



mabl新機能解説

プロンプトによるテスト生成と
ローカル/クラウド実行のシームレスな統合

2026年1月28日（水）13:00～14:00

舟木将彦（Sales Engineer, mabl）



本日のアジェンダ

-
1. 品質に対するギャップの拡大
 2. テスト自動化の現状と課題
 3. AIによるテスト生成の進化
 4. 自然言語からテストへ
 5. ローカル実行機能の活用
 6. クラウドとのシームレスな連携
 7. CLI/MCPを活用した開発ワークフロー
 8. デモ：mabl-cosmeアプリケーション
 9. ベストプラクティス



品質に対するギャップの拡大 開発スピード vs テスト

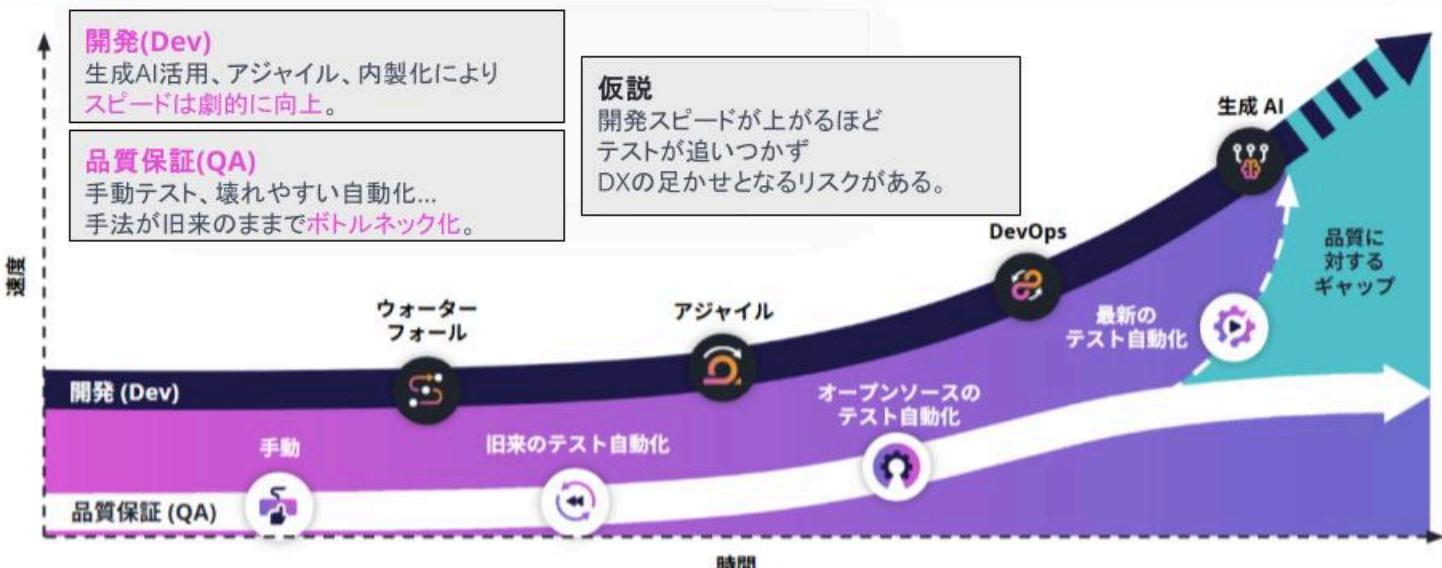
開発(Dev)

生成AI活用、アジャイル、内製化により
スピードは劇的に向上

品質保証(QA)

手動テスト、壊れやすい自動化...
手法が旧来のままでボトルネック化

品質に対するギャップの拡大





テスト自動化の現状と課題

従来のテスト作成の課題

- **時間がかかる:** UIテストの作成・メンテナンスに多大な工数
- **専門知識が必要:** セレクタ、待機処理、アサーションの知識
- **メンテナンス負荷:** UIの変更に追従するコスト
- **実行環境の分断:** ローカルとCI/CDで異なる設定

結果: テストが後回しになり、品質担保が困難に



AIによるテスト生成の進化

2026年のテスト自動化

従来	AI時代
手動でセレクタを指定	自然言語で意図を伝える
コードを書いてテスト作成	プロンプトからテスト生成
固定的なアサーション	AIによる柔軟な検証
ローカル or クラウド	シームレスな統合

キーワード: Shift Left + AI = 開発者体験の向上



自然言語からテストへ

プロンプトによるテスト生成

「ユーザーがログインして、画像をアップロードし、AI背景生成を実行して、保存するフローをテストして」

↓ mablが自動生成

- ログインフォームへの入力
- 画像アップロードの操作
- AI生成ボタンのクリック
- 保存処理の実行と検証

ポイント: 意図を伝えるだけでテストが完成



MCPによるテスト生成

Claude Code + mabl MCP

```
# テストの計画
mcp__mabl__plan_new_test

# テストの作成
mcp__mabl__create_mabl_test

# ローカルで実行
mcp__mabl__run_mabl_test_local

# クラウドで実行
mcp__mabl__run_mabl_test_cloud
```

IDE内で完結: コードを書きながらテストも作成



ローカル実行機能の活用

開発中のテスト実行

- **即时フィードバック:** コード変更後すぐにテスト実行
- **デバッグ容易:** ローカル環境でステップ実行
- **コスト効率:** クラウド実行前にローカルで検証
- **オフライン対応:** ネットワーク不要で基本テスト

```
# mabl CLIでローカル実行  
mabl tests run --id <test-id> --environment <env-id>
```



クラウドとのシームレスな連携

ローカル → クラウドの統合

開発環境（ローカル）

- ↓ テスト作成・デバッグ
- ↓ ローカルで動作確認

↓

クラウド実行

- ↓ 複数ブラウザ・デバイス
- ↓ 並列実行
- ↓ スケジュール実行

↓

CI/CDパイプライン統合

同じテストをローカルでもクラウドでも実行可能



CLI/MCPを活用した開発ワークフロー

実装とテストの並走

ステップ	開発者の作業	テスト作業
1	機能の実装	MCPでテスト計画
2	コードレビュー	テスト生成・ローカル実行
3	マージ	クラウド実行・CI統合
4	デプロイ	本番監視

ポイント: テストが開発のボトルネックにならない



デモ：mabl-cosme

生成AI系SaaSを模したデモアプリ

- **JWT認証**: サーバーサイド認証フロー
- **AI機能**: DALL-E 3による背景生成
- **data-testid**: 安定したセレクタ設計
- **多言語対応**: 日本語/英語/中国語

URL: <https://mabl-cosme-xxxxxx.run.app>



デモ：テスト生成フロー

ライブデモ

1. **プロンプト入力:** 自然言語でテスト内容を記述
2. **テスト生成:** mablがステップを自動生成
3. **ローカル実行:** 開発環境で動作確認
4. **調整・修正:** 必要に応じてステップを編集
5. **クラウド実行:** 複数ブラウザでの検証



ベストプラクティス

実装とテスト作成を並走させるために

1. **data-testidを先に設計**: UI実装前にセレクタを決める
2. **小さなテストから始める**: ログイン→基本操作→複雑なフロー
3. **ローカルで頻繁に実行**: 変更のたびにテストを回す
4. **AIアサーションを活用**: 柔軟な検証で脆いテストを回避
5. **CI/CDに組み込む**: PRごとにテストを自動実行



テスト戦略の分担

mabl と Playwright の使い分け

テスト種別	推奨ツール	理由
E2Eフロー	mabl	実際のAPI・認証を検証
APIテスト	mabl	JWTフローの統合テスト
UIバリデーション	Playwright	高速・モック可能
多言語確認	Playwright	Context差し替えで効率的
入力エラー	Playwright	クライアント側ロジック



まとめ

本日のポイント

- **プロンプトでテスト生成:** 自然言語からテストを自動作成
- **ローカル実行:** 開発中の即時フィードバック
- **クラウド統合:** シームレスなCI/CD連携
- **MCP活用:** IDE内でテスト作成から実行まで完結
- **並走開発:** 実装とテストを同時に進める新しいワークフロー

次世代の「Shift Left」で開発者体験を向上



参考リソース

ドキュメント・サポート

リソース	URL
mabl ドキュメント	https://help.mabl.com/
mabl CLI	https://help.mabl.com/docs/mabl-cli
MCP Server	npmjs.com/@anthropics/mcp-server-mabl
mabl-cosme (デモ)	github.com/mfunaki/mabl-cosme

サポート: support@mabl.com

ご清聴ありがとうございました！