションは最長8時間である。
- Microsoft Entraによる企業ユーザーのセッションは最長1時間である。
- 30分間操作がない場合はセッションが失効する。
- 重要操作では、直近のログイン確認を要求する場合がある。
- ログアウト時はプラットフォームのセッションを終了する。企業ユーザーではEntraのシングルログアウトも実行する。
- 管理者による強制ログアウト、アカウント凍結、MFAリセット時は既存セッションが無効化される。

17. プロジェクト参加、メンバーおよび権限

17.1 所属プロジェクトがない場合

初回ログイン時に所属プロジェクトがない場合は、次のいずれかを選択する。

- **新しいプロジェクトを作成**: プロジェクト名を入力する。作成者はプロジェクト管理者になる。
- **既存プロジェクトへ参加**: プロジェクト管理者に招待を依頼し、届いた招待メールのリンクを開いて参加する。招待で付与される権限は、招待時に管理者が指定した初期権限（プロジェクト管理者は指定できない）である。

`DEVTOOL_ALLOW_JOIN_BY_NAME=true` を設定した環境では、画面で参加先のプロジェクト名を正確に入力して参加できる。この場合の初期権限は閲覧者で、必要な権限はプロジェクト管理者が設定する。誤ったプロジェクトへ参加しないよう、プロジェクト名はプロジェクト管理者から確認する。

17.2 権限一覧

権限	主な操作
プラットフォーム管理者	プラットフォーム利用者、状態、MFAリセット、セッション、監査、テンプレート、監視、全体回収領域を管理する。個別プロジェクトの基本設定は変更しない
プロジェクト管理者	担当プロジェクトの設定、リポジトリ、メンバー、招待、回収領域を管理する
規約編集者	仕様・規約など、許可されたプロジェクト成果物を編集する
確認者	成果物と品質ゲートを確認し、確認結果を登録する
閲覧者	プロジェクトの内容を参照する

権限運用では次の制約を守る。

- 利用者は自分自身のプロジェクト権限を変更できない。
- プロジェクト管理者への昇格またはプロジェクト管理者からの変更は、プラットフォーム管理者だけが行う。
- プロジェクト管理者は、担当プロジェクトの一般メンバーを管理できる。
- プラットフォーム管理者は通常のプロジェクトメンバー一覧には表示しない。
- 最後のプロジェクト管理者を削除、退出または退会させてはならない。先に別の管理者を設定する。
- プラットフォーム共通値はテンプレート設定で管理し、プラットフォーム管理者が個別プロジェクト設定を直接上書きしない。

17.3 招待とメンバー管理

プロジェクト管理者は、プロジェクト設定からメールアドレスと初期権限を指定してメンバーを招待できる。招待メールのリンクは7日間有効であり、招待先と同じメールアドレスのアカウントでログインして承認する。不要になった未承認招待は取り消す。

参加後は、権限変更、メンバー削除、プロジェクト退出を行える。ただし、自分自身の権限変更と最後のプロジェクト管理者の削除・退出はできない。プロジェクト管理者を交代するときは、プラットフォーム管理者へ権限変更を依頼してから旧管理者を退出させる。

18. リポジトリ取込、タスク、容量および回収領域

18.1 取込と再取込

初回取込では、対象プロジェクトに登録するGitHub URLを確認する。非公開リポジトリではPersonal Access Tokenを入力する。再取込は現在のプロジェクトに登録されているURLを使用し、認証に失敗した場合はToken入力画面から再実行する。

取込や解析はバックグラウンドタスクとして実行される。主な状態は次のとおりである。

状態	意味と対応
待機中	Workerの実行待ち。重複して開始しない
実行中	取込または解析中
再試行中	一時的な障害後に自動再試行を待っている
完了	正常終了
失敗	認証、URL、容量、解析エラーなどを確認して再実行する
タイムアウト	規定時間内に終了しなかった。対象リポジトリの規模とログを確認する
取消済み	利用者またはシステムにより取り消された

タスク情報はデータベースへ保存される。WebコンテナやWorkerが再起動した場合も、実行期限と再試行回数の範囲で復旧または再試行される。最終的に失敗またはタイムアウトになった場合は、原因を解消して利用者が再実行する。

18.2 標準制限

標準構成では、次の制限を設ける。運用環境で値が変更されている場合は、画面表示と運用担当者の案内を優先する。

- 1ユーザーが作成・管理できるプロジェクト数: 5
- 1プロジェクトのGitHubリポジトリ数: 1
- 取込対象リポジトリの最大容量: 2 GiB
- 1プロジェクトの最大使用容量: 5 GiB
- 1ユーザーおよび1プロジェクトの実行待ちタスク数: 各5
- サーバー空き容量が安全基準を下回る場合、新しい取込を停止する

上限へ到達した場合は、不要なプロジェクトや回収領域を整理するか、プラットフォーム管理者へ連絡する。

18.3 回収領域

削除したワークスペースデータは直ちに物理削除せず、回収領域へ移動する。プロジェクト管理者は、自分のプロジェクトの回収領域を確認し、保留期間内であれば復元できる。

プラットフォーム管理者は、全プロジェクトの回収領域、使用容量、復旧期限を確認し、必要に応じて復旧期限の延長、強制的な永久削除、復旧失敗の確認を行う。永久削除後は画面から復元できないため、対象プロジェクトとバックアップの有無を確認してから実行する。

19. プラットフォーム管理者の運用

19.1 利用者管理

プラットフォーム管理者は、プロジェクトを選択しない状態でプラットフォーム全体の利用者を管理できる。利用者一覧では、ユーザー名、メールアドレス、所属プロジェクト、アカウント状態を確認する。

利用者に対して実施できる主な操作は次のとおりである。

- 一般ユーザーの凍結、ロック解除、復旧および削除
- 一般ユーザーのMFA情報のリセット
- 対象ユーザーの全セッションを強制終了
- 全一般ユーザーのセッションを一括強制終了
- プロジェクト管理者権限の付与または解除

プラットフォーム管理者自身や、他のプラットフォーム管理者を通常の利用者操作で凍結、削除または強制ログアウトしてはならない。ユーザーを削除する前に、唯一のプロジェクト管理者ではないこと、必要なデータが出力済みであること、所有権が移管済みであることを確認する。

19.2 監査とセキュリティ記録

次の事象は監査対象として記録する。

- ログイン成功、ログイン失敗、MFA失敗
- メール確認、パスワード再設定、MFA設定・変更・管理者リセット
- 権限変更、メンバー招待・削除、所有権変更
- アカウント凍結、ロック解除、削除
- 個別および一括の強制ログアウト
- 重要なプロジェクト設定、取込、削除、復元および永久削除

監査記録は定期的に確認し、必要に応じてCSVとして出力する。不審な連続失敗、通常と異なる時間帯、権限の急な変更、短時間の大量登録や大量タスクを確認した場合は、対象アカウントを凍結し、セッションを無効化して調査する。

19.3 監視、メールおよびバックアップ

管理画面と運用監視では、Web、Worker、PostgreSQL、CPU、メモリ、ディスク、タスクキュー、メール送信結果を確認する。メール送信失敗、タスク滞留、ディスク閾値超過、バックアップ失敗、復元失敗は運用担当者へ通知する。

定期バックアップでは、少なくともPostgreSQL、プラットフォームデータ、プロジェクトデータおよび設定を対象とする。バックアップの成否だけでなく、保持期限、暗号化、保存先、復元手順および復元演習の結果を運用記録へ残す。具体的なコマンド、スケジュール、Entraアプリ登録、Graph権限および秘密情報の管理は、別冊のインフラ運用手順に従う。

付録A. プロジェクト導入テンプレート

```
projectKey: <unique-project-key>
projectName: <display-name>
repository:
  url: <github-repository-url>
  defaultBranch: <branch-name>
modules:
  - id: <lowerCamelCase>
    displayName: <localized-name>
    category: <business|system|monitoring|shared>
    owner: <team-or-role>
qualityGates:
  unitTest: pass-required
  integrationTest: pass-required
  systemTest: all-required-items-pass
```

付録B. 個別プロジェクト記載例

個別プロジェクトでは、本書を変更せず、別紙にプロジェクトキー、モジュール台帳、実パス、接続先、運用責任者を記載する。これにより、製品共通仕様と導入先固有情報を混在させない。