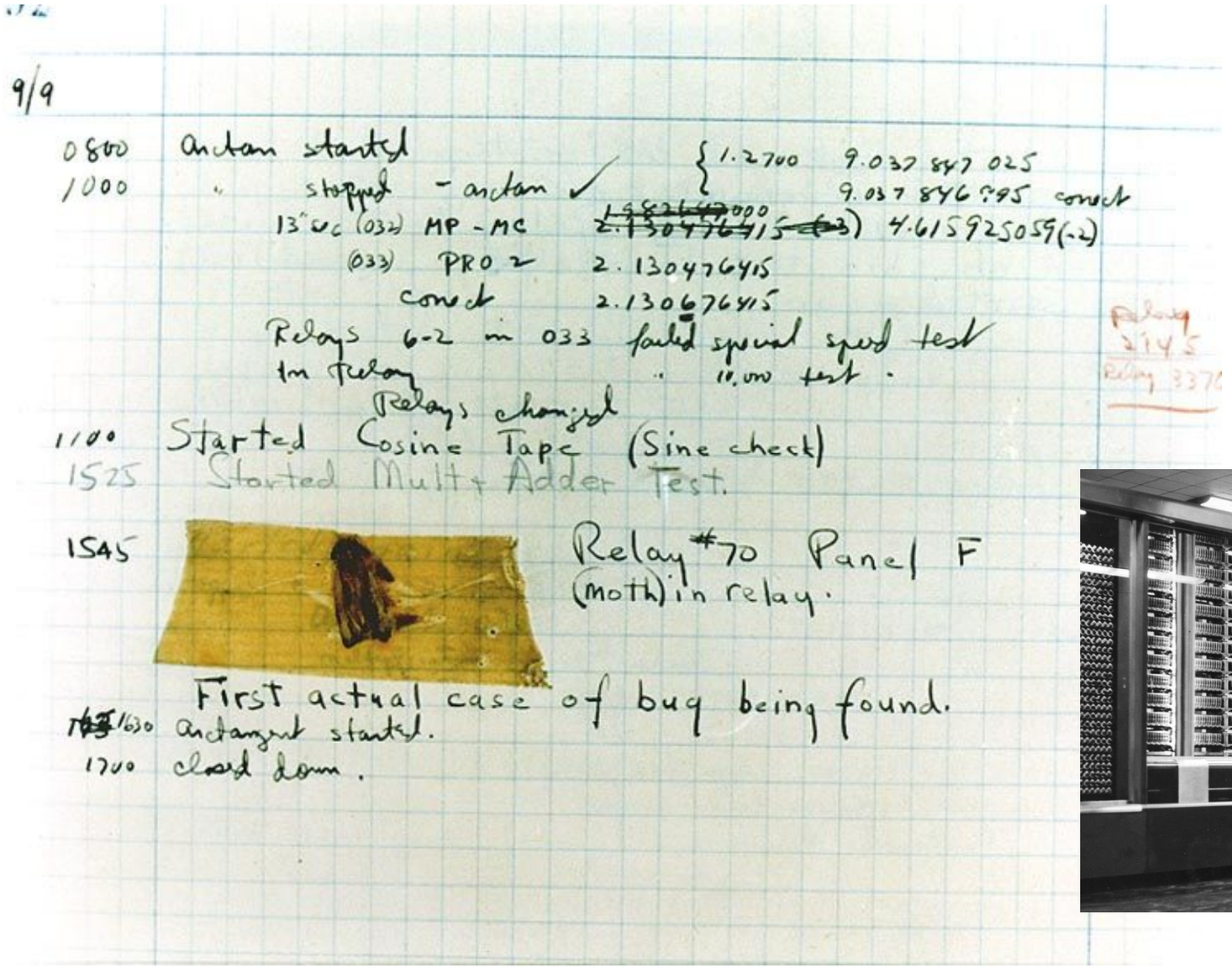
WWW.CTOMASTERMIND.IT

Test automatici

Conoscerne il valore e comunicarlo al business

Enrico Zimuel
Principal Software Engineer presso Elastic

Bug



9/9

0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 795 correct
13"sc (032) MP - MC ~~1.9821000~~ ~~2.13047641S (-3)~~ 4.615925059(-2)
(033) PRO 2 2.13047615
correct 2.13067615
Relays 6-2 in 033 failed special speed test
tm relay " 11.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.
1545 Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

1700 closed down.

Relay
2145
Relay 3370

Primo “bug” della storia, [Harvard Mark II](), 9 Settembre 1947

Alcuni bug famosi

- NASA Mars Climate Orbiter, $128 milioni persi per un errore di conversione metrico
- Pentium FDIV bug, $475 milioni persi per un errore nell'implementazione della divisione in virgola mobile
- Ariane 5 Flight 501, $370 milioni andati in fumo per l'esplosione del satellite pochi secondi dopo il decollo a causa di un overflow di 32-bit in 16-bit

Il software

“Il software è come l'entropia. É difficile da afferrare, non pesa nulla, e obbedisce alla seconda legge della termodinamica: aumenta sempre.”

Norman R. Augustine

Il software è un sistema complesso

- Il numero di possibili interazioni/flussi all'interno di un software cresce in maniera esponenziale
- Il numero variabile di utenti contemporanei di un'applicazione (es. applicazione web)
- La gestione di un team di sviluppo, con la crescita esponenziale delle comunicazioni

Misurare la complessità

- Esistono alcuni indicatori che ci consentono di stimare la complessità di un software
- Alcuni di questi sono:
  - **complessità ciclomatica**
  - **complessità NPath**

Complessità ciclomatica

- La complessità ciclomatica è **π + 1** dove **π** è il numero di istruzioni IF, FOR, WHILE, etc

```
function foo(int $a, int $b) {
    if ($a > 10) {
        echo 1;
    } else {
        echo 2;
    }
    if ($a > $b) {
        echo 3;
    } else {
        echo 4;
    }
}
```

1

2

π = 2

π + 1 = **3**

Complessità NPath

- Il numero di possibili percorsi di esecuzione all'interno di un codice (esclusi i cicli)

```php
function foo(int $a, int $b) {
    if ($a > 10) {
        echo 1;
    } else {
        echo 2;
    }
    if ($a > $b) {
        echo 3;
    } else {
        echo 4;
    }
}
```

2

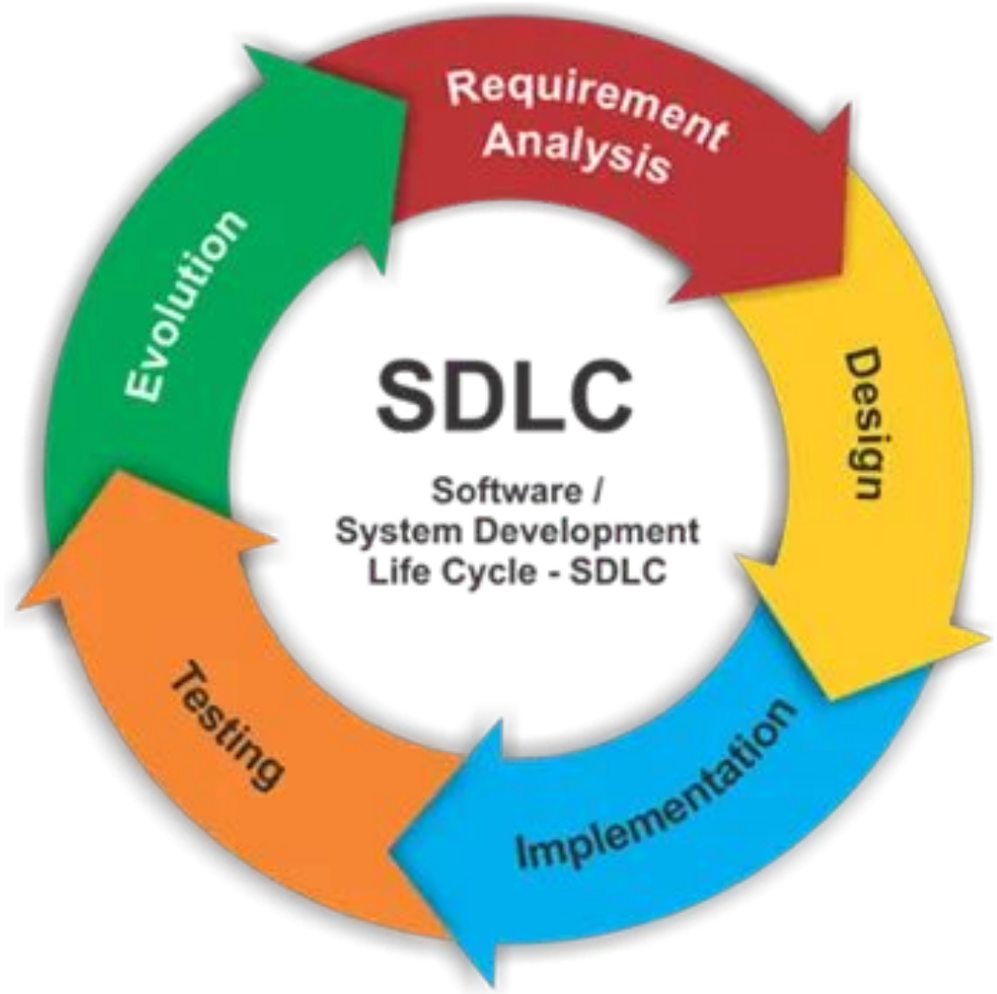
2

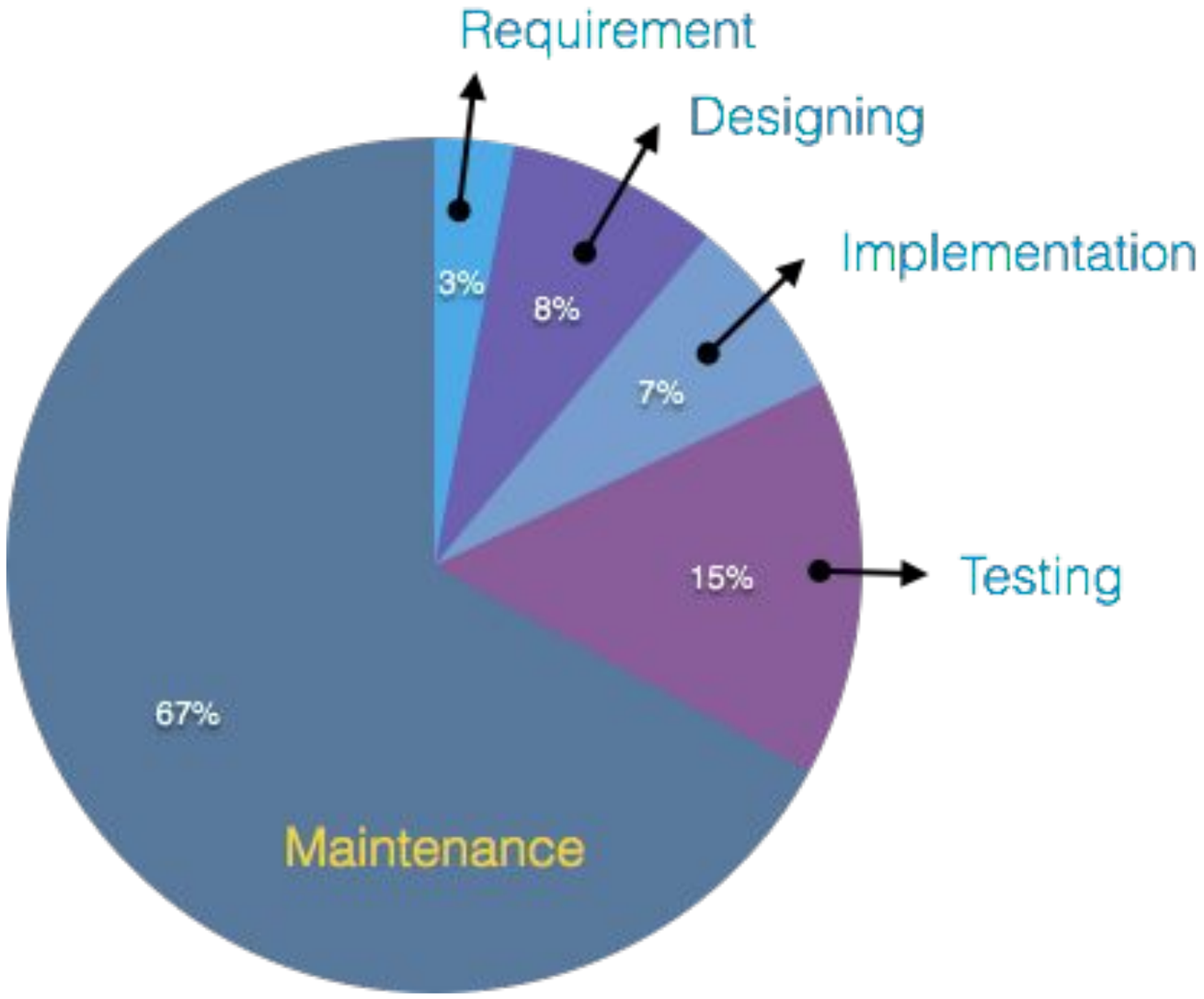
$2 * 2 = \mathbf{4}$

Combattere la complessità

- Un codice con complessità ciclomatica (o NPath) bassa è più semplice da gestire
- Esempio, consideriamo una funzione con NPath = 20. Per testare tutti i possibili flussi della funzione si dovranno scrivere 20 test
- Inoltre se la funzione accetta dei parametri in ingresso di dovranno testare i 20 flussi * numero dei possibili valori dei parametri !!!

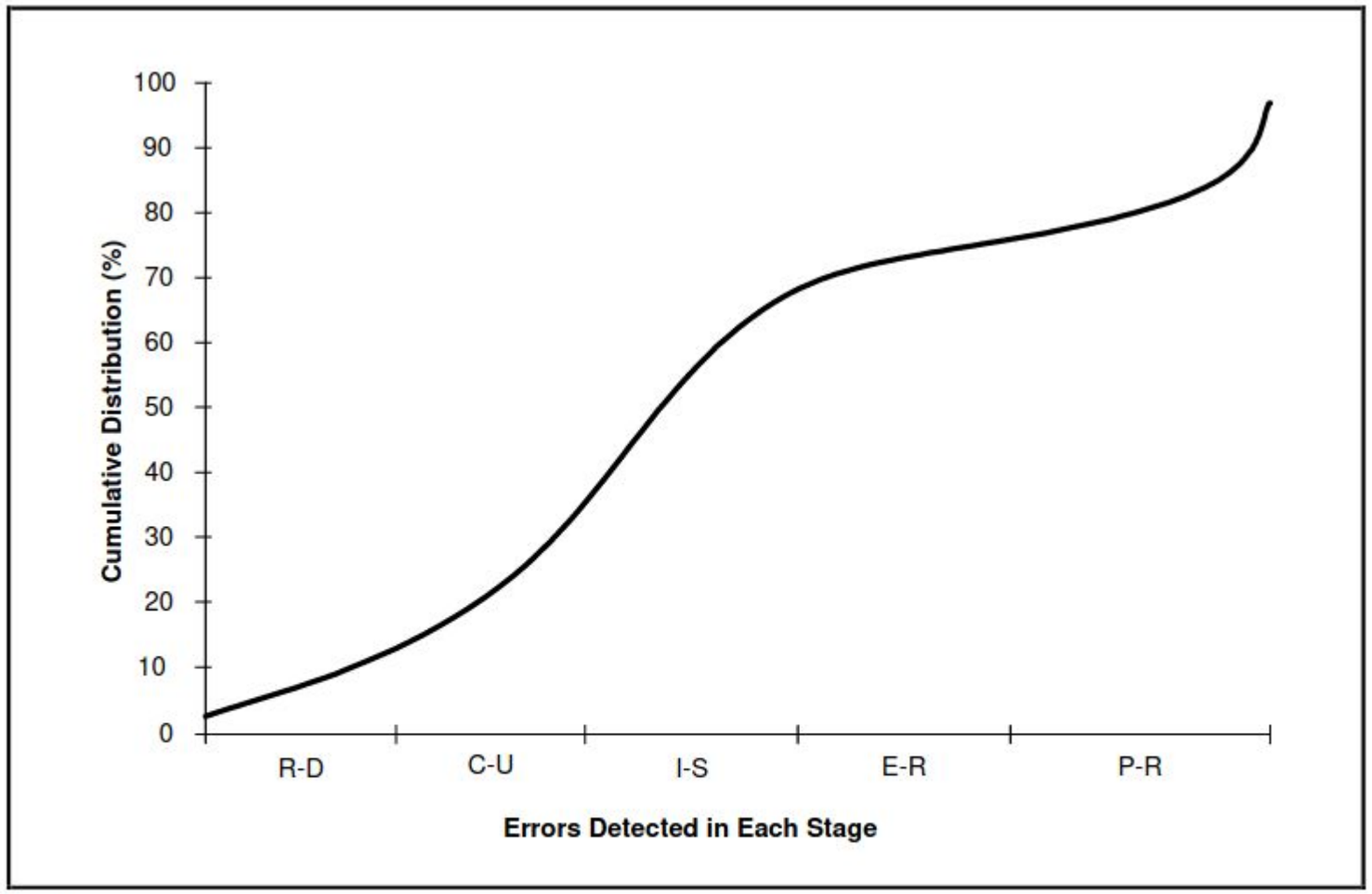
Il costo del software





Fonte: Jussi Koskinen, Software Maintenance Costs, University of Eastern Finland, 2015

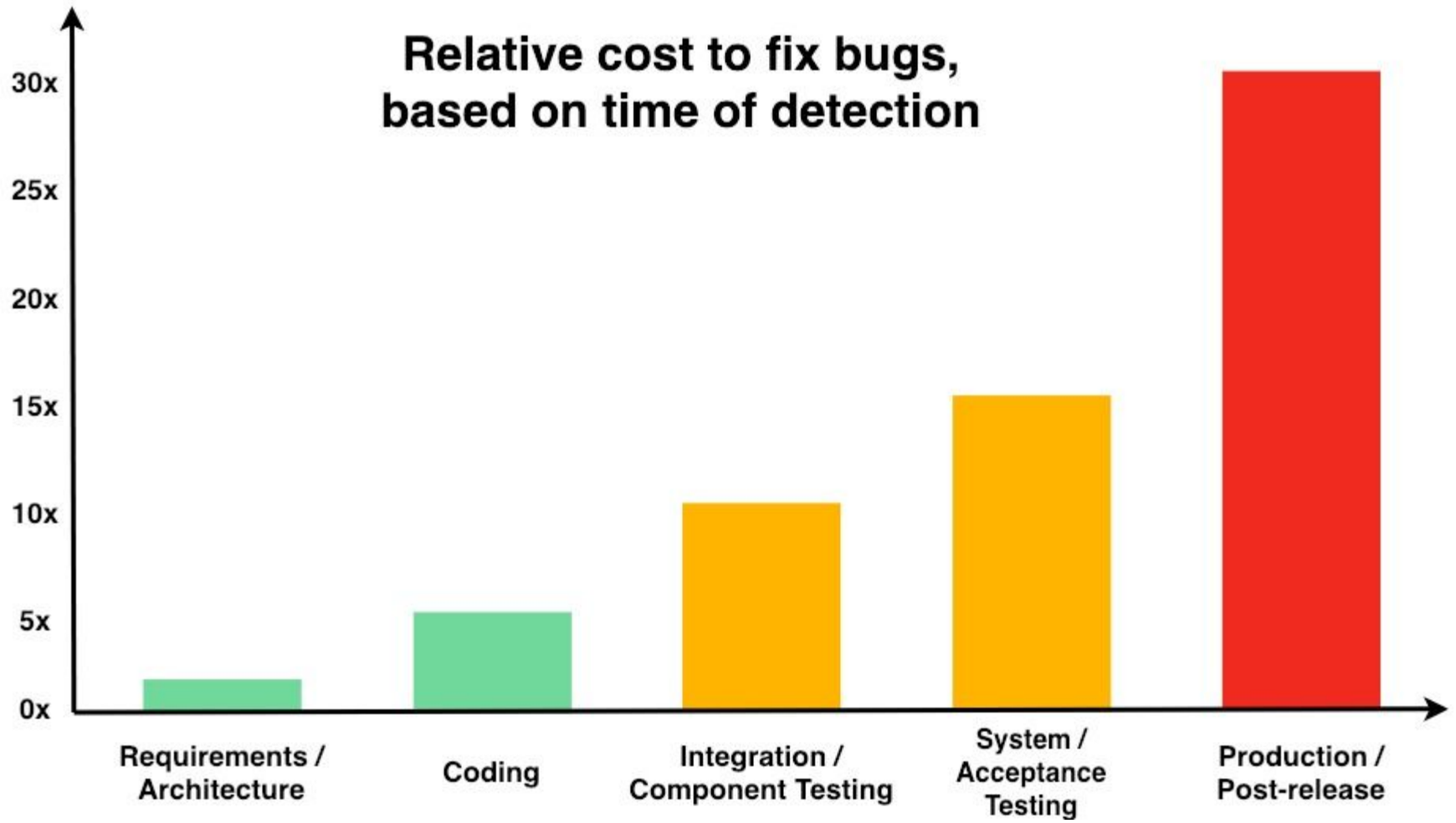
Distribuzione dei bug



Legend:
- R-D: Requirements Gathering and Analysis/Architectural Design
- C-U: Coding/Unit Test
- I-S: Integration and Component/RAISE System Test
- E-R: Early Customer Feedback/Beta Test Programs
- P-R: Post-product Release

Fonte: The Economic Impacts of Inadequate Infrastructure for Software Testing, NIST, 2002

Il costo di un bug



Fonte: Sanket, The exponential cost of fixing bugs, Deepsource, 2019

Come testate il vs. software?

CTO
MASTERMIND

Come testare il software?

- Ci sono fondamentalmente due modi per testare un software:
  - Test manuale
  - Test automatico

Test manuale

- Test del software tramite intervento umano
- Vantaggi:
  - Di facile implementazione
- Svantaggi:
  - Lento
  - Costoso
  - Soggetto ad errori
  - Noioso!

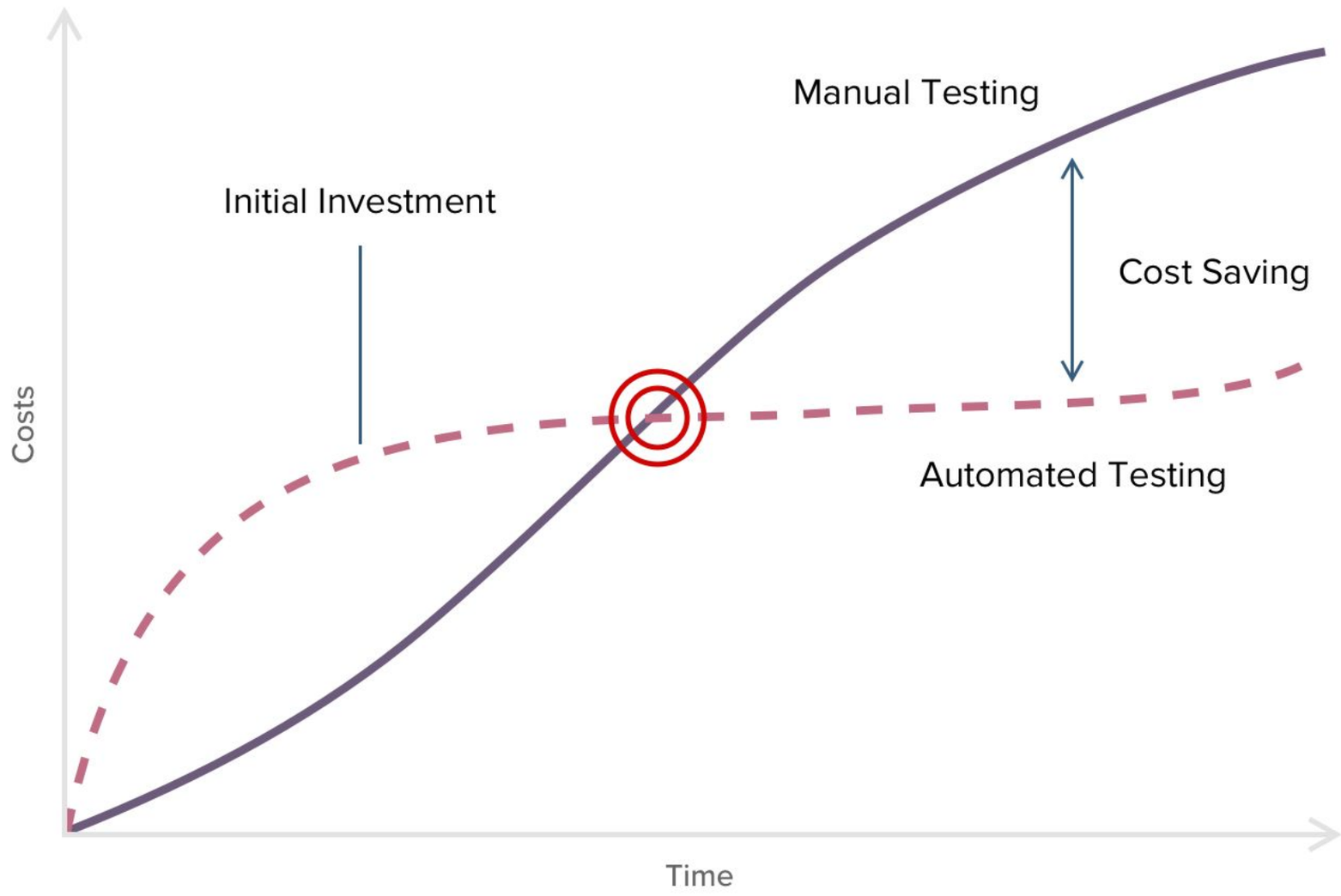
Test automatico

- Test del software tramite scrittura di un programma che utilizza il software da testare
- I test automatici sono una collezione di **asserzioni**
- Esempio: testare una funzione che implementi la somma di due numeri interi: sum(X,Y)
- Un’asserzione è una dichiarazioni del tipo se X = 5 e Y = 2 allora mi aspetto che il risultato sum(X,Y) = 7

Test automatico (2)

- Vantaggi:
  - Veloce
  - Economico (a lungo termine)
  - Completo (copertura del codice)
  - Aumenta la confidenza dello sviluppatore
  - Migliora la fase di analisi
- Svantaggi:
  - Costoso (nelle fasi iniziali)
  - Curva di apprendimento (iniziale)

Test automatici vs. manuali



Fonte: [The True ROI of Test Automation]

ROI test automatici

- Tester: $50/h junior, $75/h senior (test automatici)
- Team Manual: 10 tester junior, 135 ore lavoro/mese
- Team Automated: 7 tester junior, 3 senior (5x test/h)

Manual	Automated
Hours (10x135)= 1,350 hours	**Hours** (3x21x16x5) + (7x135) = Total of 5985 hours
Cost $78/hour	**Cost** $17.5/hour

	Coding/Unit Testing	Integration	Beta Testing	Post-Release
Hours to Fix	3.2	9.7	12.2	14.8
Cost to fix ($)	240	728	915	1,110

Un bug in produzione costa 5x rispetto al fix in fase di unit testing!

Fonte: [The True ROI of Test Automation]

Test Driven Design

- Il **Test Driven Design** (TDD) è una metodologia che ribalta le fasi sviluppo/test in test/sviluppo
- Si parte dalla definizione del test del programma per procedere poi all'implementazione del codice per fare in modo che il test passi
- Focalizza l'attenzione sulle specifiche del progetto
- Aiuta a chiarire le specifiche lavorando su dei casi concreti di utilizzo del software

Test automatici

- Esistono diversi tipi di test automatici:
  - Test unitari
  - Test di integrazione
  - Stress test (test di performance)
  - Test di sicurezza
  - Chaos test (test di resilienza)

Test unitari

- I **test unitari** sono i test che vanno a testare le funzionalità di base di singole porzioni di codice (funzioni, classi, routine, etc)
- Rappresentano le fondamenta di tutti i test
- Sono indispensabili per offrire un’impalcatura robusta sulla quale costruire ogni software

Test unitari: framework

- Tutti i linguaggi di programmazione offrono dei framework/tool per scrivere unit test
- Ad esempio:
  - Java: JUnit
  - Javascript/NodeJS: Jest, Mocha
  - C++: Catch2
  - Python: pytest
  - Go: “testing” package
  - PHP: PHPUnit

Test d’integrazione

- I **test d’integrazione** testano funzionalità di un’applicazione insieme ad altri sofware/servizi
- Ad esempio:
  - testare un’applicazione con un database
  - testare un’applicazione web su più browser (es. Selenium)
  - testare un’applicazione con una web API di terze parti (es. servizio cloud)

Stress test

- Gli **stess test** sono dei test di carico che vanno a verificare i limiti d'utilizzo di un software
- In ambito web, un test di carico viene effettuato simulano n-utenti concorrenti che utilizzano l'applicazione
- Quanti utenti contemporanei può gestire la vs. applicazione web?
- Strumenti: Apache JMeter, k6, Taurus, Locust

Test di sicurezza

- I **test di sicurezza** si concentrano su funzionalità di sicurezza di un software
- Consentono di scovare set di istruzioni a rischio di attacchi (Static Application Security Testing)
  - Utilizzo di librerie obsolete o con bug di sicurezza noti (es. Snyk)
  - Attacchi automatizzati verso applicazioni web (penetration test)

Chaos test

- I **chaos test** sono dei test che vanno a misurare la resilienza di un software inserendo delle turbolenze nel sistema
- Ad esempio, una turbolenza può essere provocata inserendo errori di comunicazione tra i servizi (ed. riavviando dei server)
- Sono stati ideati nel 2011 da Netflix
- Strumenti: Chaos Monkey, Chaos Mesh, Litmus

Riferimenti

- Arnon Axelrod, *Complete Guide to Test Automation*, Apress, 2018
- Gerald D. Everett, *The Value of Software Testing to Business: The Dead Moose on the Table,* The Magazine for Professional Tester, June 2008
- Bob Hunt, Tony Abolfotouh, *Software Test Costs and Return on Investment (ROI) Issues*, Presentation, 2014
- Jussi Koskinen, *Software Maintenance Costs*, University of Eastern Finland, 2015
- Divya Kumar, K. K. Mishra, *The Impacts of Test Automation on Software's Cost, Quality and Time to Market*, Proceeding of International Conference on Communication, Computing and Virtualization, 2016
- P. Laplante, F. Belli, J. Gao, G. Kapfhammer, K. Miller, W.E. Wong, D. Xu, *Software Test Automation*, Advances in Software Engineering, 2010
- Chris Vander Mey, *Shipping Greatness*: *Practical lessons on building and launching outstanding software, learned on the job at Google and Amazon*, O'Reilly, 2012
- Planning Report 02-3, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, NIST, May 2002
- Rudolf Ramler, Klaus Wolfmaier, *Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost*, Proceeding of International Workshop on Automation of Software Test, Shanghai, China, 2006

Grazie!

Contatti: enrico@zimuel.it

Copyright © Enrico Zimuel - https://www.zimuel.it,
CTO MASTERMIND https://www.ctomastermind.it/

Quest'opera è distribuita con Licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale.