測試可靠性

網站可靠性工程：Google的系統管理之道 第 17 章

Arrack

“

If you haven't tried it, assume it's broken

1.

Types of Software Testing

Types of Software Testing

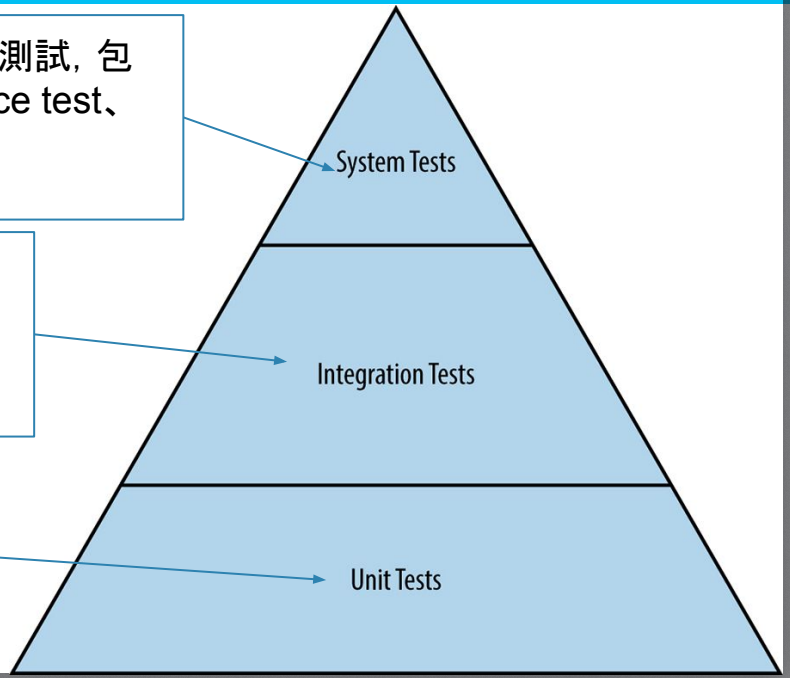
- Traditional Tests
  - Unit Test
  - Integration tests
  - System tests
- Production Tests
  - Configuration test
  - Stress test
  - Canary test

Traditional Tests

在未部署的系統上進行大型測試，包含：smoke test、performance test、regression test

把獨立的單元測試整合成較大的系統組件，用來驗證該組件的功能正確性。會 mock 掉真實依賴的部分

最小最簡單的形式，用來評估某一個軟體單元的，例如一個 class 或 function 的正確性



Production Tests

配置測試 

針對配置文件作測試，確保該服務的配置和文件一致

壓力測試 

利用壓力測試，找出 Web Service 的性能邊界

金絲雀測試 

並不真的是一個測試，而是一種結構化的用戶驗收測試，類似隨機抽查，將代碼放於真實流量下，有時候會漏掉某些 BUG。

利用 CU＝RK 來估算出 錯誤等級 U

2.

Creating a Test and Build Environment

Creating a Test and Build Environment

將測試的重點集中在用最小力氣得到最大收益的地方:

- 把原始碼分級，重要的優先(用什麼標準來衡量都可以)
- 是否有某些 function 是非常關鍵，或是對業務層面相當重要的，舉例來說:計費系統的程式通常對業務面很關鍵。
- 給其他團隊集成使用的 API

Creating a Test and Build Environment

- 一種建立強測試文化的方式：將所有遇到的問題進行測試案例化
- 建立一套良好的測試基礎設施
- 增加持續建置的系統，每次程式碼改變都進行一次建置
- 當建置系統通知異常，負責的工程師應該優先處理，原因如下：
  - 問題發生後，又有新的變動，修復會更難
  - 故障的程式會對團隊造成影響
  - 定時建置失去意義
  - 緊急發布能力受到嚴重影響

3.

Testing at Scale

Testing at Scale

- Testing Scalable Tools
- Testing Disaster
- The Need for Speed
- Pushing to Production
- Expect Testing Fail
- Integration
- Production Probes

Testing Scalable Tools

- SRE 開發工具可能負責以下的操作：
  - 從資料庫中或許並傳遞性能指標
  - 用度量指標預測未來用量，進行容量規劃
  - 重構某個用戶不可見的備份副本中的數據
  - 修改某些文件
- SRE 工具有兩個特點：
  - 這些工具的副作用基本處於被良好地測試過的主流 API 範圍內
  - 由於現存的驗證和發佈流程，這些工具基本不會對用戶造成直接的影響
- 針對危險性高的軟體設立防護邊界

Testing Disaster

- 很多災難恢復工具都設計為離線，這類的工具主要做以下的事情
  - 計算出一個可記錄狀態，一個等同於服務完全停止的狀態
  - 將核可記錄狀態推送給一個非災難驗證工具，以驗證狀態
  - 支持常見的發佈安全邊界檢 查工具，確保啟動結果是乾淨的。

The Need for Speed

- 針對某些感興趣的可能情景建立起某種假設，然後針對這些情況重複運行一定數量的測試，已獲取可靠的推斷。所以可靠且快速得到一系列可操作的假設，需要針對所有場景同時進行估算。
- 工程師關心的是他寫的代碼是否有預料之外的 race condition，從而導致這項測試更為不可靠（或是其他因素造成）

Pushing to Production

- 測試自動化的系統經常忽略生 產環境和測試環境配置不同的問題。
- 在 SRE 模型下，生 產環境和測試環境的分離，可能導致生 產環境和測試環境的不一致，這會影響想要在開發環境中重現這種不一致問題的工程師。但是至少這種分離不會導致整個研發的停滯，因為危險總不可能完全消除的。

Expect Testing Fail

- 早起軟體項目還是每年發佈一次，大部分的測試都是由人針對書面流程手工執行，這種發佈流程效率不高，但是不需要去自動化，發佈成本主要集中在文件、數據遷移、用戶培訓和其他因素上，這些發佈平均失敗週期是一年，不管寫再多測試也一樣，由於修改非常多，上次更新的可靠性數據對下次來 說沒有任何意義
- 有效的 API 和 ABI 管理工具，以及大規模的直議語言讓快速編譯執行軟體變成可能，

Expect Testing Fail

- 如果配置文件的修改比軟體更新還要更頻繁，需要注意：
  - 每個配置文件都有足 夠的測試覆蓋率，已確保可以經常修改
  - 在發佈之前，對文件的修改需要等待發佈測試完成
  - 提供一種應急機制，確保可以在測試完成之前將文件發佈，但由於應急機制會影響可靠度，通常會自動提交一份 Bug 報告，以便下次使用更好的方案處理該問題。

Integration

- 使用現成的語法（如 YMAL）和經過大量測試的分析器可以降低維護配置文件的成本
- 當一個工具行為出現異常時，工程師必須對他們的 絕大部分其他工具具有足 夠的信心，以便利用其他工具解決之前異常工具帶來的問題。
- 保障網站可靠性的關鍵因素在於找到某種可預期的異常情況，然後確保某些測試可以匯報這些問題。
- 舉例來說假設一個配置文件包含了一堆用 戶，但因為意外導致處理一半後就停止了，最近加入的用戶都沒有被處理，這機器可能還在正常運行中，負責維護用 戶目錄的工具可以很容易的發現目錄數量只有用 戶列表的一半，同時緊急回報這種不一致的情況。作用在於匯報，不應該嘗試自我修復這個錯誤（通過刪除用 戶數據）

Production Probes

- 測試機制是通過提供確定的數據檢驗系統行為是否可以接受
- 監測機制是在未知的數據輸入下確認系統行為是否可以接受
- 我們可以將任兩類作為集成測試和發佈測試，大部分測試也可以用來作為監控探針
  - 已知的錯誤請求
  - 已知的正確請求，可以針對生 產重放
  - 已知的正確請求，不可以針對生 產重放
- 生產環境探確實對服務提供了保證，提供一種清晰的反愧信號，這種信號返回的越早越好，最好這種測試也是自動化的。

4.

Conclusion

謝謝