Sviluppo software in gruppi di lavoro complessi[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Anno accademico 2022/23, II semestre

Lezione XII: Continuous integration & delivery

Continuous Integration

Configuration Management
Esplicitazione delle dipendenze
Test d'unità, d'integrazione, d'accettazione
Build automatizzate
Deployment simulato o automatizzato (domani)
$\rightsquigarrow$ *Continuous Integration & Delivery*

Continuous Integration

- Martin Fowler 2001-6 (http://www.martinfowler.com/articles/continuousIntegration.html)
- Tradizionalmente, l'integrazione è una delle parti più lunghe e rischiose dei progetti *software*
- CI $\rightsquigarrow$ integrazione continua (incrementale) per minimizzare il rischio di fallimento
  - *Reduced Deployment Risk*
  - *Believable Progress*
  - *User Feedback*

Continuous Integration

1. Lavoro su una copia locale sulla *macchina di sviluppo*
2. *Build* vs. *Compile*: oltre alla costruzione esecuzione di test
3. *build* funzionante sulla *macchina di sviluppo*
4. Caricamento sulla *macchina d'integrazione*
5. *build* funzionante sulla *macchina d'integrazione*

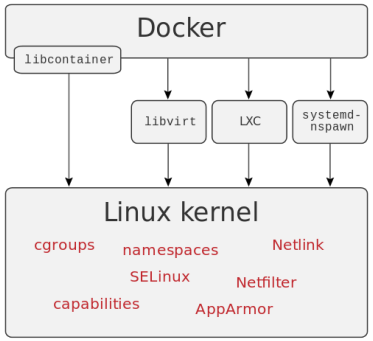
Almeno una volta al giorno.

Continuous Integration

Cambiato drasticamente negli ultimi 10 anni, grazie alla facilità di allestire **ambienti virtuali**

- Emulatori (QEmu/VMWare)
- "Meta-sistemi operativi" (*hypervisor*: Hyper-V, supporto *hardware* Intel-VT, AMD-V)
- *container*: forme di isolamento regolate dal sistema operativo che partiziona le risorse con politiche stringenti (UML, LXC, Docker)

Docker



- Sfrutta anche un *file system* a **layer**
- Le immagini docker (dati persistenti, costruite per *layer*) vengono istanziate in *container* di processi
- Le immagini si costruiscono con un `Dockerfile`

Docker

```
FROM python:3.8-alpine
LABEL maintainer="santini@di.unimi.it"
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org -r requirements.txt
EXPOSE 80
CMD ["python", "app.py"]
```

Docker

Approfondimento consigliato:

- Lezione di Massimo Santini su Docker: `https://www.youtube.com/watch?v=h_1xsPnwkOk`
- `http://docker-tutorial.surge.sh`

Continuous Integration

Il *build* non dovrebbe impiegare più di 10 minuti, altrimenti si perde il *feedback* immediato. Che fare se è necessario più tempo?

- *Deployment pipeline:* il *build* è spezzato in più fasi, *commit build*, *slower tests build*, ecc.
- Il *commit build* impiega meno di 10 min, il resto parte poi

Continuous Delivery

- Il sistema dove avviene l'**integrazione** dovrebbe essere il più possibile "simile" a quello di **produzione**
- Il *deployment* verso l'ambiente di produzione può essere automatizzato (garantendo così un maggior controllo sulla effettiva configurazione in uso!)

La suddivisione del lavoro sw

Come suddividere il lavoro, senza la continua necessità di coordinazione?
Perché un sottogruppo di lavoro possa procedere in *"isolamento"* dovrebbe conoscere i componenti sviluppati da altri (o che altri svilupperanno). Cioè il loro comportamento

- in situazioni *fisiologiche* (correttezza)
- in situazioni *patologiche* (robustezza)

A questo scopo è quindi necessario specificare il funzionamento del sistema

Correctness & robustness

IEEE *Software and Systems Engineering Vocabulary* (`http://pascal.computer.org/sev_display/index.action`):

Correctness

The degree to which a system or component is free from faults in its specification, design, and implementation.

Robustness

The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

What & How

Una specifica è una descrizione delle proprietà del marchingegno/componente utilizzato per risolvere un problema (a sua volta definito dai requisiti di progetto).
Le specifiche, perciò, sono una *descrizione* delle parti che compongono la soluzione: le modalità computazionali però sono lasciate impredicate.

What vs. How

Specifiche nel lavoro di gruppo

Le specifiche costituiscono naturalmente l'interfaccia fra gruppi che si suddividono l'implementazione di un sistema complesso.

- Il coordinamento **rimane necessario** a livello di specifica: ma accordarsi su **cosa** sembra piú facile che sul **come**;
- I sottogruppi avranno la responsabilità di **aderire alle specifiche** nelle loro implementazioni.

La suddivisione non è isolamento. . .

Perry & Evangelist (nel 1985) identificano una serie di *"Interface Fault"* che rimangono sostanzialmente comuni anche nei sistemi complessi di oggi.

- *Construction (mismatch interface/implementation).*
- *Inadequate functionality.*
- *Disagreements on functionality.*
- *Misuse of interface.*
- *Data structure alteration.*
- *Violation of data constraints.*
- *Initialization/value errors.*
- *Inadequate error processing.*
- *Inadequate postprocessing (resource deallocation).*
- *Inadequate interface support.*
- *Changes/Added functionality.*
- *Coordination of changes.*
- *Timing/performance problems.*