

Corso di Laurea in Ingegneria Informatica

Corso di Ingegneria del Software

Ingegneria del Software Agile

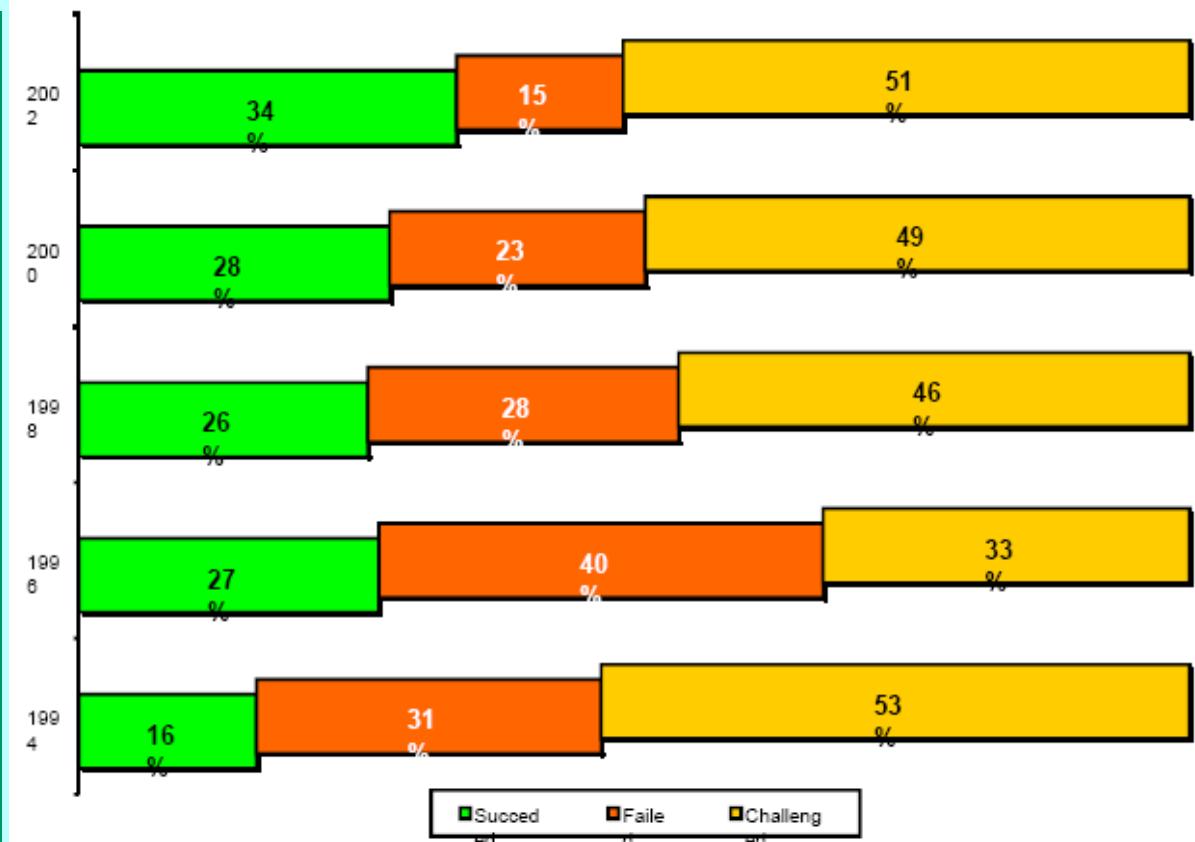
Sommario

- Metodologie agili
- eXtreme Programming (XP)
- SCRUM
- DevOps

Premessa- Molti progetti software falliscono!

CHAOS Report, Standish Group

- ◆ 66% de progetti falliti, o contestati nel 2002
- ◆ Progetti grandi falliscono più spesso dei piccoli
- ◆ Solo il 52% delle features richieste vengono prodotte



http://www.standishgroup.com/sample_research/chaos_1994_1.php

http://www.standishgroup.com/sample_research/PDFpages/chaos1998.pdf

Metodologie Agili

- ◆ L' insoddisfazione per l'eccessivo sovraccarico (*overhead*) dei tradizionali modelli di produzione del software ha portato negli anni '90 alla proposta delle metodologie agili.
- ◆ Tali metodologie:
 - Si concentrano sul codice, piuttosto che sulla progettazione;
 - Sono basati su un approccio iterativo allo sviluppo software;
 - Sono pensati per rilasciare software funzionante rapidamente, e per farlo evolvere rapidamente per soddisfare nuove esigenze.
- ◆ *Il movimento dell' Agile Software Development si "formalizza" nel febbraio 2001 in Utah*

Metodologie Agili

- ◆ Sono un sottoinsieme dei **modelli evolutivi**, nate in alternativa alle metodologie tradizionali
- ◆ Si propongono come modelli di sviluppo “**leggeri**”:
 - **Adattivi più che predittivi**
 - ◆ Non cercano di programmare lo sviluppo nel dettaglio e in modo da soddisfare tutte le specifiche, ma progettano programmi pensati per cambiare nel tempo
 - **People-oriented anziché process-oriented**
 - ◆ L'approccio prevede di adattare il processo alla natura dell'uomo... lo sviluppo software deve diventare un'attività piacevole per chi lo opera.
- ◆ Fanno largo uso di **best practices** nello sviluppo del software

Motivazione

- ◆ Numerosi progetti sono caratterizzati da una *mission* non sufficientemente chiara, i cui requisiti sono **poco chiari, instabili e variabili**
- ◆ La varietà di problemi che un'azienda si può trovare ad affrontare richiede verosimilmente strategie sempre diverse
- ◆ Talvolta le numerose prescrizioni da seguire, la quantità di documenti richiesta e una eccessiva rigidità rendono “pesante” il processo di sviluppo, aumentando il rischio di fallimento
- ◆ In molti casi è pertanto richiesto un approccio **flessibile** (dunque un modello evolutivo) e “**leggero**”

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck - Mike Beedle - Arie van Bennekum - Alistair Cockburn - Ward Cunningham - Martin Fowler - James Grenning - Jim Highsmith - Andrew Hunt - Ron Jeffries - Jon Kern - Brian Marick - Robert C. Martin - Steve Mellor - Ken Schwaber - Jeff Sutherland - Dave Thomas

Caratteristiche

- ◆ Sono metodologie pragmatiche per la **modellazione efficace** di sistemi software
- ◆ Mirano a sviluppare software in finestre temporali molto limitate (di pochi giorni o settimane)
- ◆ Prediligono la **comunicazione** in tempo reale e verbale piuttosto che la documentazione
- ◆ Mirano ad abbattere i tempi e i costi di sviluppo
- ◆ Non stabiliscono procedure dettagliate su come creare modelli, ma piuttosto suggerimenti su come essere efficaci modellatori

Caratteristiche

- ◆ Tipicamente richiedono la **collaborazione** del cliente
- ◆ Prediligono lo sviluppare programmi pensati per cambiare nel tempo piuttosto che sviluppare tutto nel dettaglio da subito (i.e., **rispondere ai cambiamenti** piuttosto che seguire rigidamente un piano)
- ◆ Includono **valori** e **principi** che incoraggiano l'agilità: una risposta ai cambiamenti rapida e flessibile

Valori

“Ottenere un'efficace **comunicazione** fra tutti gli attori del progetto, sviluppare la soluzione più **semplice** possibile che soddisfi tutti i bisogni, ottenere ogni **feedback** sempre e presto su quanto svolto, avere il **coraggio** di prendere e perseguire le decisioni, avere **l'umiltà** di ammettere che non si conosce tutto, ma che gli altri possono aggiungere valore alle proprie attività.”

Principles behind the Agile Manifesto

<http://agilemanifesto.org>

We follow these principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Practices

- ◆ Condivisione della conoscenza (***knowledge sharing***
=> “documentazione verbale”, co-locazione dei soggetti coinvolti)
- ◆ **Gestione adattativa** dei progetti
- ◆ **Feedback rapido** (cliente *on-site*, integrazione continua, iterazioni brevi, retrospettive)
- ◆ Enfasi sul ***testing***
- ◆ ***Pairing*** (si lavora a coppie)
- ◆ ***Simple design & refactoring***

Metodologie più diffuse

- ◆ eXtreme Programming (XP)
- ◆ Scrum
- ◆ Feature Driven Development
- ◆ Dynamic Systems Development Method (DSDM)
- ◆ Crystal
- ◆ DevOps

Extreme programming

- ◆ L'**eXtreme Programming**, o **XP**, è una delle prime metodologie agili
- ◆ Approccio basato su iterazioni veloci che rilasciano piccoli incrementi delle funzionalità
- ◆ Tutto il *design* è incentrato sull'iterazione attuale e non ci si concentra sulle necessità future
- ◆ Si basa sul *refactoring* ad ogni iterazione di un semplice sistema base
- ◆ Miglioramento costante e continuo del codice (verifica e adeguamento in tempi estremamente ridotti)

Extreme programming

- ◆ Partecipazione attiva del committente al team di sviluppo
- ◆ Il risultato è un processo che combina disciplina e adattività

“Comunicate con il cliente ed i vostri colleghi, usate semplicità nello sviluppo, testate il software continuamente per verificare cosa funziona e cosa no; molti dei processi “pesanti” sono basati sulla paura: siate invece coraggiosi”

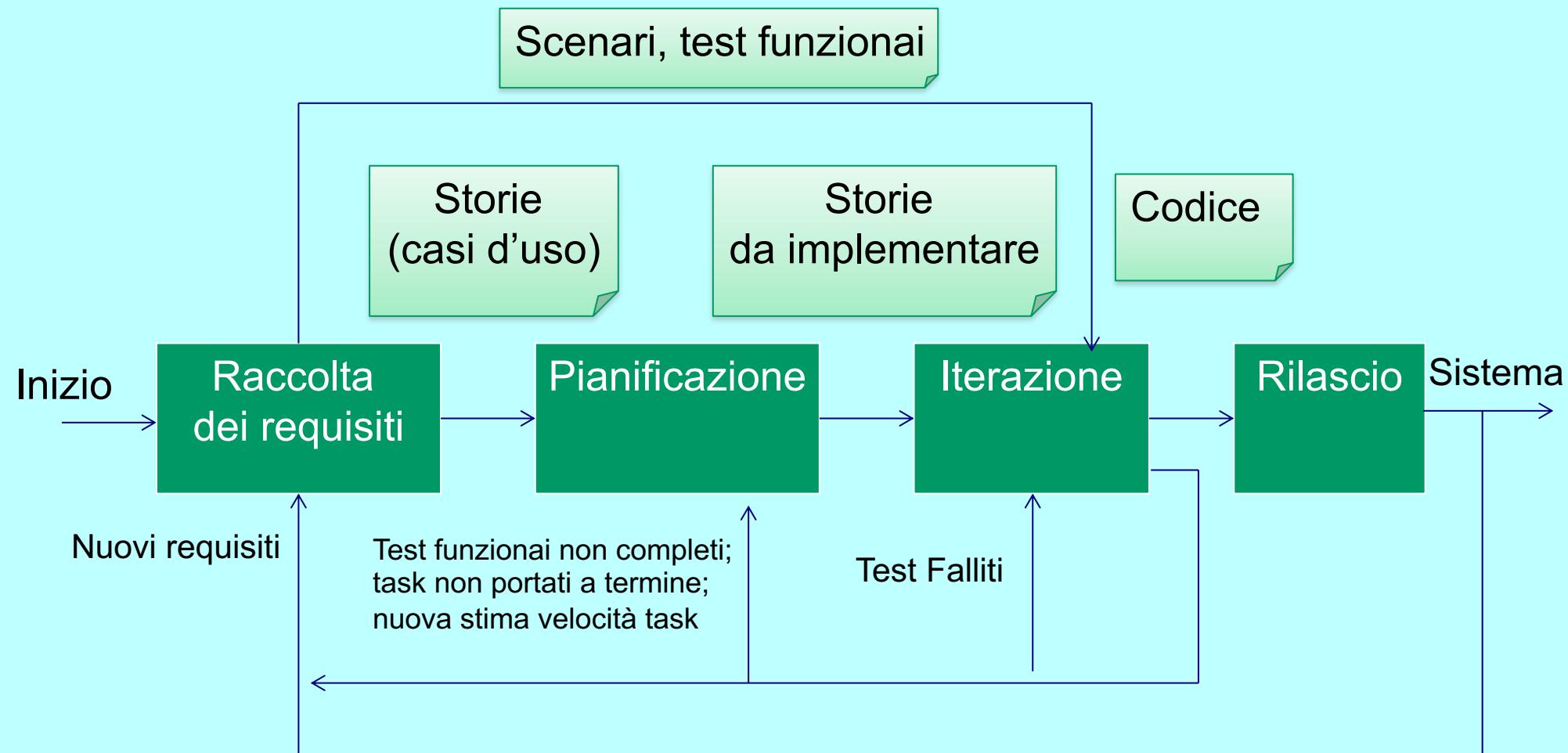
XP Practices

- ◆ **Planning game:** Determina obiettivo e tempi della prossima *release*
- ◆ **Small releases:** Rilasciare velocemente un piccolo incremento
- ◆ **Metaphor:** Guida lo sviluppo attraverso una “storia” condivisa che descrive l'intero sistema
- ◆ **Simple design:** Il sistema è concepito nel modo più semplice possibile
- ◆ **Refactoring:** Ristrutturare il sistema senza cambiarne il comportamento
- ◆ **Testing:** Da effettuare costantemente durante lo sviluppo (*Test Driven Development, TDD*)

XP Practices

- ◆ **Pair programming:** il codice è scritto da due programmatore in coppia sulla stessa macchina
- ◆ **Collective ownership:** Chiunque può cambiare qualsiasi parte del codice quando vuole
- ◆ **Continuous integration:** l'integrazione viene effettuata anche più volte al giorno
- ◆ **40-hour week:** È sconsigliato lavorare più di 40 ore la settimana
- ◆ **On-site customer:** Un rappresentante del cliente deve essere sempre a disposizione
- ◆ **Coding standards:** Supportano la comunicazione durante la produzione del codice

Processo XP



SCRUM

- ◆ Processo per migliorare la flessibilità e velocizzare lo sviluppo di nuovi prodotti
 - ◆ Il nome deriva dalla terminologia del gioco del rugby
-
- ◆ È iterativo, incrementale, per lo sviluppo e gestione di ogni tipologia di prodotto
 - ◆ Adottato a partire dagli anni '90 per gestire e controllare lo sviluppo del software



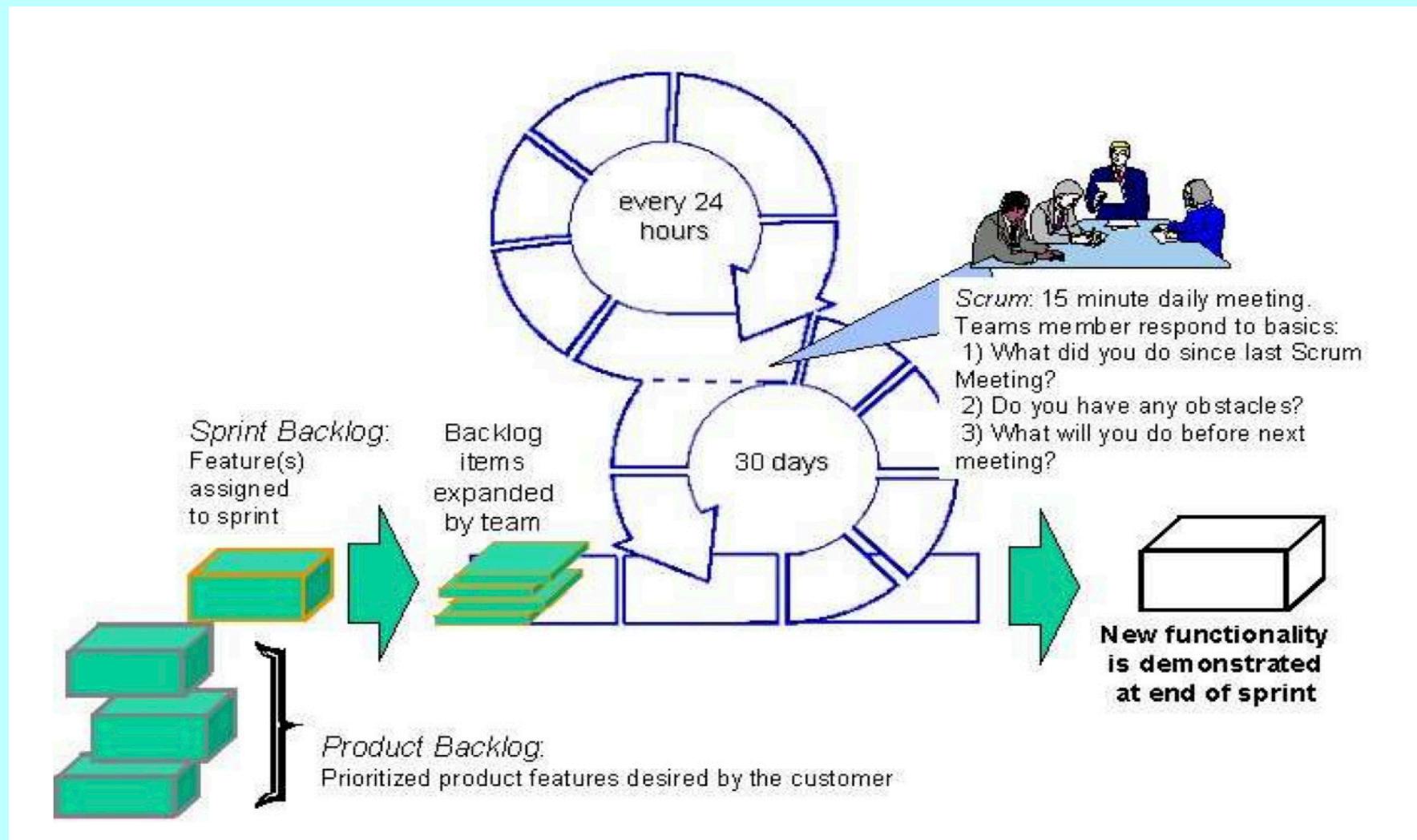
SCRUM

- ◆ Fornisce alla fine di ogni iterazione un set di funzionalità potenzialmente rilasciabili
- ◆ Tre fasi:
- ◆ **Pre-game phase**
 - ◆ *Planning sub-phase*
 - ◆ *Architecture sub-phase*
- ◆ **Development (Game) phase**
 - ◆ Il sistema viene sviluppato attraverso una serie di **Sprint**
- ◆ **Post-game phase**
 - ◆ Contiene la chiusura definitiva della *release*

SCRUM

- ◆ **Sprint:** ciclo iterativo in cui vengono sviluppate o migliorate una serie di funzionalità
- ◆ Ogni *Sprint* include le tradizionali fasi di sviluppo del software; dura da una settimana ad un mese
- ◆ Ruoli
 - **Scrum Master:** gestisce il sottoprogetto
 - **Product Owner:** rappresenta gli *stakeholders*
 - **Team:** gruppo che esegue analisi, progettazione, implementazione, test, etc.

Processo Scrum



Altri termini comuni nell'Agile

◆ **Storie (User stories):**

- Rappresentano le specifiche requisiti
- Descrivono come il sistema si deve comportare rispetto ad una singola caratteristica o funzione
- Devono poter essere implementate in uno Sprint/iterazione

◆ **Tema:** Un insieme di User Story che hanno attributi in comune, condividono uno stesso argomento.

◆ **Epiche:** storia che è troppo generica per essere implementata

- Descrive un insieme di funzioni collegate o un insieme di sotto-funzioni che realizzano un singola funzione complessa.

Problemi dei metodi agili

- ◆ Non sempre i clienti possono partecipare a tempo pieno al processo di sviluppo.
- ◆ I membri del team non sono sempre adatti al coinvolgimento intensivo con altri membri che caratterizza i metodi agili.
- ◆ Dare la priorità alle modifiche può essere difficile dove esistono vari stakeholders, ognuno con suoi interessi.
- ◆ Mantenere la semplicità del sistema può richiedere lavoro extra (e tempo extra, non sempre disponibile).
- ◆ Il contratto può non essere facilmente definito, come accade in altri approcci allo sviluppo iterativo.

DEVOPS

DevOps

Insieme di pratiche
per colmare il gap tra
lo sviluppo (agile)
del software e la fase
operazionale (IT
operations)



L'idea è che i team di sviluppo e di IT *operations* (che si preoccupano di garantire l'operatività del software dopo il rilascio) collaborino durante tutto il ciclo di vita, dal design al processo di sviluppo fino al supporto alla produzione

DevOps: Principi

1. *Systems thinking*
 2. Enfasi sul *feedback* dalla fase operazionale
 3. Cultura della “sperimentazione” continua
- ◆ **CAMS**
- **Culture**
 - ◆ Enfasi sulle persone, comunicazione, cooperazione
 - **Automation**
 - ◆ *Automatic Build, Integrate/Test, Deploy, Monitor*
 - **Measurement**
 - ◆ Supporta la pianificazione, l'analisi dei trend, la qualità
 - **Sharing:** collaborazione, feedback

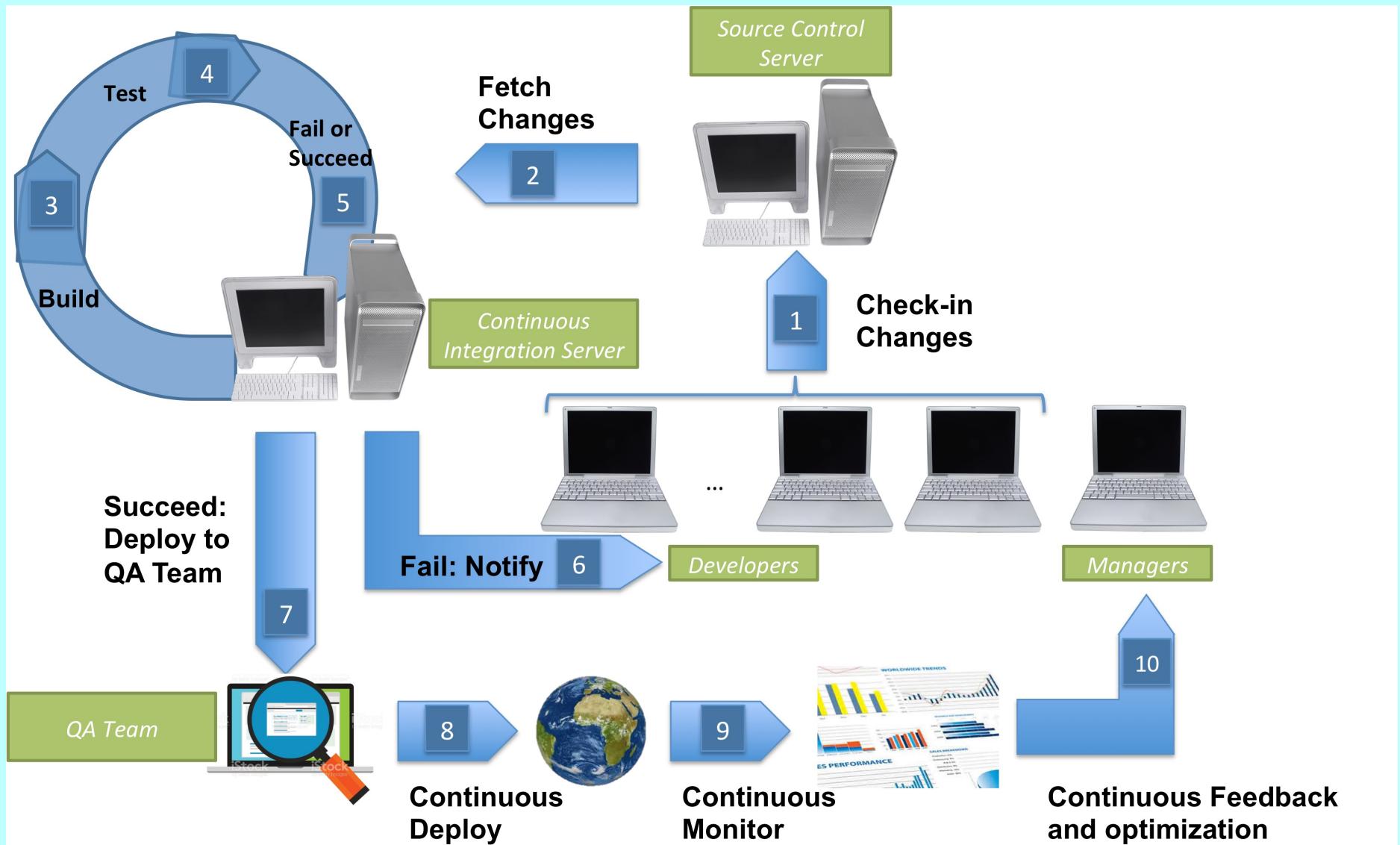
DevOps: Pratiche

- ◆ Test automatico
- ◆ Monitoraggio proattivo
- ◆ Continuous Integration/Deployment/Release
- ◆ Virtualisation/Cloud/Containers
- ◆ Processi (Dev e Ops) “visibili”
- ◆ Toolchain per l'automazione
- ◆ Version control per ogni artefatto

DevOps: Example of Tools

- ◆ Continuous Integration & Deliver (build, test, integrate, deploy)
 - Jenkins, Atlassian Bamboo
- ◆ Cloud (container-based) Deployment
 - Docker, Kubernetes
- ◆ Version Control
 - Github
- ◆ Data/Log Analysis
 - Elastic
- ◆ Monitoring
 - Icinga
- ◆ Resource and configuration management
 - Puppet, Ansible

Ciclo DevOps



Confronto

Confronto

- ◆ Approccio “**code and fix**”. Non può essere considerato un modello: consiste nel seguire l'intuizione, senza alcuna pianificazione
- ◆ L'estremità opposta è costituita dal **modello a cascata**: rigido, pre-specificato, non adattativo, monolitico
- ◆ È solitamente guidato da una corposa documentazione che misura il progresso del progetto => **processo guidato dalla documentazione**
- ◆ Il **modello evolutivo** il progresso è misurato dagli incrementi => **processo guidato dagli incrementi**
- ◆ Similmente, il **modello trasformazionale** è **guidato dalle specifiche**
- ◆ Mentre il meta-**modello a spirale** è **guidato dai rischi**

Confronto

Modello	Punti Forti	Punti Deboli
Code & Fix	Adatto a progetti molto piccoli	Non applicabile a progetti di media/alta complessità; non ripetibile; non strutturato
Waterfall e sue Varianti	Comprensibile, controllabile, ampia documentazione, adeguato ai contratti, chiare <i>milestones</i>	Difficile gestione di requisiti instabili/incompleti; ritarda l'individuazione dei problemi; eccessiva distanza tra specifica dei requisiti e la consegna; bassa tempestività
Evolutivo	Utilizzo di Feedback; gestione efficiente della varabilità dei requisiti; tempestività dei prodotti	Necessità di competenze e preparazione adeguata sia nello sviluppo che nella gestione (ad es. dei rischi)
Agile	Adatto a progetti particolarmente innovativi e con requisiti non ben definiti	A causa della poca strutturazione, necessità di una grande disciplina e motivazione; non molto adeguato ai contratti; difficilmente comprensibile dal cliente

Analisi dei rischi

♦ Modello a cascata

- Alto rischio per sistemi nuovi con requisiti non chiari
 - ◆ Possibili incomprensioni dei requisiti utenti e requisiti basati su assunzioni errate o parziali
 - ◆ Requisiti che evolvono
- Alto rischio di durata eccessiva dello sviluppo => bassa tempestività
- Ciò ha portato a considerare modelli sempre più flessibili
- Basso rischio per sistemi con specifiche e tecnologie assestate

Analisi dei rischi

♦ Modelli evolutivi

- Alto rischio per mancanza di visibilità
- Basso rischio per applicazioni nuove, requisiti ambientali variabili ed alta evolvibilità

♦ Modello trasformazionale

- Alto rischio per necessità di tecnologie avanzate ed alte competenze
- Ogni progetto è caratterizzato dai **propri rischi specifici**, da individuare nella fase iniziale. Il modello può essere scelto in base a tale analisi (come nella spirale di *Bohem*)
- Ad es. *rischi legati alla gestione del personale, pianificazione budget e tempi, rischi di sviluppo funzionalità errate, interfacce inadeguate, requisiti instabili, rischi nell'acquisto di componenti esterni, ...*