Development Methods

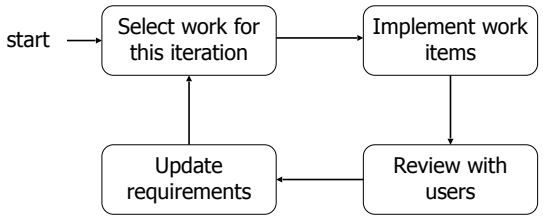
Prof. A. Calvagna

Definition of Process

- A process meets the goals of a project by:
  - Orchestrating activities to achieve the goal
  - Defining artifacts to be produced by the activities
  - Determining the roles and skills needed
  - Setting criteria for progressing from an activity
  - Specifying events for planning and tracking

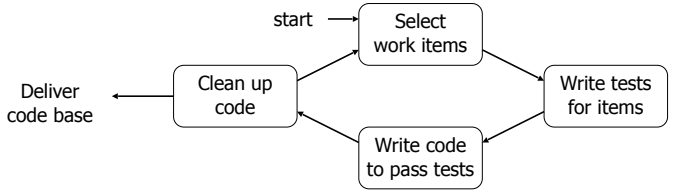
An Iterative Process

- Typically, diagrams show only the activities (as boxes)
  - This particular process shows what is to be done, not how to do it



What Does this process do?

- The diagram has 4 activities and 1 event. Identify the event.



Terminology

- Process Model:
  - A class of processes; e.g., Iterative Model, Waterfall Model
- Framework:
  - Partial specification of a process class; e.g., Scrum, Spiral
- Practice:
  - Specific actionable descriptions of an activity

Common types of activities

- Analisi dei requisiti (specifiche)
- Progettazione (design)
- Implementazione (codifica)
- Convalida (testing e approvazione)
- Rilascio e manutenzione operativa (Deployment & Maintenance)

Analisi dei Requisiti

- porta a definire le specifiche del software: i servizi richiesti ed i vincoli operativi
  - Requisito: ciascuna delle caratteristiche che il software deve avere
  - Specifica: descrizione rigorosa di un requisito
    - I requisiti tendono ad essere granulari (ovvero: molti e piccoli)
  - Feature: un insieme di requisiti correlati tra loro
    - Una feature corrisponde alla soddisfazione di un obiettivo utente (requisito funzionale)

Analisi dei Requisiti

- Passi per l'ingegneria dei requisiti
  - (1) Studio di fattibilità,
  - (2) Analisi dei requisiti,
  - (3) Specifica dei requisiti,
  - (4) Convalida dei requisiti
- Requisiti
  - Funzionali: Cosa il sistema deve fare (funzionalità)
  - Non-funzionali: Come il sistema lo fa (es. affidabilità, efficienza, prestazioni, manutenibilità, etc.)

Esempi di Feature e Requisiti

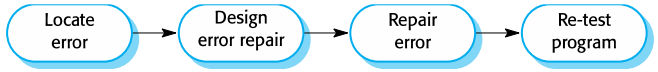
- Feature di Firefox 3.6
  - Browsing privatamente: navigazione del web senza lasciare tracce
  - Password manager: ricordare le password dei siti, senza usare pop-up
  - Awesome Bar: trovare i siti preferiti in pochi secondi
  - One-click bookmark: bookmark, cerca e organizza siti web velocemente e facilmente
- Requisiti (sintetici) di Firefox 3.7
  - Eseguire i plug-in in un processo separato per migliorare la stabilità dell'applicazione e diminuire i tempi di risposta
  - Migliorare i tempi di startup
  - Ottimizzare caricamento delle pagine

Progettazione

- Il processo che stabilisce la struttura software che realizza le specifiche
- Attività della progettazione
  1. Suddivisione dei requisiti
  2. Identificazione sottosistemi, ovvero progettazione architettura software
  3. Specifica delle responsabilità dei sottosistemi
  4. Progettazione di: interfacce, componenti, strutture dati, algoritmi
- Ognuna delle attività suddette produce un documento corrispondente (o integra un documento già esistente) che descrive un modello
  - Modello degli oggetti, di sequenza, di transizione stati, strutturale, data-flow

Implementazione

- Consiste nella programmazione ovvero nella traduzione dei modelli del progetto in un programma (codice) e della rimozione degli errori dal programma
- I programmatori effettuano alcuni test sul programma prodotto per scoprire errori (o bug) e rimuoverli
- Per rimuovere gli errori
  1. Localizzare l'errore nel codice
  2. Rimuovere l'errore nel modello e poi nel codice
  3. Effettuare nuovamente il test nel programma

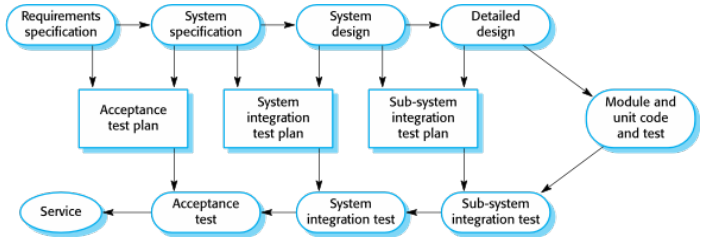


Convalida (Verifica & Validazione)

- L'attività di convalida o Verifica e Validazione (V & V) del sistema software intende mostrare che il sistema software è conforme alle specifiche e che soddisfa le richieste (aspettative) del cliente
  - Viene condotta tramite processi di revisione e test del sistema software
  - I test mirano ad eseguire il sistema software in condizioni derivate dalle specifiche di dati reali che il sistema software dovrà elaborare

Quadro generale di sviluppo

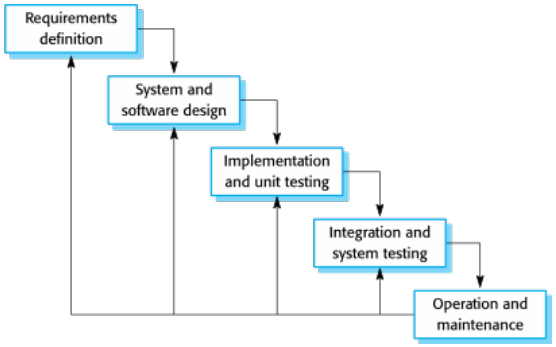
- Dai requisiti (R) otteniamo il documento della specifica dei requisiti (SRS)
- Dall'SRS ricaviamo il design del sistema (DS)
- Dal DS ricaviamo il design dettagliato (DD)
- Da DD ricaviamo codice e test
- Da DS e da DD ricaviamo come integrare i sottosistemi e come fare i test di sistema
- Da R e SRS ricaviamo come fare i test di accettazione



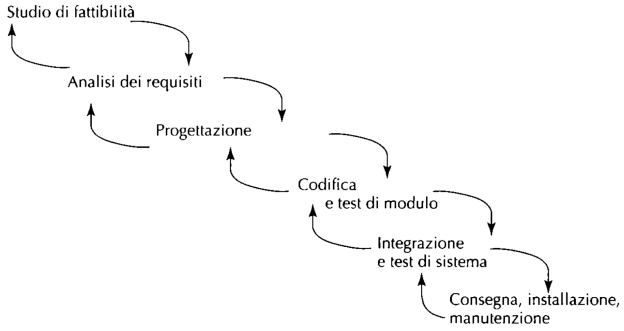
Processo: orchestrazione di attività

- Diverse filosofie/approcci
  - Waterfall (processo di sviluppo rigidamente lineare)
  - Evolutionary (una versione iniziale si trasforma via via in quella finale)
  - Incremental (partiziono lo sviluppo)
  - Component Based (massimizzo il riuso, minimizzo il coding)
  - Spiral (è un framework iterativo, pianifico per ridurre il rischio di cattivo design)
  - Iterativi (XP, Scrum)
  - Agile (è una filosofia, suggerisce processi "leggeri" per agevolare i cambiamenti)
    - XP, Scrum sono agili

Cascata (Waterfall) [Royce 1970]



Waterfall con feedback



Waterfall

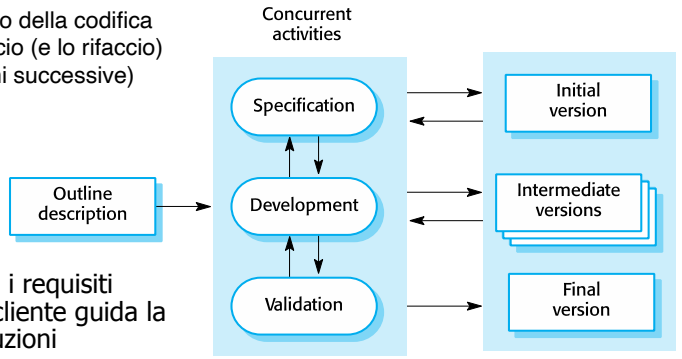
Processo statico con tanta documentazione

- Lungo tempo per ottenere il prodotto
- Poche interazioni con i clienti (solo nella fase iniziale)
- Difficoltà ad introdurre i cambiamenti richiesti dal cliente

+ Consistenza tra artefatti
+ Ampia documentazione
+ Utile se i requisiti sono stabili e chiaramente definiti
+ Adatto a sistemi grandi, complessi, critici, per gestire team numerosi
+ Alta qualità del codice prodotto

Processo Evolutivo

Non voglio ritardare l'inizio della codifica dell'intero sistema: lo faccio (e lo rifaccio) **per intero** (trasformazioni successive)



- Variante esplorativa: i requisiti iniziali ben chiari, il cliente guida la sequenza delle evoluzioni
- Build & Fix: documentazione quasi inesistente, bassa qualità

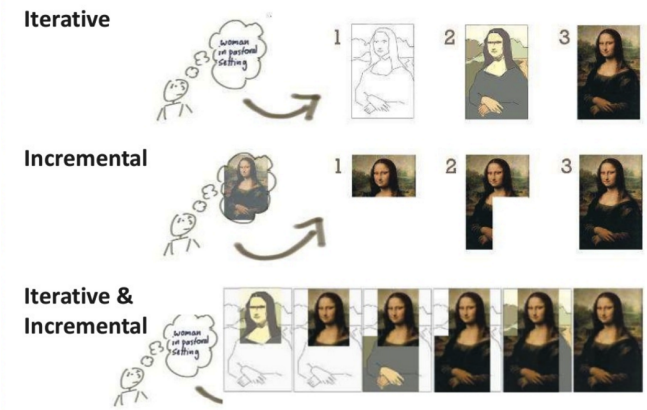
Evolutivo

- Problemi
  - Tempi lunghi
  - Sistemi difficilmente comprensibili e modificabili, probabilmente non corretti
  - Mancanza di visione d'insieme del progetto
- Applicabilità
  - Sistemi di piccole dimensioni
  - Singole parti di sistemi grandi (es. interfaccia utente)
  - Sistemi con vita breve (es. prototipi)

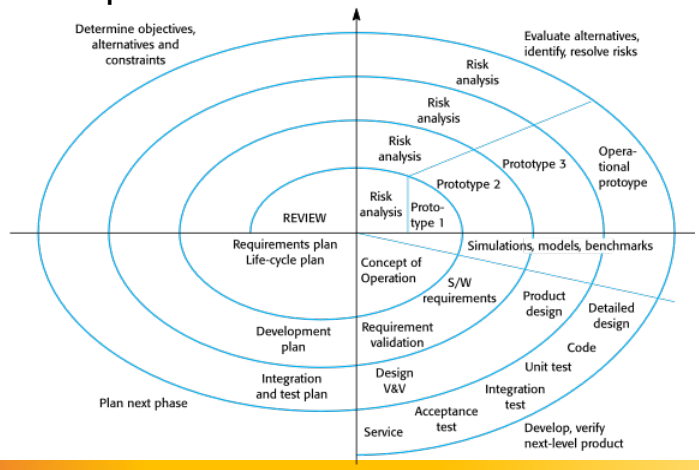
Altri Processi

- Processo di Sviluppo Incrementale
  - Sono implementate prima le funzionalità di base (o prioritarie)
  - Al codice sviluppato in precedenza è aggiunto altro codice per un altro gruppo di funzionalità
  - Si ripete il passo precedente, fino a completamento
- Processo CBSE o basato su COTS
  - COTS = componenti esistenti (Components Off The Shelf)
  - Analisi dei componenti esistenti
  - Modifica dei requisiti (?)
  - Progettazione tramite riuso
  - Sviluppo (di adattatori) ed integrazione

Iterativo vs Incrementale



Modello a Spirale



Settori del processo a Spirale

- Stabilire obiettivi
  - Gli obiettivi per la fase corrente sono identificati
- Valutare il rischio e ridurlo
  - I rischi sono valutati ed attività sono intraprese per ridurre quelli più importanti
- Sviluppo e convalida
  - Secondo uno dei modelli precedenti
- Pianificazione
  - Il progetto è revisionato e la prossima fase della spirale è pianificata

Agile (Software Development)

- The term was coined at a February 2001 meeting
  - By self-described "independent thinkers about software development"
- The same meeting resulted in the Agile Manifesto
  - Attendees included Ken Schwaber and Jeff Sutherland who created Scrum
  - And Kent Beck, responsible for Extreme Programming (XP)
  - Plus proponents of several other agile methods
- Developers have the flexibility and accountability for doing the work

Contrasting Agile and Plan-Driven Development

- Agile Manifesto
  - –Individuals and interactions
  - –Working Software
  - –Customer Collaboration
  - –Responding to Change
- Plan Driven (Plan then Build)
  - –Processes and tools
  - –Comprehensive documentation
  - –Contract negotiations
  - –Following a plan

What is an Agile Method?

- The Agile Alliance's definition of an agile method:
  - – It's agile if it conforms to the Agile Manifesto and the principles behind it
- Scrum and XP are pre-Manifesto agile methods
  - – Both date back to the mid 1990s

Development Methods Help Manage Risks

- We use term risk informally for a potential challenge
  - – Risk can be thought of as the cost of recovering if something goes wrong
- Process models tend to focus on specific risks, as we'll see
  - – Agile methods are iterative, so they manage risk related to requirements
  - – Plan-driven methods hope to manage risk related to design

Agile Development with XP

Extreme Programming (XP)

- Kent Beck defined XP in the mid-1990s, while on a project for Chrysler
- The name comes from taking common-sense practices to the extreme:
  - Example: "If design is good, we'll make it part of everybody's daily business (refactoring)."
- XP was refined in the years following the Chrysler project
  - There are now XP values, principles, activities, and practices

Overview of XP

- Beck describes XP as follows:
  - "an always deployable system to which features, chosen by the customer, are added and automatically tested on a fixed heartbeat."
- The "heartbeat" or iteration length is often a week or two (mai oltre 4)

The Four Main XP Activities

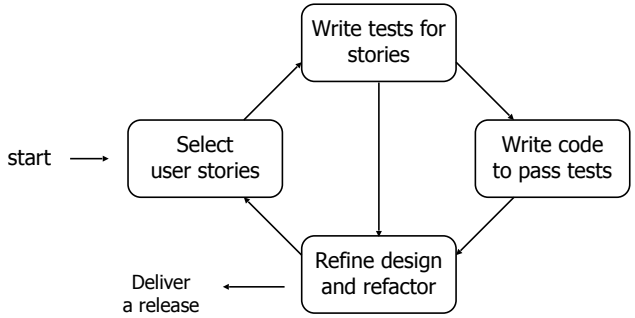
- Listening
- Testing
- Coding
- Design
  - XP teams can face design risk if they defer design until the last minute
    - When to Design remains an issue

Four Main Activities

- Listening: to what users want from a system
- Testing: Make it central to development
- Coding: Write simple readable code
- Design: Refactor to clean up the design and implementation
- These activities come together in the XP development cycle (next slide)

The Test-Driven XP Development Cycle

User stories express requirements, as we'll see



Note that the XP cycle allows an iteration to bypass coding

Listening: Writing User Stories (Requirements)

User Stories are for Snippets of Functionality

Write from the user's perspective

- Connextra User Story Template
- As a <role> I want to <action> so that <benefit>
- Example:
  - As a customer,
  - I want to withdraw cash from an ATM
  - so that I can get cash when the bank is closed

Examples: Stories for a Ride-Sharing Service

- Example 1:
  - As a rider
  - I want flexible on-demand transportation
  - so that I can get to my destination whenever I want
- Example 2:
  - As a rider
  - I want a trusted driver
  - so that I will feel safe getting into the driver's vehicle

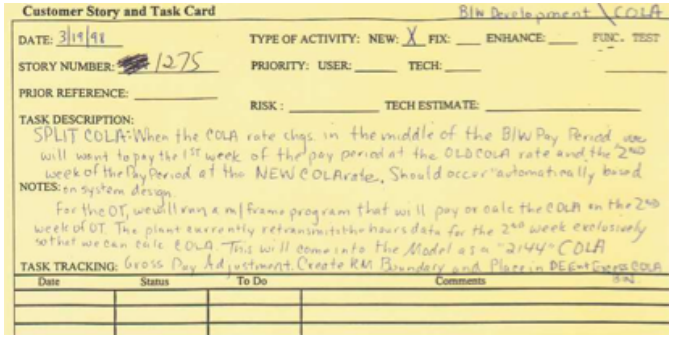
General Guidelines for User Stories

Purpose is identifying a user requirement

- Keep it small enough to fit on a 3x5 inch index card
- Use simple plain language, from the user's point of view
- Write about a user need, not about technology to meet the need
- Use the template: As a <role> I want to <action> so that <benefit>

  – Confirm the story by writing one or more acceptance tests

Story Card


Customer Story and Task Card

DATE: 3/19/98

STORY NUMBER: 1275

PRIOR REFERENCE:

TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST

PRIORITY: USER: ___ TECH: ___

RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:

SPLIT COLA: When the COLA rate chgs in the middle of the BIW Pay Period we will want to pay the 1st week of the pay period at the OLD COLA rate and the 2nd week of the Pay Period at the NEW COLA rate. Should occur automatically based on system design.

NOTES:

For the OT, we will run a mainframe program that will pay or calc the COLA on the 2nd week of OT. The plant currently retransmits the hours data for the 2nd week exclusively so that we can calc COLA. This will come into the Model as a "2144" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in PEE at Exec COLA

Date	Status	To Do	Comments

Rather than a lengthy requirements document, the customer writes user stories, which define the functionality the customer would like to see, along with the business value and **priority of each of those features**.

With XP ,Testing Plays a Central Role

Testing Supports All Activities

- Three forms of testing
  - Acceptance tests for user stories
  - Unit tests for the code
  - Regression tests for guarding against breaking existing functionality
- Testing after every change increases confidence in the code
  - Aids debugging: either the change was faulty or it broke something
  - Such Testing can actually shorten overall development time

Validation and Verification

- Validation is for functionality and requirements
  - The focus is on ”Am I building the right product?”
- Verification is for implementation correctness
  - The focus is on “Am I building the product right?”
- During test-driven development
  - Before coding, write acceptance tests for validation
  - After coding, write additional unit tests for verification

Acceptance-Test Template for User Stories

- In effect, acceptance tests define what a product does
- Simple form of the template:
  - Given <some context> when <triggering event> then <expect outcome>
- There may be multiple acceptance tests for a user story
- Write acceptance tests so they are executable

General Form of the Acceptance Template

- Given <some initial context>
- and <some more context>
- when <triggering event>
- then <expect outcome>
- and <another outcome>

- There may be multiple and-clauses for context and outcomes

Example: Case 1 for Cash Withdrawal

Account is in Credit

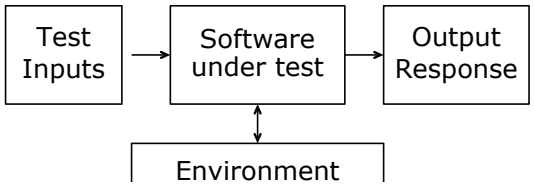
- Given the account is in credit
- and the card is valid
- and the dispenser has cash
- when the customer requests cash
- then update account balance
- and dispense cash
- and return card

Example: Case 2

Account is Overdrawn Past the Credit Limit

- Given the account is overdrawn
- and the card is valid
- when the customer requests cash
- then display "sorry" message
- and return card

Relating the Acceptance Template to Testing



- The "Given" context sets up the environment
- The "when" event corresponds to an input
- The "then" response corresponds to the expected output

Coding: Write Simple Readable Code

Test-Driven Development

Dates back to NASA's Project Mercury in the 1960s

- Write acceptance tests before coding
  - The tests should fail for the right reasons
- Then, write just enough code to pass the tests
  - Add unit tests to verify the correctness of the new code
- Refactor to clean up the design and the code

Refactoring: Code Clean UP and Restructuring

Refactoring consists of a sequence of correctness-preserving changes

- Restore modularity, so each module has a specific responsibility
- Put common (duplicated) functionality into a single method or module
- Improve control flow for readability and predictability of behavior
- Ensure that variable and method names reflect their purpose
- Add comments for rationale and what not to change

Design: When and How Much?

When to Design?

The answer varies from project to project

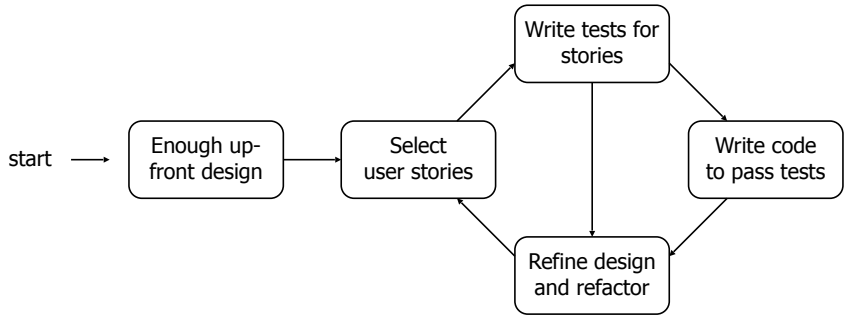
- Too much design too soon
  - Can lead to wasted effort on something that will not be needed
- Too little design too late
  - Can result in an unwieldy design that needs rework
- How do we avoid either wasted effort or poor design?

Early Failures Due to Last Minute Design

From the 2nd edition of Extreme Programming Explained by Beck and Andres

- Kent Beck writes that early XP teams misinterpreted the 1st edition
  - They deferred design until the last possible moment
  - And ended up with brittle poorly designed systems
- He recommends deferring design until the "last responsible moment"
  - "Keep design investment in proportion to the needs of the system"

Modified XP Cycle with Some Up-Front Design



When to Design Depends on Product and Requirements

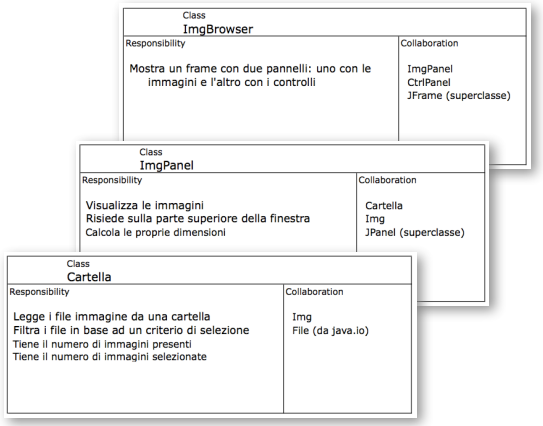
Specifically, on product complexity and requirements stability

PRODUCT	Stable	Changing
Complex	Design properties into the system up front	Start with a modifiable modular design
Simple	Minimal initial design to set direction	Defer major decisions until needed

REQUIREMENTS

CRC cards

- Class Responsibility Collaboration
- Documentano l'architettura in XP
- Aiutano a discuterne e ragionarci
- Molto più informale e sintetico di un class diagram



12 Pratiche di XP

- Gioco di pianificazione: users vs dev.s
- Piccole release: tipicamente una ad iterazione
- Usare Metafore: nomi significativi aiutano a capire e comunicare in modo immediato
- Testing continuo: anche più volte al giorno
- Refactoring: per design e debito tecnico
- Pair Programming: static review, test
- Cliente in sede: comunicazione continua in tempo reale, collabora e vede i progressi
- Design semplice: cambiarlo deve essere sempre possibile
- Possesso del codice collettivo: tutti sono responsabili di tutto
- Integrazione continua: non meno di una volta al giorno
- Settimana di 40 ore: evitare rischio stress
- Usare gli standard per il codice: migliora la leggibilità

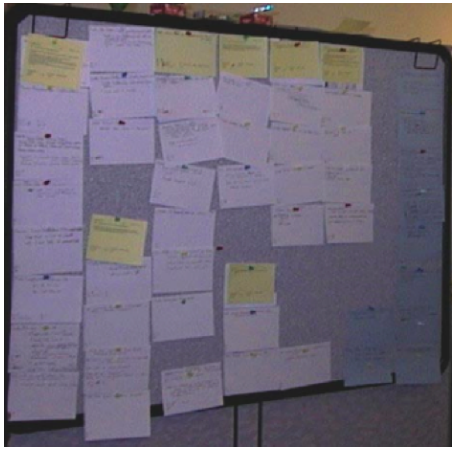
http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

Story Board

- Gli utenti (clienti) scrivono le storie (i requisiti) e gli assegnano **priorità**
- Gli sviluppatori stimano il **tempo** per lo sviluppo
- I clienti riempiono **fino a 3-4 settimane** scegliendo le storie (aggiornabili) per coprire almeno la prossima micro release/iterazione

- Per l'attuale release, **gli sviluppatori:**
  Dividono ciascuna storia in task, stimano i task, **ciascuno si impegna** per realizzare un task

- Gli addetti al business prendono decisioni su
  - Date per le varie main release (1-3 mesi)
  - **priorità dei task** (prima i più rischiosi)



Pair Programming Remains Controversial

- Pair programming is two people on the same task at the same screen
  - The proposed benefit is that code is reviewed all the time
  - Also, if someone leaves, there is a person who knows the code
- The practice has not caught on
  - Incompatibilità caratteriali, antipatie