JaSST'23 Tokyo 基調講演

Chaos Engineering to Continuous Verification

Casey Rosenthal さん
(Verica)

Netflixの例

Chaos Monkey

↓ 予告なくランダムにインスタンスを落とす (週1, 月1 etc...)

対処することにより対応力を上げる!!

Netflixのシステム

他にも……

- Game day (システム障害にどう対応するか考える日)でステージングを落としたら本番も落ちた
  ↳ロギングでステージングのKafkaクラスタ作ってた
- オンラインサービスで、初めから404エラーが返ってくることを想定しておらず、マイクロサービスが止まった
- 何か月も前に入れたオブザーバビリティツールでメモリリークが発生していたが、普段はリブートをよくしていたので気づかず、コードフリーズしてからサーバが止まるようになってから気づいた

全て予想することなんてできない!! →

All components could be 100% correct, and yet the system exhibits undesirable behavior.

VOID

The Verica Open Incident Database

- 1万件以上のインシデントデータを公開
- 特にアベイラビリティのインシデントが多い。セキュリティ系も。
- 2022年のレポートが昨年12月に出た。

見えてきた 7つの神話(誤解)

1 ✗ 事件を起こすんは取り除けばいい!

腐ったリンゴを除く!

事件を起こすん=悪いん、優秀ではないん

できるん: スキルがある → 高リスクの作業多い → さらにスキルUP↑ + 事件になることも多くなる

そうでもないん・経験浅いん: スキルそんなにない → 高リスクの作業担当 → 大事件!!

2 ✗ ベストプラクティス・Run bookを作ればうまくいく!

- 作るんは必ずしもアベイラビリティに明るくない
- その時にしか起きない事象もある

作ったものを見るだけより、原因調べて対応する方に時間を使うべき

3 ✗ 根本原因を直していけば問題が減る!

RCA (Root Cause Analysis)

Root Cause の例

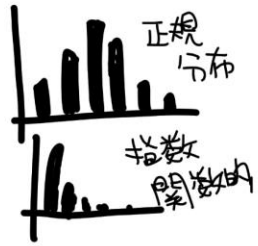
犯人(仮): 設定ファイルに1行ミスってた・誰がミスったかもわかった
↑ 設定ミスってもカバーできるパーサにすべきだった?
↑ 対策に予算をつけなかったCTOが悪い?
↑ ……

もっと根本の原因があるかも

犯人探しするだけではダメ

4 ✕ システムを継続的に測ればいい！

例）MTTR（平均復旧時間）



- 役に立てるのは正規分布のときだけ
  でもだいたいは指数関数的
- MTTR下げれば良いよね！
  ↳そうとも限りない
  ユーザメリット的には悪くなることも。

わからないことも多い。定数的には測れないこともある。

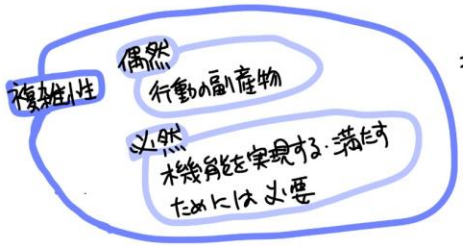
5 ✕ リスクを回避すればいい！

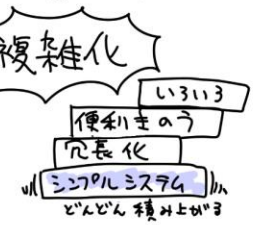
チームが育たなくなる

- リスクの裏になる経験も一緒に回避
- 問題の対処方法がわからない
  経験できない

6 ✕ システムを簡素化すれば安全になる！

安全性、アベイラビリティ、信頼性を上げる → 複雑化





複雑にしたいわけじゃない…

システム簡素化はムリ

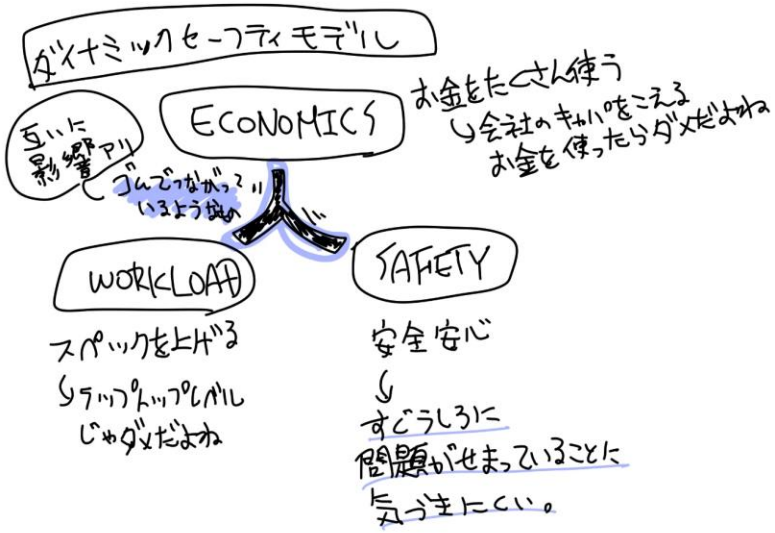
7 ✕ 冗長性を付加すればいい！

↳これが壊滅的な問題になることもある

（例）チャレンジャー号の爆発事故
↳原因はOリング
冗長性をもたせるために2本入れていたが…数増やすだけではダメだった

じゃあどうすればいい？？

↳ テストが重要



カオスエンジニアリング = 安全性のマージンがどれくらいあるかを実験して教えるため

Economic Pillars of Complexity（4つの柱）

状態 STATES	関係 RELATIONSHIPS	環境 ENVIRONMENT	可逆性 REVERSIBILITY
コントロールできない	限定・制約できない	コントロールできない	コントロールできる！
勝手にステート増える 増加を阻止できない	どんどんふえる デカクなりがち リモート マイクロサービス	個々の力だけでどうにかできるものではない…	ソフトウェアエンジニアリングから何かあったと迅に展せる 設計の意思決定細分化 ＝ 可逆性UP↗↗ (例)ウォーターフォール→アジャイル CI/CD… フィードバックループ →エンジニア教育にも！

複雑なシステムの運用進化

CI Continuous Integration "継続的インテグレーション"
- ソースコントロール → コードを継続的に

CD Continuous Delivery "継続的デリバリー"
- コードを作ってユーザが見えるところに置く
- feature はやく出せる!!
- DevOps、アジャイル とできるようになってきた

CV Continuous Verification
- システムがどのように継続的に機能しているか、ビジネスがどのように確認するか？

"継続的な検証"
- テスト … 想定できる範囲しかできない
- 実験 … 新しい気づき・想定できないことに気づく

なぜ"実験"が大事か？

カオスエンジニアリングで問題が起こる前に防ぐ"

Don't fight complexity. Navigate it.

参考

- ChAP (Chaos Automated Platform)
- VOID thevoid.community
- verica.io
- オライリー『Chaos Engineering』← 著作!!
- CHAOS COMMUNITY BROADCAST chaos.community