Progetto di Informatica III

Sviluppo Agile
(Agile Software Development)

Patrizia Scandurra

Università degli Studi di Bergamo
a.a. 2008-09

Sommario

- Metodologia agile
  - Agile Manifesto
- Che cos'è l'"agilità"
- Processi agili (XP, ASD, DSDP, AUP, AMDD)
- Sviluppo agile tramite AMDD
- Agile Best Practice (*iterazione 0*)
  - *Envisioning*: focus su requisiti e architettura iniziale
  - Modellazione agile di requisiti e architettura di un'applicazione (esempi+ esercizi)

Metodologia agile

- In Ingegneria del SW, per **metodologia agile** (o leggera) o **metodo agile** si intende un particolare metodo per lo sviluppo del software che coinvolge quanto più possibile il committente, ottenendo in tal modo una elevata reattività alle sue richieste
- Esistono un certo numero di tali metodologie
- *Agile alliance*, una organizzazione no-profit creata allo scopo di diffonderle

Manifesto for Agile Software Development

"Scopriamo modi migliori di sviluppare il software facendolo ed aiutando gli altri a farlo. Da queste esperienze, siamo giunti a privilegiare i seguenti elementi :

- *Gli individui e le loro interazioni* rispetto ai processi ed agli strumenti
- *Software funzionante* rispetto ad un'ampia documentazione
- *La collaborazione col cliente* rispetto alla negoziazione dei contratti
- *La pronta risposta ai cambiamenti* rispetto all'esecuzione di un piano

Ovvero, anche se attribuiamo un valore agli elementi riportati a destra, riteniamo più importanti quelli a sinistra"

Kent Beck et al

http://agilemanifesto.org/

Che cos'è l'*Agilità*?

- **Reazione** efficace (rapida e adattiva) ai cambiamenti
- **Comunicazione** efficace fra tutti gli stakeholder
- **Assorbimento del cliente nel team** di sviluppo
- Organizzazione del team che lo ponga in diretto controllo del proprio lavoro

Producendo …

- **Consegne incrementali e frequenti di software**
  - Ogni iterazione è un piccolo progetto a sé stante: pianificazione (*planning*), analisi dei requisiti, analisi, implementazione, test e documentazione
  - Alla fine di ogni iterazione il team deve rivalutare le **priorità di progetto**

Processi agili

- Guidati dalle descrizione del cliente di che cosa gli serve (scenario)
- Basati sull'assunzione che i piani hanno vita breve
- Sviluppano software in maniera iterativa con forte enfasi sulle attività di costruzione
- Producono e consegnano molteplici *incrementi software*
- Si adattano ai cambiamenti

Modelli Agili vs Approcci Convenzionali

- Quando sono da preferire i modelli agili?
  - Nel caso di team di sviluppo piccoli, molto coesi e con elevata esperienza
  - Quando si deve garantire la consegna di un prodotto in tempi estremamente rapidi
- Le dimensioni del sistema da realizzare possono precludere l'applicazione dei modelli agili
  - La realizzazione di sistemi di grandi dimensioni difficilmente può prescindere da una corretta documentazione di tutte le sue parti realizzate e da realizzare

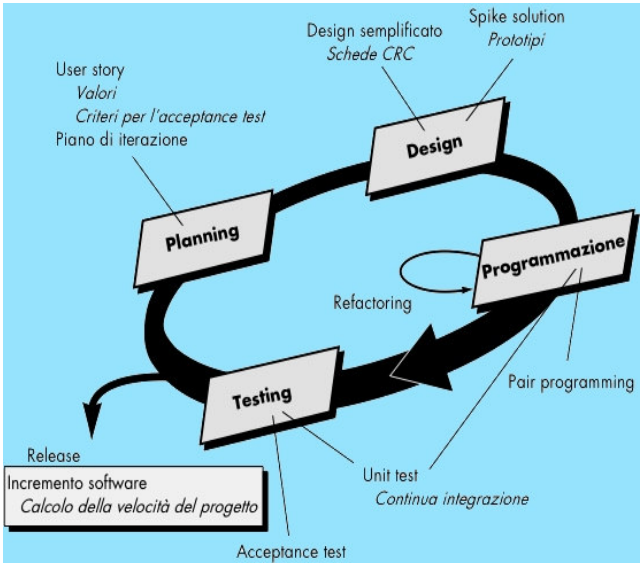
Extreme Programming (XP)

- Il più largamente adottato dei processi agili, proposto inizialmente da Kent Beck
- Il planning in XP
  - Inizia con la creazione di varie "user story"
  - Il team agile valuta ciascuna user story e le assegna un costo
  - Le varie user story sono raggruppate per determinare un incremento da consegnare in tempi brevi
  - Si stabilisce la data di consegna
  - Dopo il primo incremento la "velocità del progetto" viene usata per definire le date delle consegne successive

Extreme Programming (XP)

- Il design in XP
  - Segue il principio KISS (Keep It Simple, Stupid!)
  - Incoraggia l'uso di schede CRC
  - Per problemi di design difficoltosi, suggerisce la creazione di "spike solutions"— **prototipi operativi** per il design di un aspetto
  - Incoraggia il "refactoring"— un raffinamento iterativo del design interno del programma
- La programmazione in XP
  - Prevede la costruzione di **unit test** per le varie **user story** *prima* di iniziare a implementare il sistema
  - Incoraggia il "pair programming" (ruoli driver/observer)
- Il testing in XP
  - Tutti gli unit test vengono eseguiti quotidianamente
  - Test di accettazione (test funzionali) sono **definiti dal cliente** ed eseguiti per verificare le funzionalità visibili al cliente
    - mettere in grado il cliente di accettare o meno il prodotto SW

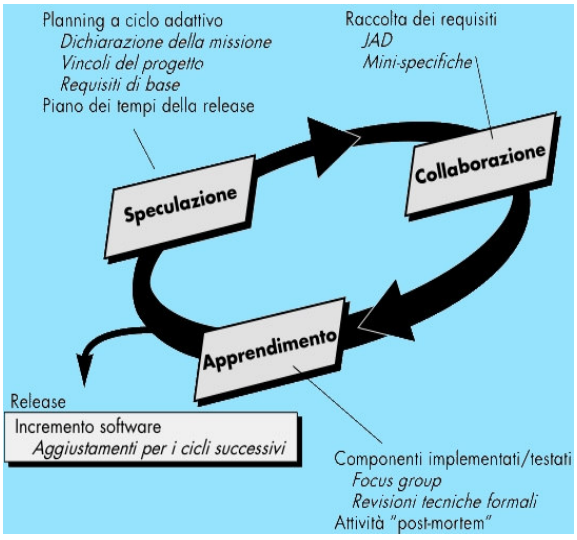
Extreme Programming (XP)



Sviluppo Adattivo di Software
(ASD - Adaptive Software Development)

- Proposto originariamente da Jim Highsmith
- ASD — aspetti caratteristici
  - Planning Mission-driven (guidato dalla *mission* del cliente)
  - Focus basato sulle componenti
  - I rischi sono esplicitamente presi in considerazione nel planning
  - Enfatizza la collaborazione per la raccolta dei requisiti
  - Enfatizza l'apprendimento durante tutto il processo

Adaptive Software Development



Il metodo DSDM
(Dynamic Systems Development Method)

- Promosso e supportato dal DSDM Consortium (www.dsdm.org)
- DSDM — **aspetti caratteristici**
  - Simile sotto molti aspetti a XP e/o ASD
  - Nove principi guida
    - Il coinvolgimento attivo degli utenti è indispensabile
    - Il team DSDM deve avere l'autonomia per prendere decisioni
    - Ci si focalizza su frequenti consegne incrementali
    - La rispondenza agli scopi del business è il criterio essenziale per l'accettazione dei prodotti intermedi
    - Lo sviluppo iterativo ed incrementale è necessario per convergere su una soluzione accurata dal punto di vista del business
    - Tutti i cambiamenti durante lo sviluppo devono essere reversibili
    - I requisiti sono stabiliti ad un alto livello
    - Il testing è integrato durante tutto il ciclo di vita del progetto
    - Continua….

E molte altre…SCRUM, Feature-driven development ecc..

Agile Model Driven Development

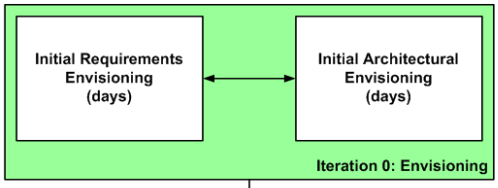
- AMDD = la versione "agile" del Model Driven Development
  - Rispetto ad approcci tradizionali alla modellazione del SW, "With AMDD you do **a little bit of modeling and then a lot of coding**, **iterating back** when you need to"
  - "instead of creating extensive models before writing source code you instead create **agile models** which are just barely good enough that drive your overall development efforts"
- Esempi concreti:
  - **Agile UP (Unified Process)** -- con UML 2 e profili UML
  - **OMG Agile MDA (Model Driven Architecture)** standard -- con UML 2 e profili UML
  - Feature Driven Development (FDD), Use Case Driven Development (UCDD), ecc..

ma AMDD in genere non prescrive il "tipo" dei modelli da utilizzare

- "apply the right modeling artifact", "use the simplest tools"

AMDD life cycle

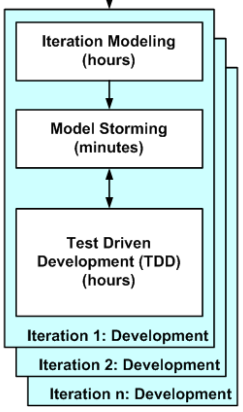
- Identify the high-level scope
- Identify initial "requirements stack"
- Identify an architectural vision



- Modeling is part of iteration planning effort
- Need to model enough to give good estimates
- Need to plan the work for the iteration

- Work through specific issues on a JIT manner
- Stakeholders actively participate
- Requirements evolve throughout project
- Model just enough for now, you can always come back later

- Develop working software via a test-first approach
- Details captured in the form of executable specifications

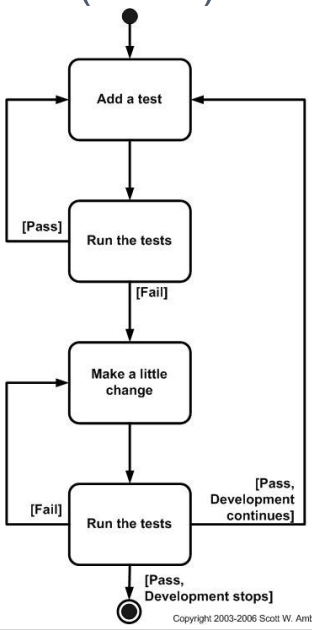


Reviews (optional)

All Iterations (hours)

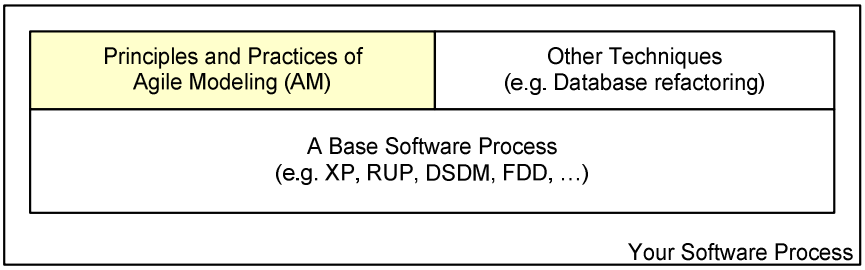
Test-driven development (TDD)

- **TDD = TFD + Refactoring**
- **TFD = test-first design** (in figura)

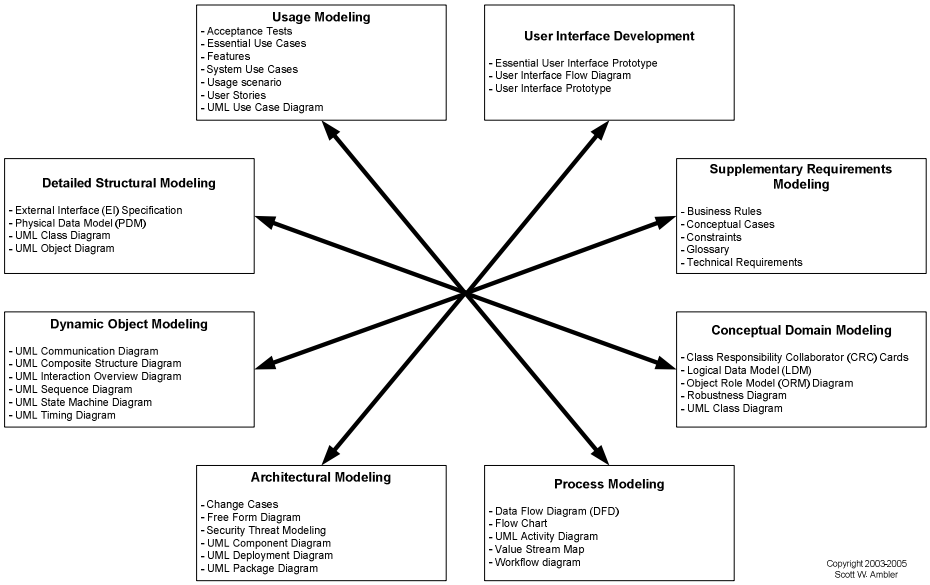


Agile Modeling (AM)

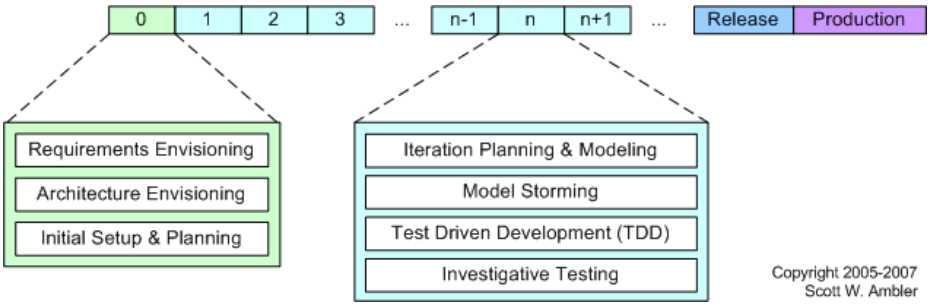
AM = processo leggero e practices-based per modellare e documentare, all'interno di un qualunque processo agile (XP, DSDM, ecc..)



Agile Modeling www.agilemodeling.com/artifacts/



Sviluppo agile con AMDD



Agile Best Practice (iterazione 0)

- *Envisioning*
  - Attività eseguita durante la prima settimana del progetto
  - L'obiettivo è identificare le
    - **caratteristiche**/**requisiti** iniziali del sistema
      Requirements envisioning
    - e una **architettura iniziale** adatta allo scopo
      Architecture envisioning

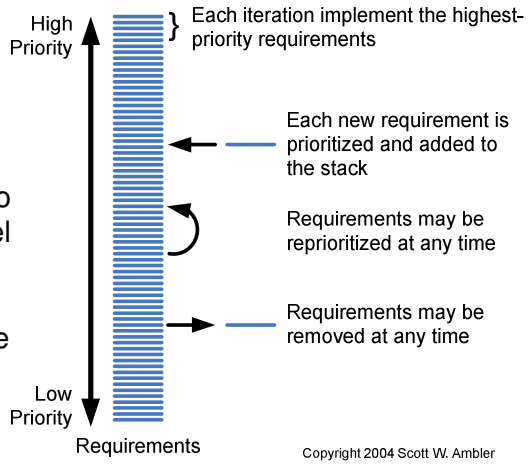
    con AMDD

Requirements Envisioning: An Agile Best Practice

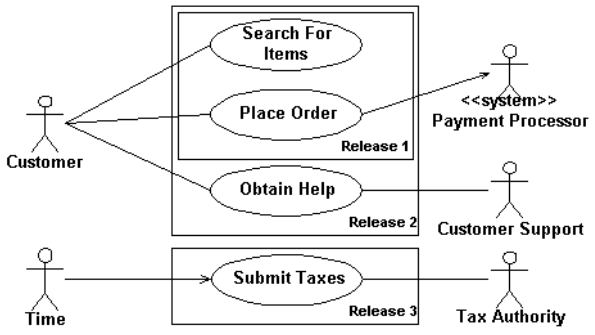
www.agilemodeling.com/essays/changeManagement.htm

Agile requirements change management process

- richiede model storming con strumenti che rendono attiva la partecipazione del cliente
- ad es. con use-case diagram UML, user story e scenari su carta, ecc..



Esempio di use-case diagram UML

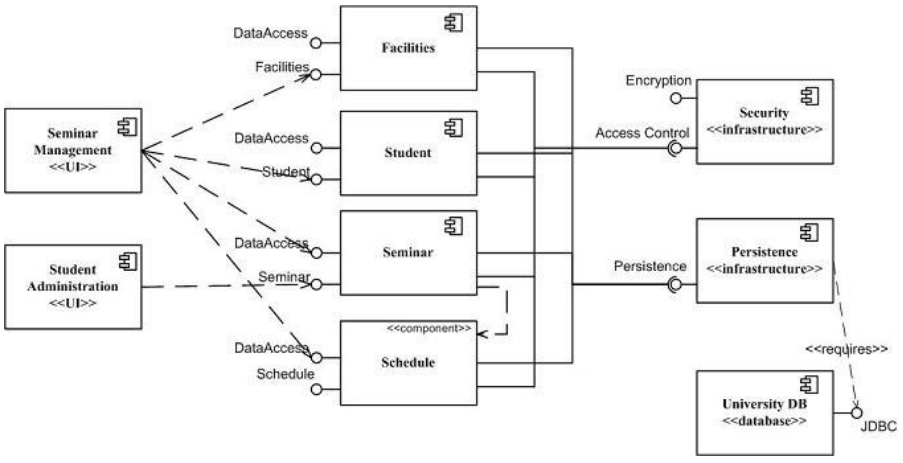


- Decomporre i casi d'uso da implementare in release, stabilendo delle priorità
- Raccogliere i casi d'uso in package (<<subsystem>>)

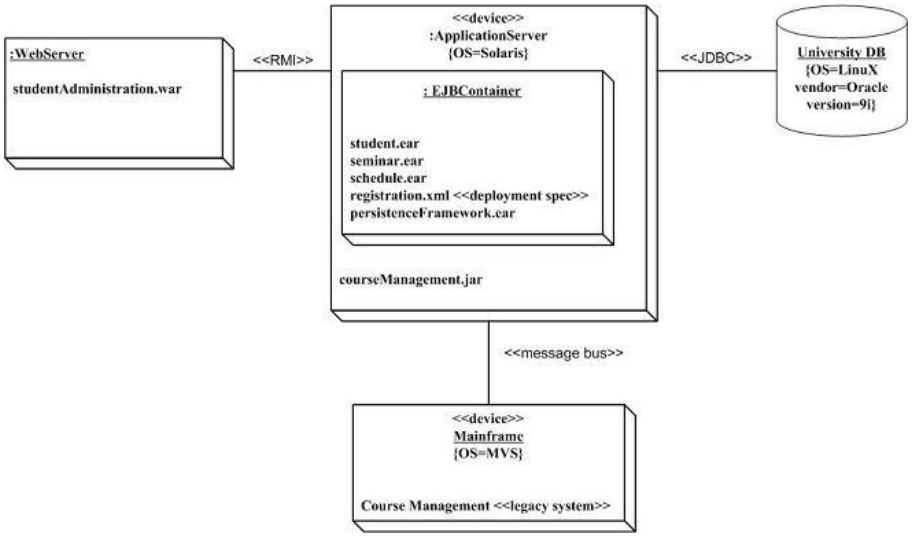
Architecture Envisioning: An Agile Best Practice

- Stabilire l'architettura iniziale del sistema che evolverà poi just in time (JIT) durante le iterazioni di sviluppo con delle sessioni di model storming e per riorganizzare il team stesso
- Modellazione con "**High-level free-form diagrams** which overview how we think we'll build the system"
  - **Technology diagrams** per ragionare sulle componenti software e hardware da impiegare e sulle loro interazioni
    - Ad es. i component/package/deployment diagram UML
  - **User interface (UI) flow** per ragionare sulla navigazione degli elementi della UI (possibilmente grafica -- GUI), inclusi terminali, report, ecc..
  - **Domain models** per *business application* (simili a ER, UML class diagrams) descrivono una *ontologia* delle entità e loro relazioni
  - **Change cases** descrizione testuale di *architecture-level requirements* relativi ad un cambiamento ad es. nella tecnologia o nella regola di business

Esempio di component diagram UML (2.x)



Esempio di Deployment diagram UML



Esempio di requirements/architecture envisioning

University system per l'iscrizione di studenti a dei seminari/corsi

- **Requirements envisioning** con UML
  - use-case diagram
    http://www.agilemodeling.com/artifacts/useCaseDiagram.htm
  - Descrizione testuale dei use-case diagram
    http://www.agilemodeling.com/artifacts/systemUseCase.htm
- **Architecture envisioning** con UML
  - Component diagram (Fig. 1)
    http://www.agilemodeling.com/artifacts/componentDiagram.htm
  - Deployment diagram (Fig. 2)
    http://www.agilemodeling.com/artifacts/deploymentDiagram.htm

Esercizi

1. Requirements envisioning
2. Architecture envisioning

di un “software per la vendita di biglietti”
(vedi *SW_vendita_biglietti_specifiche_utente.pdf*)

Riferimenti

- Libri:
  - Roger S. Pressman, *Principi di Ingegneria del software*, McGraw-Hill, 2008
  - James Shore e Shane Warden. *The Art of Agile Development*. O'Reilly, 2007
  - *The Object Primer 3rd Edition: Agile Model Driven Development with UML 2*, Cambridge University Press, 2004 ISBN 0-521-54018-6
- Siti web:
  - *Manifesto for Agile Software Development*
    - *http://agilemanifesto.org/*
  - *Agile Model Driven Development (AMDD)*
    - http://www.agilemodeling.com/essays/amdd.htm
- Articoli:
  - Scott Ambler's Articles and Other Writings
    http://www.ambysoft.com/onlineWritings.html