

投稿日 2022/12/22

システムの耐障害性を高めるためのアプローチ

はじめに

本書は、システムの耐障害性を高めるための心構えとテクニックを紹介するものです。システム開発において耐障害性を高めるには要件定義からシステムテスト、さらには運用・保守を含め一貫した対策が必要になります。本書では汎用性があり比較的効果を得られやすい耐障害性の「評価」にフォーカスしています。耐障害性を実現するための設計手法や実装手法など開発サイクルにおける網羅的なテクニックを紹介するものではありません。

本書の対象読者

本書は、主に以下の方を対象としています。

- 本番でのシステム障害に悩まされているが、自身のシステムの耐障害性のレベルが分からない
- カオスエンジニアリングに興味があるが、「本番で障害を発生させるなんてあり得ない」「ウォーターフォール開発とはミスマッチだ」など遠い存在と感じている
- オンプレミスでのノウハウはあるが、クラウドでの耐障害性評価の方法が分からない

なぜ耐障害性が重要なのか

耐障害性とは、障害が発生したときに影響を最小限に留め回復するシステムの性質のことです。昨今システムを構成するコンポーネントは非常に多くなり、すべてを把握して安定したサービスをローンチし運用を続けることの難易度が高まっています。また、システムを構成する各々のサーバやサーバ間のネットワークは決して安全とは言えません。したがって、「障害が発生すること」を前提として、耐障害性を作りこむことが重要となります。耐障害性への配慮によりシステムは複雑になりますが、我々がビジネス価値を創出するためには「システムの複雑性」に向き合わなければなりません。

◇ 理解しておかなければいけない2つのポイント

- 「障害が発生すること」を前提として影響範囲を最小限に留め回復させることを考えなければならない
- ビジネス価値を創出するために「システムの複雑性」は避けて通れず、向き合わなければならない

前提

◇ 目標とする耐障害性レベルを定義すること

耐障害性の実現はコストとのトレードオフとなるため、ビジネスが求める非機能要件に対して適切な耐障害性を実現すべく設計しなければなりません。あるいは、SLO（Service Level Objective）やSLA（Service Level Agreements）として定義したサービスレベルに対しても同様です。そのため、目標とする耐障害性レベルが定義されていることが前提となり、その定義には「非機能要求グレード」の可用性に対する記述が参考になります。

- [システム構築の上流工程強化（非機能要求グレード）：IPA 独立行政法人 情報処理推進機構](#)

◇ 耐障害性を実現できる設計・実装ができていないこと

目標とする耐障害性を実現できる設計が成されていないシステムを評価しても結果は明白で意味がありません。実装においても同様で、不具合であることが分かっている機能をテストしても、貴重な人的リソースをムダにするだけで期待する結果を得られることはありません。こういった場合は、設計や実装の見直しを優先しましょう。設計や実装を評価する前に、本番で発生することが想定される障害事象に対するシステムのふるまいにおいて、目標とする耐障害性の実現できるという仮説を立て評価に挑みます。

「設計」の耐障害性を評価する

システム開発のサイクルにおいて、問題の検出が遅くなればその対策にかかるコストは膨れ上がるため少しでも前の工程で検出したいと誰もが考えることでしょう。しかし、実際に動作するアプリケーションが存在しない設計フェーズでその評価を機械的に行うことは困難で、有識者によるレビューに頼りがちです。また、有識者のアサインは困難で、仮に有識者をアサインできたとしても、人間がチェックするがゆえにチェック漏れは避けられません。

ここでは、耐障害性の設計時に設計の妥当性を機械的に評価できる2つのテクニックを紹介します。

◇ モデル検査手法

シーケンス図や状態遷移図で表現し、人間のレビューにより正しさを評価している設計をモデル化することにより機械的かつ網羅的に検査し不備を検出できます。モデル検査手法は次のようなシステムに対する品質担保において有効な手段のひとつとなります。

- 複雑で人間がすべてを理解した上で設計やレビューをすることが困難なシステム
- 設計の正しさを証明することが困難なシステム

そして、モデル検査手法のためのツールとしてTLA+などが存在します（参考書籍：[実践TLA+（Hillel Wayne 株式会社クイープ 株式会社クイープ）](#)）。本手法は一度導入すれば効果が期待できますが、導入するにはTLA+など「設計をモデル化するための表現方法」の習得が必要です。

◇ AWS Resilience Hub

AWS上にシステムを構築する場合は[AWS Resilience Hub](#)が有効です。AWS Resilience HubはAWS上のアプリケーションの耐障害性を定義・検証・追跡する場所を提供します。ここでの耐障害性はRPO（目標復旧地点）とRLO（目標復旧レベル）を指しており、AWSに構築した環境やCloud Formationなどからこれらを予測し目標に達しているかを検証できます。算出された値はアプリケーションを含んだサービスそのものに対するものではなく、基盤の耐障害性であり、実際の実験結果ではないため設計内容の妥当性評価として参考にとすると良いでしょう。AWS Resilience Hubの利用は有償となります。料金はAWSの[公式サイト](#)でご確認ください。

耐障害性をテストする

機能面の正しさを検証するためのテストとは別に、システムが障害に見舞われたとき、設計・実装フェーズで作り込んだ耐障害性に対して想定通り動作するかを検証する必要があります。その手段として、ここでは一般的な「障害テスト」と「カオスエンジニアリング」と言われる手法について紹介します。この2つには共通点もあります。カオスエンジニアリングについては、一般的なウォーターフォール開発のプロセスにそのプラクティスを如何に適用するかという観点から紹介します。

◇ 障害テスト

障害テストでは、以下を行うことで可用性要件を満たすかを検証し問題を発見します。

- システムが正しく動作すること
- 自動または準備した手順によりシステムが回復すること

障害テストを「効果的」かつ「効率的」に実施する以下のコンテンツを公開しています。

- [障害テスト計画ガイド](#)・[障害テスト計画書サンプル](#)

障害テスト計画で検討すべきトピック、および障害テストを推進・実施する上で考慮すべきポイントを解説するものです。

- [障害テストツール](#)

カオスエンジニアリングに必要な、「障害シミュレーション」「ユーザーリクエストの送信」「オブザーバビリティ」を実現する環境をまとめて構築できます。

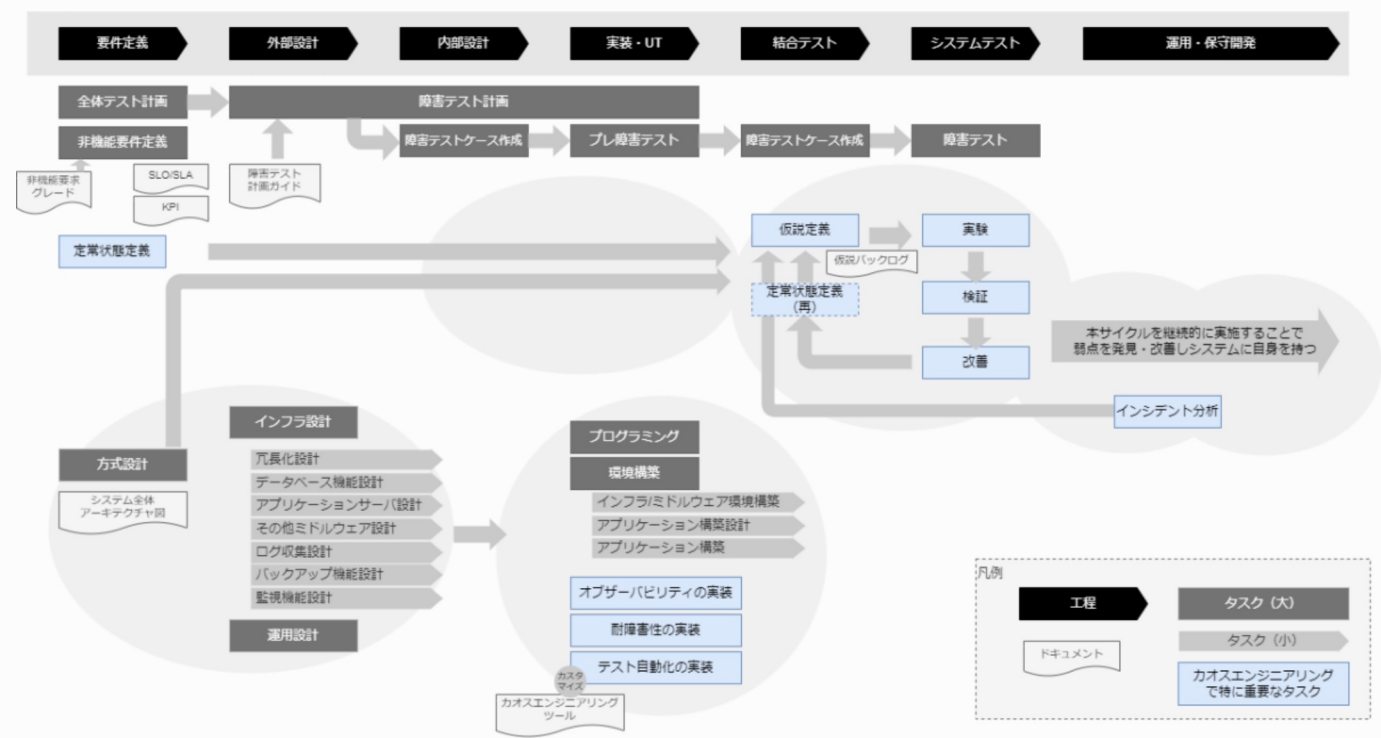
◇カオスエンジニアリング

複雑なシステムを構築するにあたりすべての異常系の事象を把握してテストケースを作成することは困難です。そこで、耐障害性をテストする手法としてカオスエンジニアリングが有効です。カオスエンジニアリングでは、仮説を立て障害をシミュレーションすることで反証を試みる一連の流れを「実験」と呼びます。既知の事象を評価する「テスト」と対比して、未知の事象を発見するため障害をシミュレーションするというニュアンスから「実験」と呼ばれているのです。ウォーターフォールの開発においてシステムテストフェーズに行う障害テストで、カオスエンジニアリングのプラクティスである「仮説定義」「実験」「検証」「改善」のサイクルを回まわすと良いでしょう。新しいアーキテクチャを採用する場合は、より早い段階で本サイクルを回すことでシステムのふるまいを知り改善すべきです。また、障害時のふるまいは外部環境も含めたシステムが動作する環境に強く依存するため、ローンチ前の本番環境や本番と同等の構成のステージング環境でテストを行わなければなりません。

さらに、実験はサービスローンチ時の品質確保だけではなく、ローンチ後の環境変化に対して継続的に実施する必要があります。自分たちが提供するサービスに自信を持ち続けるためです。ただし、本番環境への障害シミュレーションは、テスト環境での十分な訓練をクリアしカオスエンジニアリングに習熟してからにしましょう。

次に、ウォーターフォール開発にカオスエンジニアリングのプラクティスを適用する際のイメージと主要なトピックの解説を記載します。

ウォーターフォール開発におけるカオスエンジニアリングのプラクティス適用イメージ



カオスエンジニアリングの主要なトピック

カオスエンジニアリングについては、「[カオスエンジニアリングの原則 – Principles of chaos engineering](#)」を参照してください。以下にウォーターフォール開発へ適用する場合の主要なトピックについて解説します。


ステップ	説明
定常状態定義	システムの正常なふるまいを示す測定可能な出力を「定常状態」として定義する。
仮説定義	定義した「定常状態」が、障害をシミュレーションする環境とない環境のいずれにおいても継続すると仮説を立てる。
実験	サーバのクラッシュやネットワーク切断など実際に起こり得る障害をシミュレーションする。 既知のシステム特性を評価するテストに対して、新しい特性を生み出すものとして「実験」と表現します。
仮説反証	障害をシミュレーションする環境としない環境の間に発生する定常状態に関する差異を探し、仮説の反証を試みる。
改善	反証される挙動を検出した場合はシステムの改善または定常状態定義の見直しを行う。
インシデント分析	運用中のシステムでインシデントが発生した際は分析・対策を行う。

カオスエンジニアリングについてさらに深く知りたいという方はO'Reillyの書籍「[カオスエンジニアリング](#)」がオススメです。


/* Recommend */


「ソフトウェアテスト」のおすすめ記事はこちら

この記事に関連する記事もお読みください。


 ソフトウェアテスト

障害テスト計画ガイド

2022/12/22  13

 ソフトウェアテスト

カオスエンジニアリング導入の試み

2022/09/29  4

 ソフトウェアテスト

性能テスト計画ガイド

2022/04/05  21

最近投稿された記事も用意しました。

 ソフトウェアテスト

TIS AIChatLab：MagicPodをつかった自動テスト導入戦略

2025/02/26  8

 ソフトウェアテスト

DialogPlay：AIテスト自動化プラットフォーム『MagicPod』の活用

2024/10/30  22

 ソフトウェアテスト

Sales Driveの安定リリースを支える自動テスト：MagicPodの活用状況

2023/12/22  14

「ソフトウェアテスト」で最も読まれている記事を以下にまとめています。


 ソフトウェアテスト

全体テスト計画ガイド

2018/10/01  35

 ソフトウェアテスト

テスト種別&テスト観点カタログ

2018/10/01  19

 ソフトウェアテスト

性能テスト計画ガイド

2022/04/05  21