消費者駆動契約テストパターンとその課題

中川 尊雄[1,a)]　宗像 一樹[1,b)]

概要：Web API やメッセージングによるサービス間連携を伴うソフトウェア開発手法やアーキテクチャが提唱・実践されている．各サービスごとの独立性を高めることによって，開発の高速化や品質の向上が期待される一方，サービス間の結合部分の品質を担保しづらくなるという課題がある．実際の結合を伴わず，開発の独立性を保ったままサービス間が成功裏に連携することを保証する手段として，消費者駆動契約（CDC）テストパターンが提唱されている．しかし，CDC テストパターンを実際に適用する際の課題や解決策についての研究は少なく，議論が不足している．本稿では議論のため，不要な契約が生成される可能性や，適用範囲が限定されるなど，CDC テストパターンの適用上の課題を述べる．

Barriers on the Application of Consumer-Driven Contract Testing

1. はじめに

今日のソフトウェア開発においては，異なる機能やデータを提供する Web サービス同士を，Web API やメッセージングによって連携させる場合がある．昨今採用の進むマイクロサービスアーキテクチャでは，適切な粒度に分割したサービス群を疎に結合することを基礎としている [3]．このような設計は、各サービスの内部実装の自由や，独立したタイミングでの修正・改良が可能といった利点をもたらす．

一方で，サービス間を疎結合に構成する場合，結合部分の品質に関する新たな課題があるとも指摘されている．Wittern らは Web API 利用に関する課題として，版変更への追随の困難さ，実行時まで誤りに気付かないことなどを挙げた [6]．Savchenko らは，個々のサービス内の結合テストが容易になる一方，サービス間の結合に関するテスト方法を考える必要があると述べた [5]．

サービス間が成功裏に連携することを保証する手法として，消費者駆動契約（Consumer-Driven Contract; CDC）テストパターンの有効性を唱える文献がある [3][1]．

CDC テストパターンでは，「API 提供側（以降，提供側）の振舞い」と「API 利用側（以降，利用側）の期待する振舞い（期待動作）」が一致することをテストする．仮に提供側が利用側の期待に反した変更を行っても，テスト失敗によって運用環境へのデプロイが抑制され，結合時の不具合を防ぐことができる．

ただし，CDC テストパターン，およびそれを実装したツールには，契約の（提供側にとっての）厳格さや，期待動作の列挙が難しいといった理由から生じる適用上の課題が存在する．また，現時点では CDC テストパターンの適用や課題を取り上げた研究は少なく，適用上の課題が十分に洗い出されていない状態にある．

そこで本稿では，CDC テストパターンおよびその実装例を紹介し，実世界で指摘されている課題，および著者の考える課題を述べる．ワークショップにおいては，未発見の課題や課題の解決策について議論したい．

2. CDC テストパターン

2.1 概要

CDC テストパターンは，サービス間結合の整合性やテストに関する次のような課題を解決するために提案・実践されているテスト方法・パターンである [4]．

運用環境での思わぬ破壊的変更：運用環境での提供側の版変更によって，利用側との結合に不整合が生じる場合がある．

テスト結果に対する責任の所在が不明確：実際の結合を伴うテストにおいて，テスト結果が利用側だけでなく，提供側の動作状況にも左右されてしまう．

ユースケースが不明確である：提供側にとって，利用側の

[1] 株式会社富士通研究所
FUJITSU LABORATORIES LTD.
[a)] nakagawa-takao@jp.fujitsu.com
[b)] munakata.kazuki@jp.fujitsu.com

ユースケースが具体的に把握しづらいため，API 部分の使われ方についての網羅的なテストが難しい場合がある．

すなわち，運用環境上で提供側と利用側が成功裏に結合することを，実際の結合を伴わず，利用側のユースケースに沿って保証する必要がある．CDC テストパターンは，これを以下の手順で実現する．

(1) 利用側は，自身のプログラムのテスト時に，依存する提供側 API のモックプログラムを用いる．
(2) テスト中の，モックプログラムとモックサーバの間でやり取りを，要求・応答ペアの形式で記録する．
(3) 利用側は，テストがすべて成功した場合，要求・応答ペアの一覧を契約として提供側に送付する．
(4) 提供側は，自身のプログラムのテスト時に，利用側から送付された要求・応答ペアを用い，自らのプログラムが利用側の期待する動作を行うかをテストする．
(5) 期待動作と異なる動作をした場合，CI/CD ツールがプログラムの運用環境へのデプロイを差し止める．

2.2 実装例

CDC テストパターンを実現するためのツールは複数実装されており，特に著名なものとして，Pact[*1] と Spring Cloud Contract[*2] が挙げられる．

これらのツールは CDC テストパターンの機械的な支援を利用側・提供側に提供する。たとえば、利用側に対してはテストプログラム中にモック設定が記述できる言語拡張や，テスト時に契約ファイルを自動生成する機能を提供する．提供側に対しては，契約ファイルを集積するための契約ブローカ（アプリ）や，契約ファイルをもとに半自動でテストを実行するテストドライバを提供する．

3. CDC テストパターンの課題

CDC テストパターンの適用を難しくする課題が，開発者コミュニティ等からいくつか指摘されている．また，その他にもいくつか適用上の課題が考えられるため，併せて紹介する．

契約を破棄する場合の対処法が原始的である：必要に迫られて破壊的変更を行う場合，提供側と利用側の間で直接対話する必要がある．こうした対処法は，提供側に対する利用側の数が多い場合や，組織間のコミュニケーションが疎な環境では困難であり，適用上の課題となりうる．

過剰な・矛盾した契約が発行される可能性：利用側がテスト対象機能に影響しない値まで契約に含めてしまい，提供側インタフェースを過剰に制約する危険性がある．また，複数の利用側が居る場合，仕様の誤解などを理由に契約と契約の間で矛盾が起こる可能性がある．

CDC テストパターンを双方向に適用すればこれらの状況はある程度防げるが，現状では利用側を制約するような CDC フレームワークが提唱されていない状態にある．

提供側のモチベーションの維持：前述の通り，提供側の API に関する変更は厳格に制約され，必要な版変更があった際はその都度対話が必要となることから，開発が遅延する可能性がある．

実際の Web API を対象にしたサーベイ研究では，バグ修正や機能追加，性能の調整のために頻繁な変更が起こることが指摘されており，提供側にとって CDC テストパターンの採用を忌避する理由になりうる．

組織外に開かれた開発への適用が難しい：CDC テストパターンは，単一の組織内や顧客・提供企業の関係のように，密な連携を持つ組織での適用を前提としている．

一方で，一般利用者向けのパブリックな API や，特定多数の参加者が API を公開しあうコミュニティといった環境での品質維持も無視できない．Ioini らのように，テスト用サーバを公開してテストケースを集める手法が提案されている [2] が，提案に留まっており，実践や評価が欠けた状況である．

4. おわりに

本稿では，複数のサービスが API を呼び合うことで動作するシステムを検証する CDC テストパターンについて解説し，現時点で考えられる適用上の課題を挙げた．

CDC テストパターンは，採用を公言する企業もあり，教則本やガイドラインでも頻繁に挙げられる技術であり，課題の解決によって広い範囲への適用が見込める．未発見の課題や解決策について，ワークショップにて議論したい．

[*1] https://docs.pact.io/
[*2] https://spring.io/projects/spring-cloud-contract

参考文献

[1] Daya, S., Van Duy, N., Eati, K., Ferreira, C. M., Glozic, D., Gucer, V., Gupta, M., Joshi, S., Lampkin, V., Martins, M. et al.: *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*, IBM Redbooks (2016).
[2] El Ioini, N. and Sillitti, A.: Open web services testing, *Services (SERVICES), 2011 IEEE World Congress on*, IEEE, pp. 130–136 (2011).
[3] Newman, S.: *Building Microservices*, O'Reilly Media, Inc., 1st edition (2015).
[4] Robinson, I.: Consumer-Driven Contracts: A Service Evolution Pattern.
[5] Savchenko, D. I., Radchenko, G. I. and Taipale, O.: Microservices validation: Mjolnirr platform case study, *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 235–240 (2015).
[6] Wittern, E., Ying, A., Zheng, Y., Laredo, J. A., Dolby, J., Young, C. C. and Slominski, A. A.: Opportunities in Software Engineering Research for Web API Consumption, *Proceedings of the 1st International Workshop on API Usage and Evolution*, WAPI '17, pp. 7–10 (2017).