

1章: 障害はどのように起こるのか

ANDREAS ZELLER 著 中田秀基 監訳

デバッグの理論と実践

-なぜプログラムはうまく動かないのか-

学籍番号 60206299

氏名 吉富 楓雅

目次

1. プログラムがうまく動かない！
2. 欠陥から失敗へ
3. 時間と空間の迷路
4. 障害から修正へ
5. 自動デバッグテクニック
6. まとめ

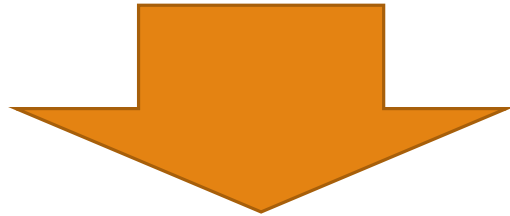
第1章は今後の
章の紹介です

プログラムがうまく動かない！



トラブル発生！

急にプログラムが動かなくなった...
今までは正常に動作していた



避けようのないもの、つまり**バグ**

バグはプログラムの一部



欠陥の発見

バグは書かれたプログラムの一部である.

どんなバグでも, 発端はプログラムを作り出した人間である.

本章では:

欠陥はどのように失敗を引き起こすのか?

どうしたら直すことができるのか?

「バグ」という曖昧な表現

一言でバグといっても様々なものがある

- 不正なプログラムコード: **欠陥**
(この行には**バグ**がある！)
- 不正なプログラム状態: **感染**
(このNULLになっているポインタは**バグ**だ！)
- 不正なプログラム実行: **障害**
(プログラムがクラッシュする. これは**バグ**だ！)



欠陥から失敗へ



失敗はどのように起こるのか

プログラマが欠陥を作る

欠陥は感染する

感染の伝播

感染は障害を引き起こす

感染の仕方

正常な状態

欠陥コードがまぎれる

プログラムの一部が感染

さらに欠陥コードがまぎれる

様々な箇所が感染

テストの呪い

- ・すべての欠陥が感染につながるわけではなく、またすべての感染が障害につながるわけではない。
- ・障害が発生していないからといって、欠陥がまったくないというわけではない。

Dijkstraはこれを**テストの呪い**(curse of testing)と表現

**テストにできるのは欠陥の存在を明らかにすることだけで
欠陥が存在することを証明できない**

感染の連鎖

感染の発端となっているのは**欠陥**

欠陥から障害へつながる因果関係は、感染の連鎖(infection chain)と呼ばれる。

デバッグに必要なこと:

感染の連鎖を発見して、根本原因(欠陥)を特定し、障害が二度と発生しないように取り除くこと。

ここからは以降の
章の紹介です

時間と空間の迷路



pixta.jp - 22334618

TRAFFIC

デバッグにおける7つの手順

1. 問題をデータベースに記録する(**T**RACK)
2. 障害を再現する(**R**EPRODUCE)
3. テストケースの自動化と単純化を行う(**A**UTOMATE)
4. 感染源の疑いがある箇所を見つける(**F**IND)
5. 感染源の疑いが最も高い箇所に着目する(**F**OCUS)
6. 感染の連鎖を分離する(**I**SOLATE)
7. 欠陥を修正する(**C**ORRECT)

なぜ障害を理解しにくいのか

欠陥を見つけるためには、正常な状態から、感染している状態に遷移する箇所を特定する必要がある。

空間探索 + 時間探索

たとえ非常に小さなプログラムでも、膨大な工数がかかる

空間と時間の迷路に迷い込んでしまう...

探索の原理

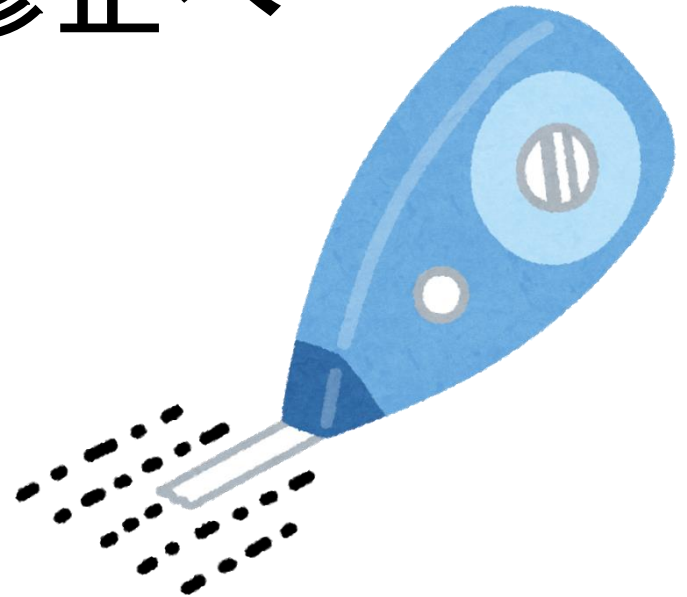
1. 感染状態と正常状態の分離

状態が感染しているのなら、その状態は、欠陥から障害まで伝播しているのかもしれない。

2. 関係するものと無関係のものの分離

ある変数の値は、それに先行するごく一部の変数の値からしか影響を受けない。

障害から修正へ



TRAFFICの手順を参考に

- 問題の記録
- 障害の再現
- テストケースの自動化と単純化
- 感染の起源の検出
- 最も感染起源の可能性が高い箇所に着目する
- 感染の起源を特定する
- 欠陥の修正

自動デバッグテクニック



自動デバッグテクニック

今後の章の紹介

- ・入力の単純化
- ・プログラムスライス
- ・状態の観察
- ・状態の監視
- ・アサーション
- ・異常
- ・原因と結果の連鎖
- ・ミスから学ぶ

まとめ



まとめ

- ・欠陥がプログラム状態の中に感染を引き起こす
- ・感染が障害を引き起こす
- ・プログラムをデバッグするための7つの手順
(TRAFFIC)