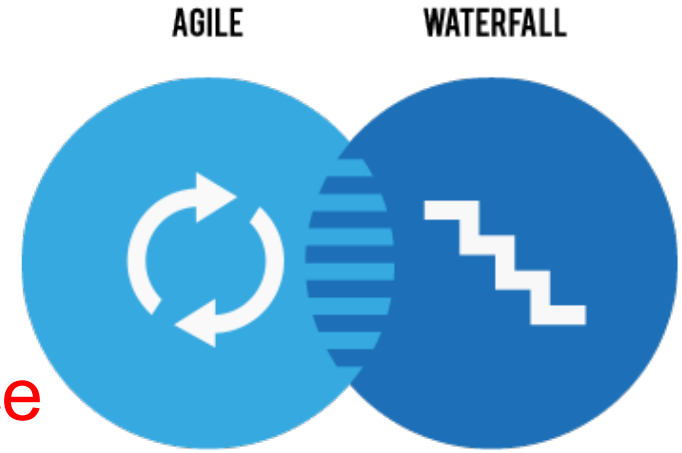
Modelli di processo per lo sviluppo del software: modelli agili, quality driven, open source

Prof. Paolo Ciancarini
Corso di Ingegneria del Software
CdL Informatica
Università di Bologna

Obiettivi di questa lezione

- I modelli di processo agili
- I modelli orientati alla qualità
- I modelli orientati all'open source



Metodi agili

- Extreme Programming (XP)
- Scrum
- Feature-Driven Development (FDD)
- Adaptive Software Process
- Crystal Light Methodologies
- Dynamic Systems Development Method (DSDM)
- Lean Development

Etica del Movimento Agile

Stiamo scoprendo modi migliori di costruire il software facendolo e aiutando altri a farlo. Attribuiamo valore a:

Individui e interazioni più che a processi e strumenti
Software che funziona più che a documentazione completa
Collaborazione col cliente più che a negoziazione contrattuale
Reagire al cambiamento più che a seguire un piano

I concetti a destra sono importanti, ma noi preferiamo quelli a sinistra

`www.agilemanifesto.org`

Anti-methodology?

The Agile movement is not anti-methodology, in fact, many of us want to restore credibility to the word *methodology*.
We want to restore a balance.

We embrace *modeling*, but not in order to file some diagram in a dusty corporate repository.
We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes.
We plan, but recognize the limits of planning in a turbulent environment.

Jim Highsmith, *History: The Agile Manifesto*

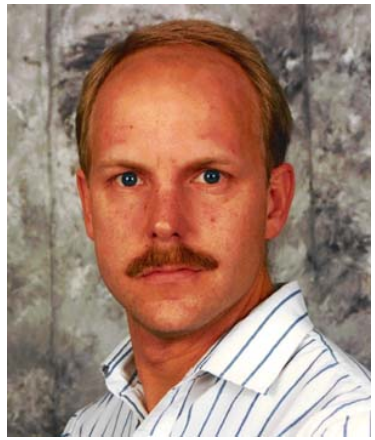
Metodi agili

I metodi agili sono una famiglia di metodi di sviluppo che hanno in comune:

- **Rilasci frequenti** del prodotto sviluppato
- **Collaborazione** continua del team di progetto col **cliente**
- **Documentazione** di sviluppo **ridotta**
- **Valutazione** sistematica e continua di **valori** e **rischi** dei **cambiamenti**

eXtreme Programming (XP)

“Extreme Programming is a discipline of software development based on values of *simplicity*, *communication*, *feedback*, and *courage*”.



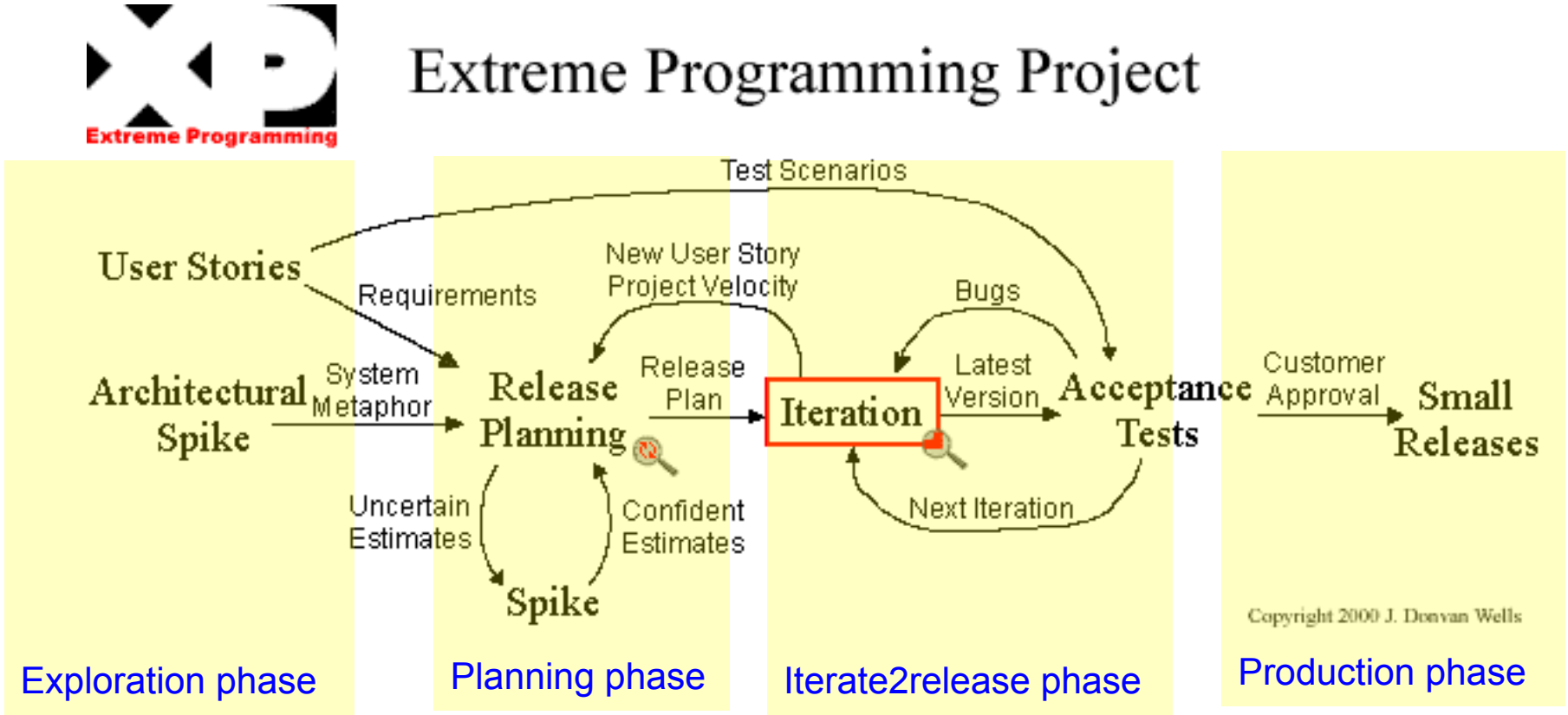
Kent Beck

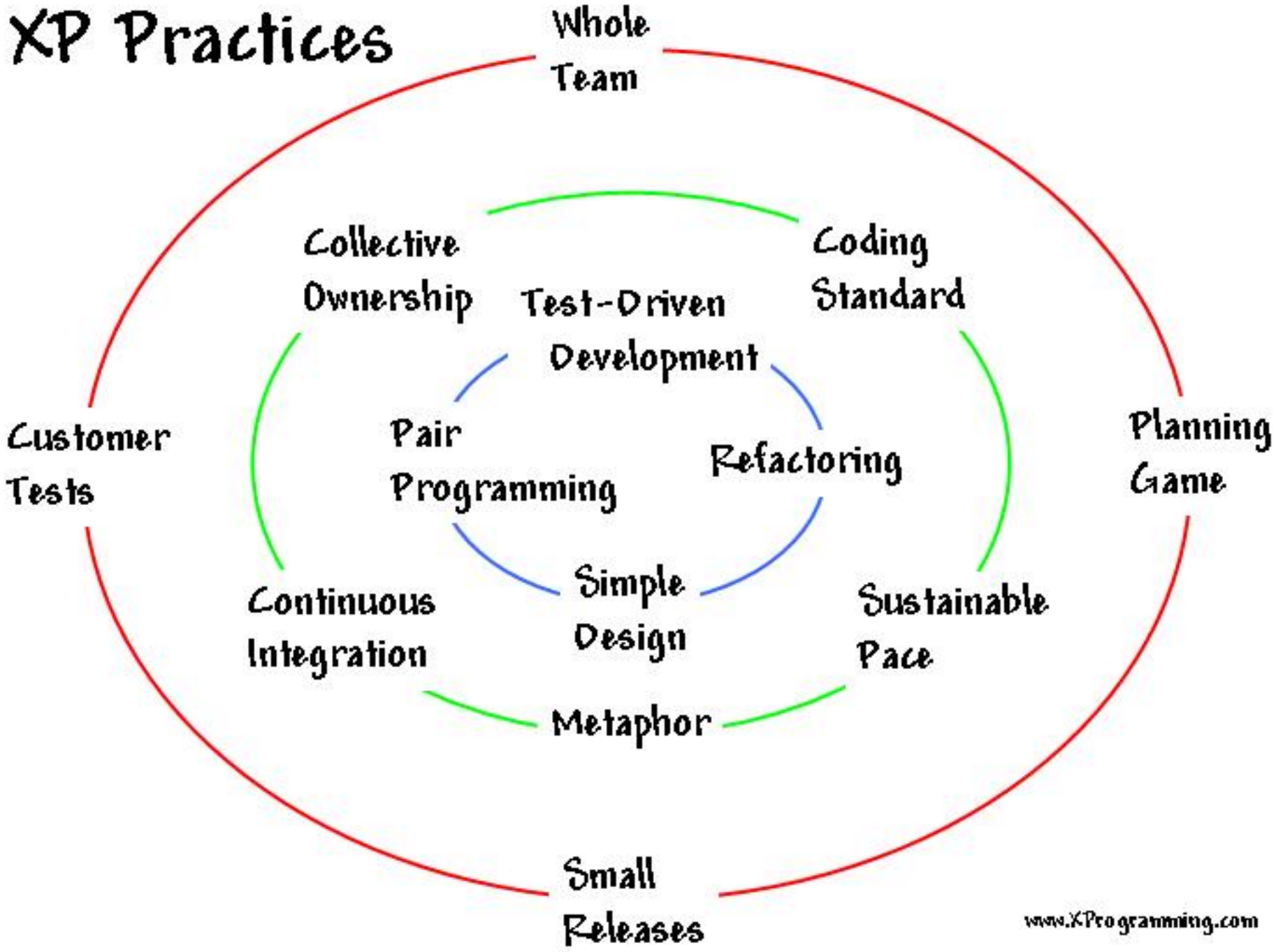
Il team

- Il team di sviluppo, di solito costituito da meno di dieci persone, è riunito nello stesso locale, con un rappresentante del cliente
- Il team deve usare comportamenti di sviluppo **semplici**, allo stesso tempo **capaci di informare tutti** sullo stato del progetto ma anche di **adattare** i comportamenti alla situazione specifica

eXtreme Programming (XP)

www.extremeprogramming.org




XP Practices
Whole Team
Collective Ownership
Coding Standard
Test-Driven Development
Pair Programming
Refactoring
Customer Tests
Planning Game
Continuous Integration
Simple Design
Sustainable Pace
Metaphor
Small Releases
www.XProgramming.com

I principi di Extreme Programming

- I requisiti sono “user stories”
- Il “planning game”
- Piccoli rilasci
- Cliente On-site
- “Prima i test, poi il codice”
- La metafora di riferimento
- Integrazione continua
- Proprietà collettiva del codice
- Settimana di 40 ore di lavoro
- Uso sistematico di standard di codifica
- Programmazione di coppia
- Refactoring
- Progettazione semplice

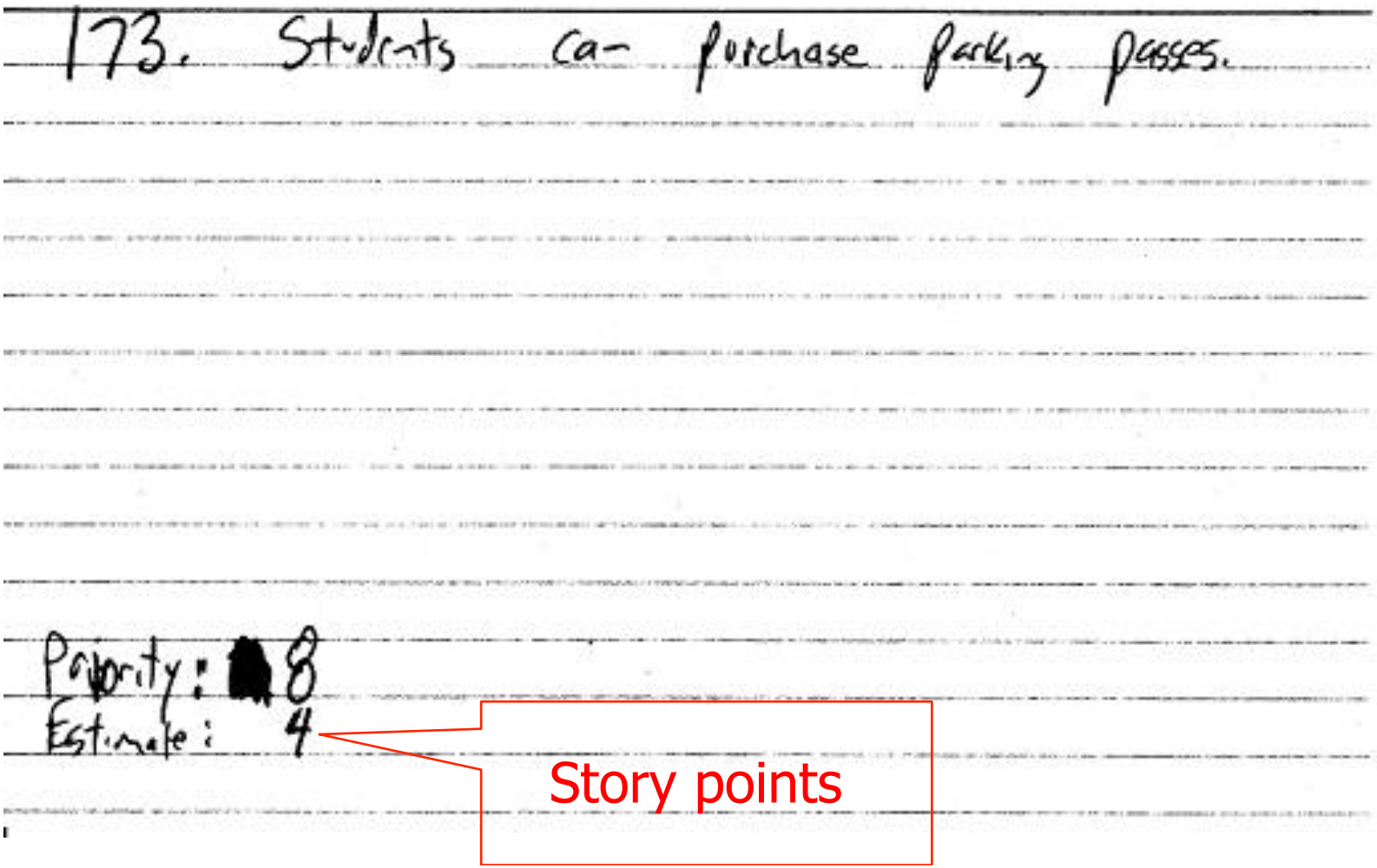
I requisiti sono “user stories”

- User stories:
  - usate al posto di documenti dettagliati di requisiti
  - scritte dai clienti: cosa si aspettano dal sistema
  - una storia è descritta da una o due frasi in testo naturale, con la terminologia del cliente (*no techno-syntax*)
  - utili per stimare i tempi/costi di un rilascio (*release planning)*

Esempi di user stories

- Students can purchase monthly parking passes online
- Parking passes can be paid via credit cards.
- Parking passes can be paid via PayPal
- Professors can input student marks
- Students can obtain their current seminar schedule
- Students can order official transcripts
- Students can only enroll in seminars for which they have prerequisites
- Transcripts will be available online via a standard browser

Esempio di scheda per user story



Il “Planning game”

- I requisiti vengono organizzati in “**User Stories**”
  - Una storia è una breve descrizione di qualcosa che vuole il cliente
  - La descrizione può essere arricchita da altre storie durante lo sviluppo
  - Le **priorità** tra le storie sono definite dal **cliente**
  - **Rischi** e **risorse** necessarie sono valutati dagli **sviluppatori**
- “The **Planning Game**”
  - Le storie di più alto rischio e priorità sono affrontate per prime, in incrementi “*time boxed*”

- Il Planning Game si rigioca dopo ciascun incremento

Storie e iterazioni

La presenza del cliente

- Il cliente (un suo rappresentante) è sempre disponibile per chiarificare le storie e per prendere rapidamente decisioni critiche
- Gli sviluppatori non devono fare ipotesi
- Gli sviluppatori non devono attendere le decisioni del cliente
- La comunicazione “faccia a faccia” minimizza la possibilità di ambiguità ed equivoci

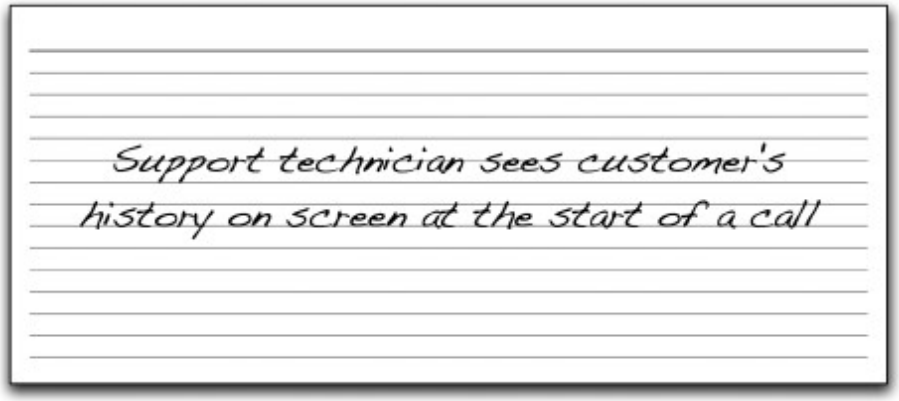
Sviluppo Test-driven

- Test-Driven Development (TDD)
  - Scegliere una user story
  - Definire i test prima del codice
  - Automatizzare i test
  - Usare xUnit
  - Deve girare tutto prima di proseguire con altro codice
- *Unit test* e *acceptance test*

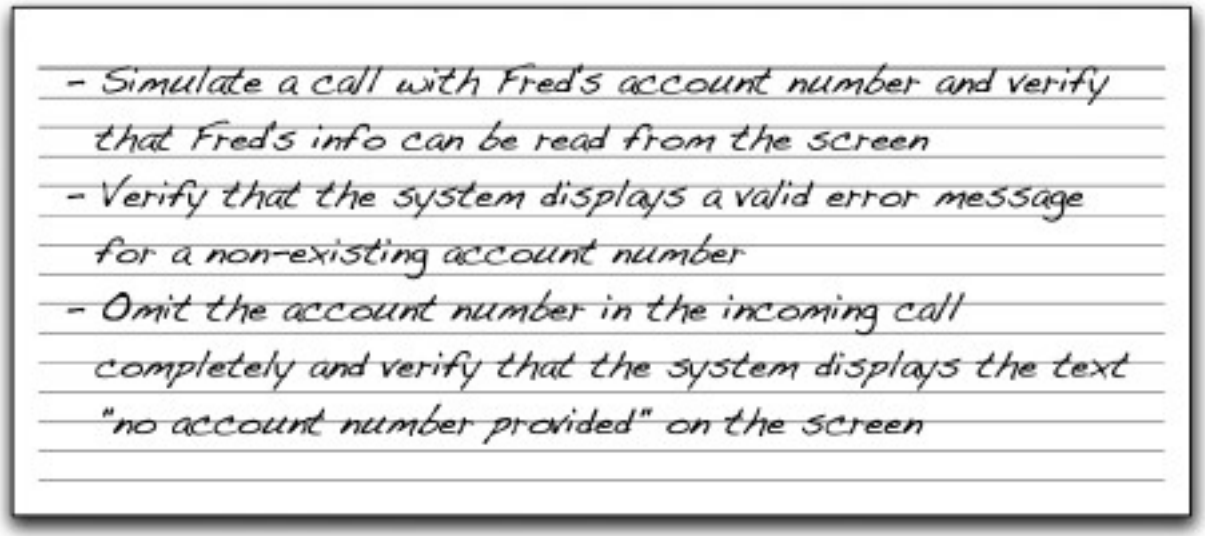
Customer Tests

- Test di accettazione
  - Guidati dalle user stories
  - Scritto col cliente
  - Funziona come “contratto”
  - Misura del progresso

Esempio



Esempio di user story su scheda



Esempio di test scritto sul retro della user story

Piccoli rilasci

- Timeboxed (ovvero di durata “breve” prefissata)
- Minimali, ma comunque utili (microincrementi)
  - Mai cose come ‘implementare il database’
- Ottenere feedback dal cliente presto e spesso
- Eseguire il “planning game” dopo ciascuna iterazione
  - Si voleva qualcosa di diverso?
  - Le priorità sono cambiate?

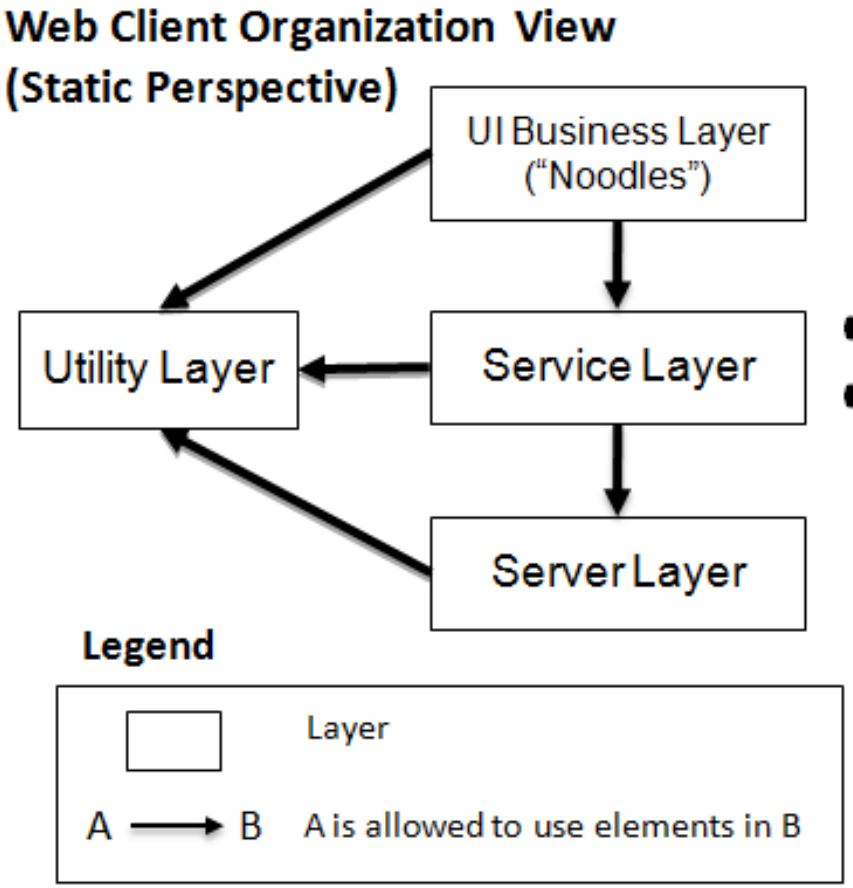
La metafora

- I progettisti XP sviluppano una visione comune di come funzionerà il programma, detta "**metafora del sistema**"
- Esempio: "*questo programma funziona come uno sciame d'api, che cerca il polline a lo porta nell'alveare*" (sistema di information retrieval basato su agenti)
- Non sempre la metafora è poetica. In ogni caso il team deve usare un glossario comune di nomi di entità rilevanti per il progetto

http://agilemetaphors.com

Esempio di metafora

Fonte: neverletdown.net/topics/architecture/



The "Bento Box"

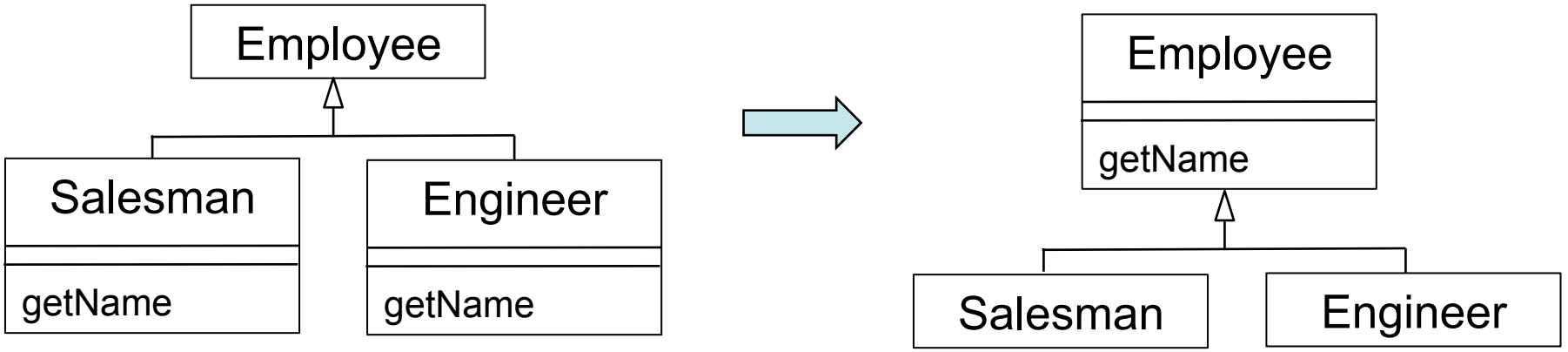
Progettare in modo semplice

- No Big Design Up Front (BDUF)
- “Fare la cosa più semplice che possa funzionare”
  - Includere la documentazione
- “You Aren’t Gonna Need It” (YAGNI)
- Opzione: usare schede CRC
  (vedere le lezioni sui requisiti e sul design per un’introduzione alle schede CRC)

Rifattorizzare (refactoring)

- Refactoring: migliorare il codice esistente senza cambiarne la funzionalità
- Avere il coraggio di buttare via codice
  - Semplificare il codice
  - Rimuovere il codice ridondante
  - Cercare le opportunità di astrazione
- Il refactoring usa il testing per assicurare che con le modifiche non si introducano errori

Refactoring: esempio



Le sottoclassi hanno ciascuna metodi con risultati identici

Spostando il metodo comune nella superclasse, si elimina la ridondanza.
Eliminare le ridondanze nel codice è importante

Pair Programming

La programmazione di coppia (pair programming) è tipica di eXtreme Programming (XP)

Con la programmazione di coppia:

- Due progettisti lavorano allo stesso compito su un solo computer
- Uno dei due, il driver, controlla tastiera e mouse e scrive l'implementazione
- L'altro, il navigatore, osserva l'implementazione cercando difetti e partecipa al brainstorming su richiesta
- I ruoli di driver e navigatore vengono scambiati tra i due progettisti periodicamente (es. giornalmente)

Integrazione continua

- La coppia scrive i test ed il codice di un task (= parte di una storia utente)
- La coppia esegue tutto il test di unità
- La coppia esegue l'integrazione della nuova unità con le altre
- La coppia esegue tutti i test di regressione
- La coppia passa al task successivo a mente sgombra (capita una o due volte al giorno)
- L'obiettivo è prevenire l' "*Integration Hell*"

Proprietà collettiva del codice

- Il codice appartiene al progetto,
  non ad un individuo
- Durante lo sviluppo tutti possono osservare e **modificare** qualsiasi classe
- Uno degli effetti del *collettive ownership* è che il codice troppo complesso non sopravvive a lungo
- Affinché tale strategia funzioni occorre l'"integrazione continua"

Passo sostenibile

- Kent Beck dice: “ . . . freschi e vogliosi ogni mattina, stanchi e soddisfatti ogni sera”
- Lavorare fino a tarda notte danneggia le prestazioni: uno sviluppatore stanco fa più errori, e alla lunga il gioco non vale la candela
- Se il progetto danneggia la vita privata dei partecipanti, alla lunga lo scotto lo paga il progetto stesso

Codifica

- Usare convenzioni di codifica
  - A causa del Pair Programming, delle rifattorizzazioni e della proprietà collettiva del codice, occorre avere metodi per poter addentrarsi velocemente nel codice altrui
- Commentare il codice
  - Priorità al codice che “svela” il suo scopo
    - Se il codice richiede un commento, riscrivilo
    - Se non puoi spiegare il codice con un commento, riscrivilo

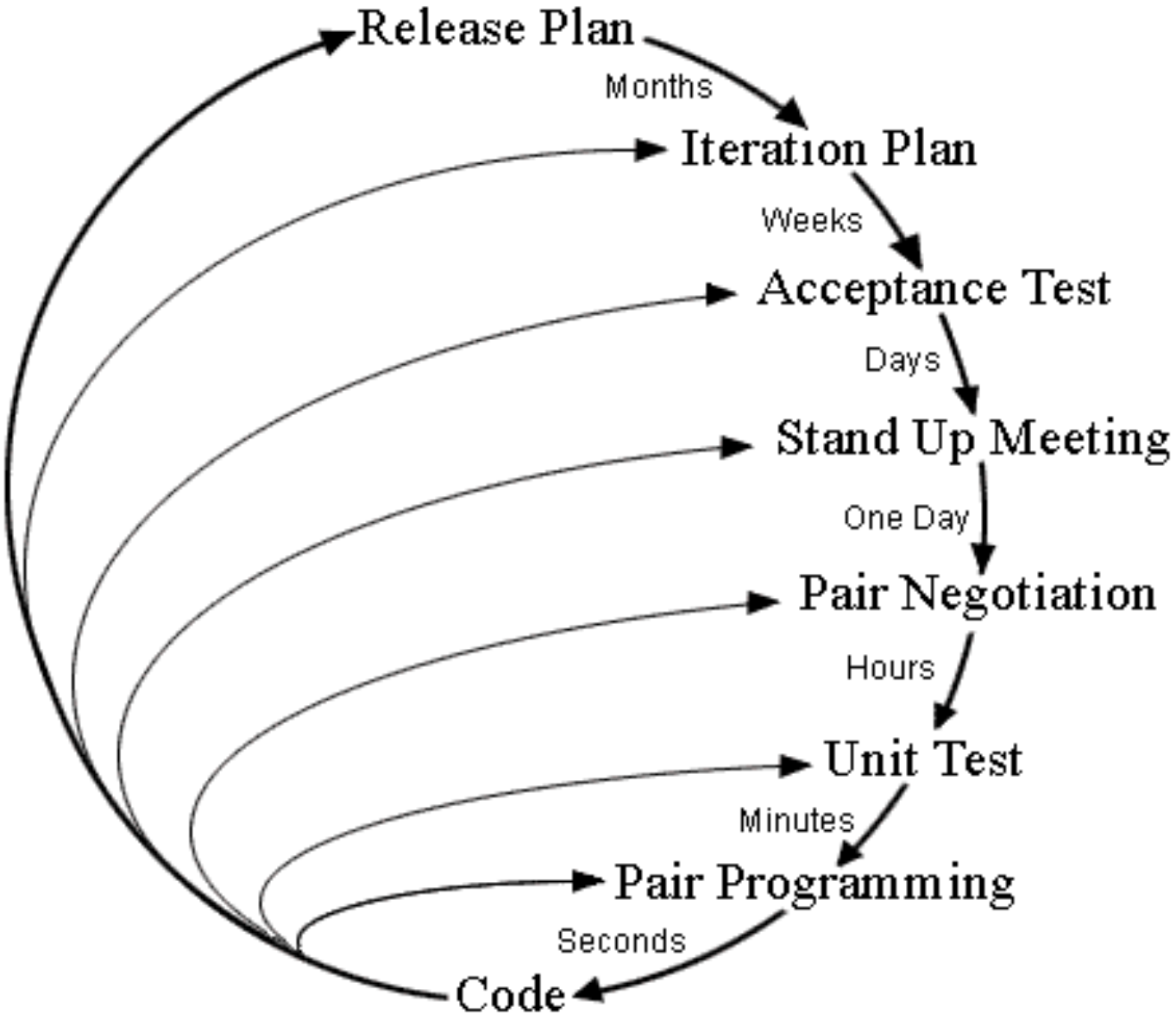
http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

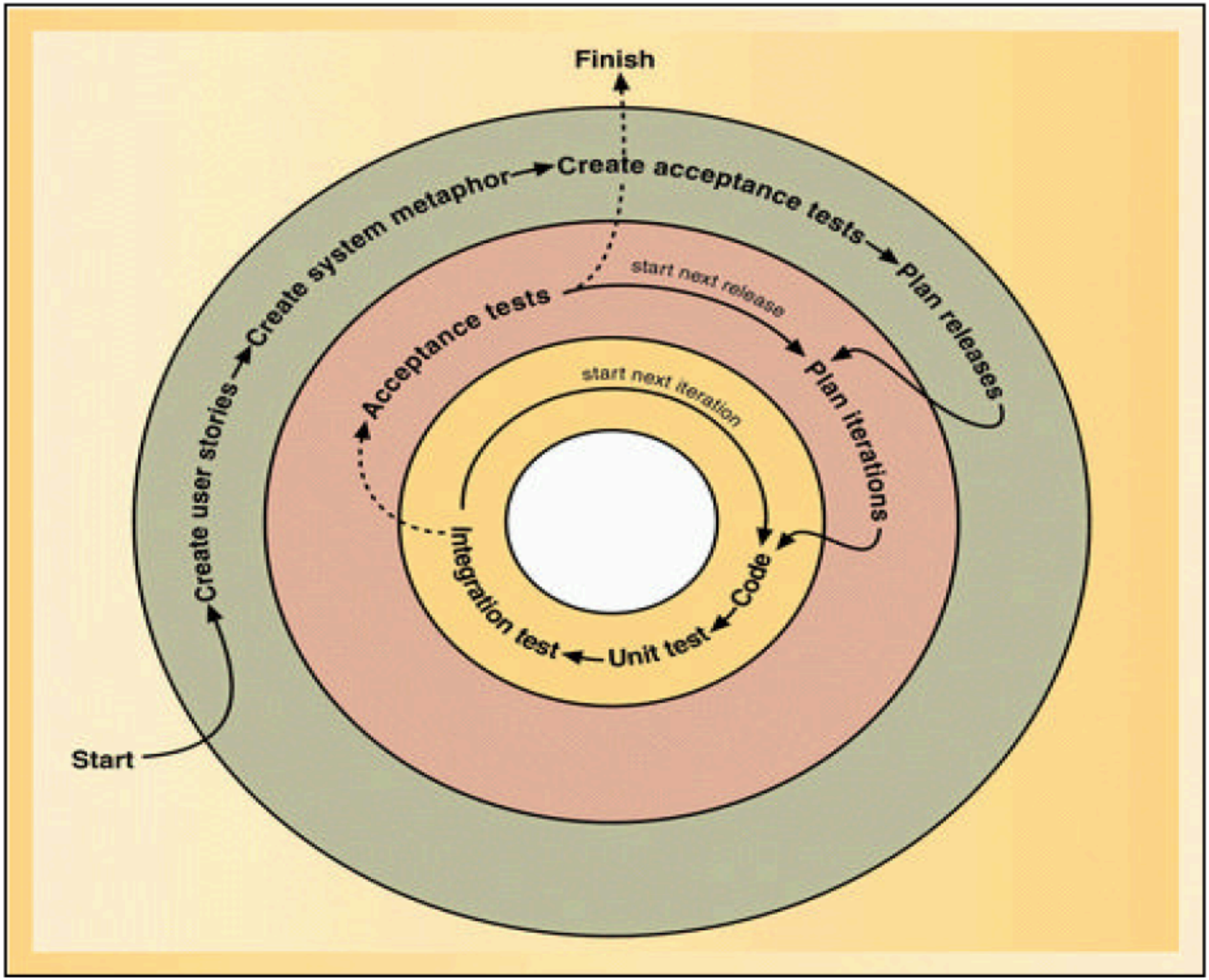
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsent7/html7vxconcodingstandardscodereviews.asp

Il 13º Principio: La riunione in piedi

- Ogni giorno inizia con una riunione di 15 minuti
  - Tutti in piedi (così la riunione dura meno) in cerchio
  - Ciascuno a turno dice:
    - Cosa ha fatto il giorno prima
    - Cosa pensa di fare oggi
    - Quali ostacoli sta incontrando
  - Può essere il momento in cui si formano le coppie

Planning/Feedback Loops




Finish
Create acceptance tests
Plan releases
Create system metaphor
Create user stories
Acceptance tests
start next release
Plan iterations
start next iteration
Integration test
Unit test
Code
Start

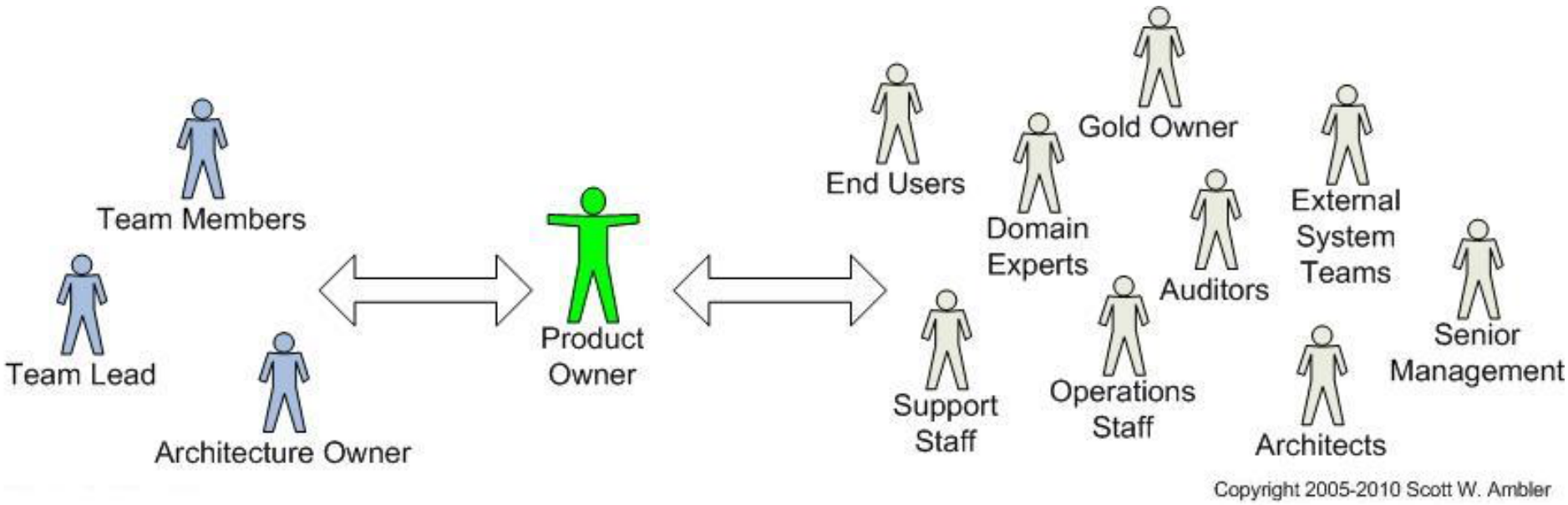
SCRUM

- Ideato da Schwaber e Sutherland e presentato a OOPLSA 1995
- “A flexible, holistic product development strategy where a development team works as a unit to reach a common goal”
- Iterativo e incrementale
- Basato su controllo empirico del processo
- Chiara distinzione dei ruoli: *core* e *ancillary*

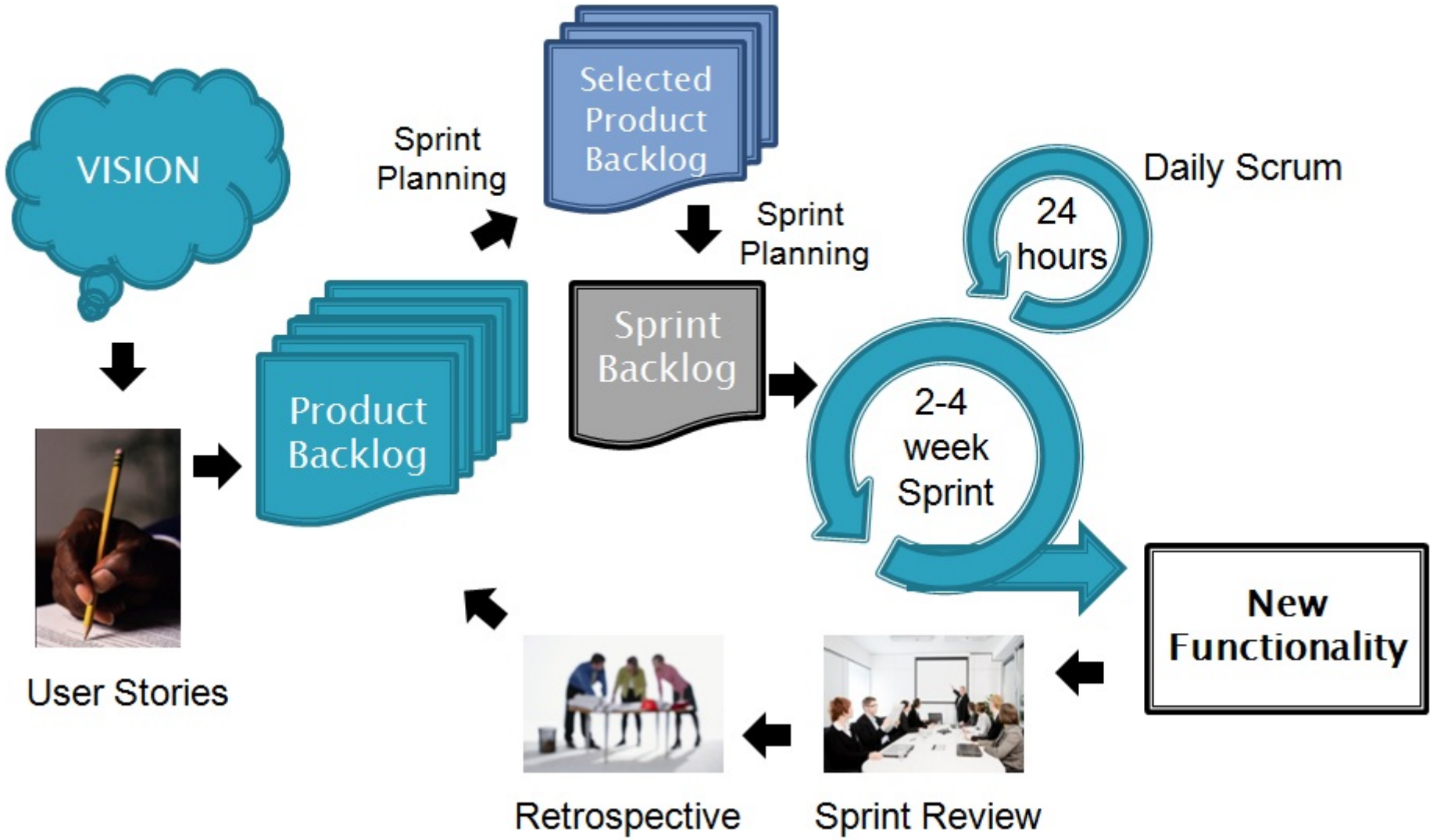
Team Scrum

- Il Team Scrum (core) è formato da:
  - **Product Owner**: rappresenta gli stakeholder (la voce del cliente), scrive il *product backlog*, *user stories*,
  - **Development Team**: 3-9 membri con skill diversi responsabili della consegna di un PSI (Potentially Shippable Increment)
  - **Scrum Master**: facilita la corretta esecuzione del processo Scrum, elimina gli ostacoli
    - Meglio se non è coperto dalla persona con ruolo Product Owner
    - Non ha responsabilità di gestione del personale o di project management “tradizionale”

Product owner



SCRUM Overview



SCRUM Meeting

- **Sprint planning meeting**: cosa fare (*sprint backlog*) e come aggiornare il *product backlog*
  - 8 ore divise in due blocchi da 4
- **Daily scrum o stand-up**: ogni sviluppare dice cosa ha fatto, cosa pianifica per oggi, che impedimenti ha trovato
  - 15 minuti, in piedi
- **Sprint Review**: cosa è stato o non è stato completato in questo sprint, demo
  - 4 ore al massimo
- **Sprint Retrospective**: cosa è andato bene e quali impedimenti sono stati trovati
  - 3 ore al massimo

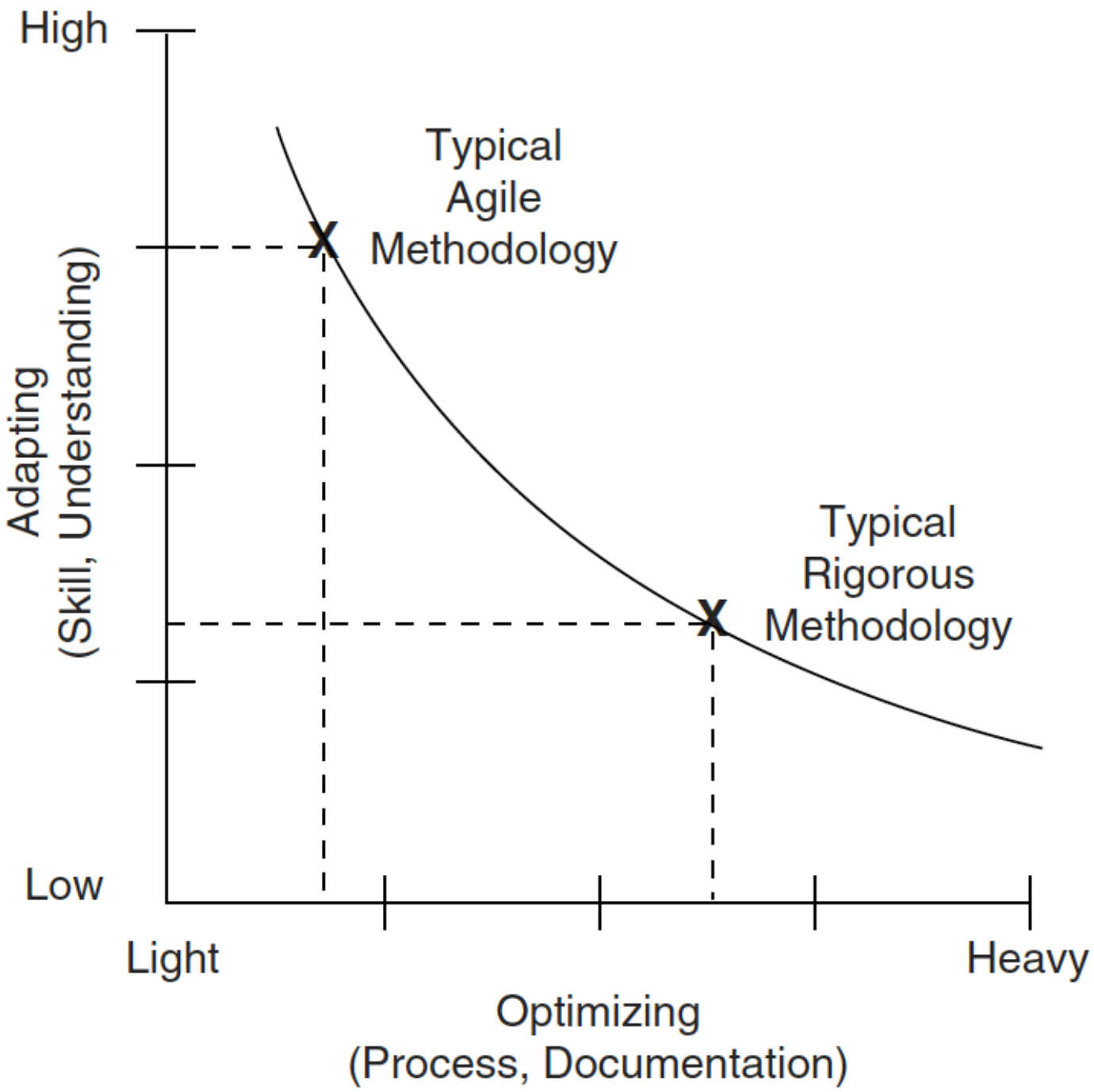
XP o Scrum?

XP	Scrum
Orientato alla qualità (test driven)	Orientato al project management
Iterazione: 1-2 settimane	Sprint: 2-4 settimane
Requisiti sempre modificabili	Req modificabili alla fine dello sprint
Il cliente ordina le storie	Il team ordina le storie
Coaching informale	Scrum master certificato
Buone pratiche tipiche di XP: TDD, Pair programming, planning game, refactoring	Buone pratiche tipiche di Scrum: Retrospettiva post-mortem, uso di strumenti di PM, planning poker

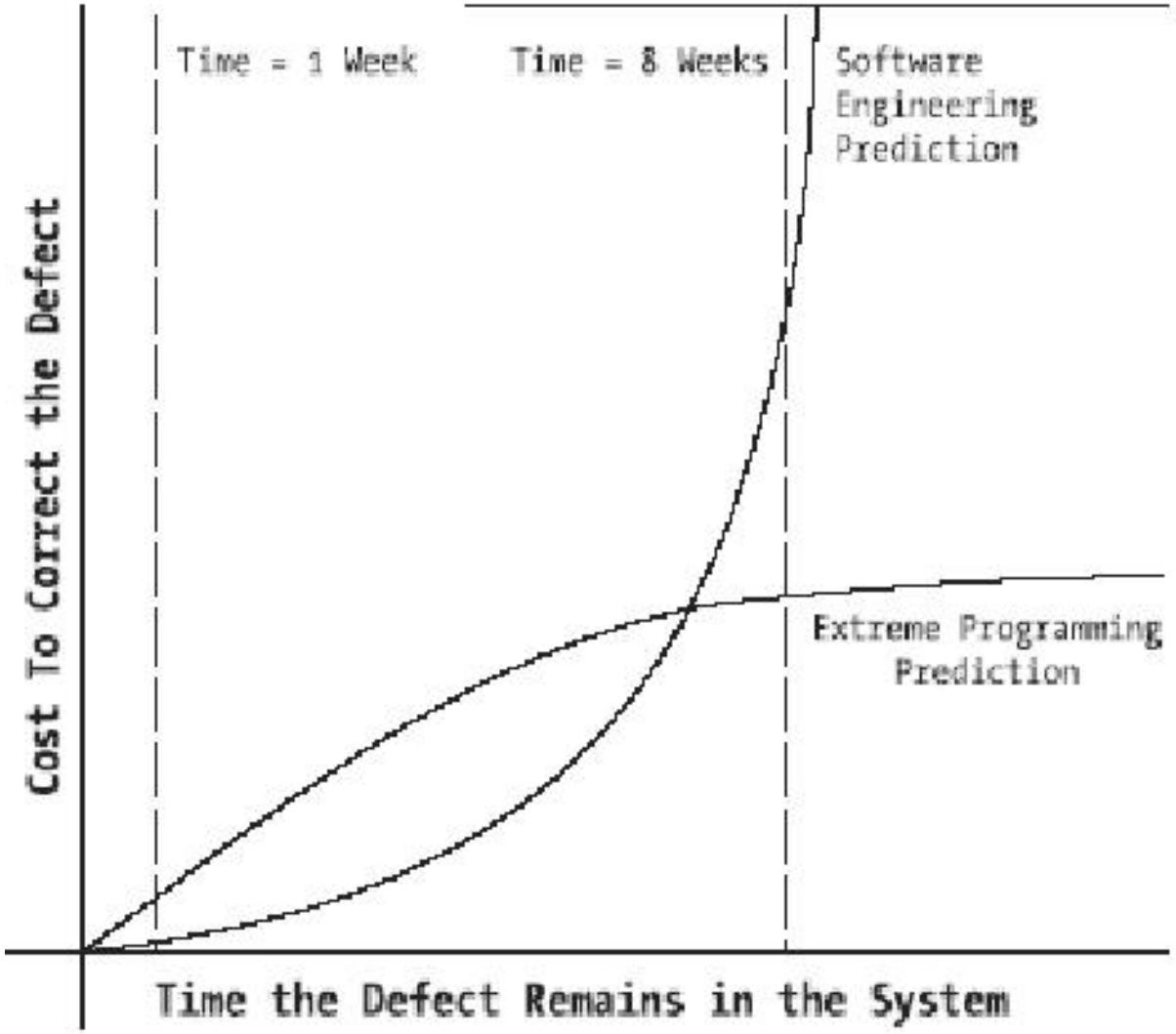
Nota bene: XP e Scrum possono coesistere

Agile o pianificato?

Fattore	Pro-agile	Pro-pianificato
Dimensione	Adatto a team che lavorano su prodotti sw “piccoli”. L’uso di conoscenza tacita limita la scalabilità	Adatto a grandi sistemi e team. Costoso da scalare verso i prodotti sw “piccoli”
Criticità	Utile per applicazioni con requisiti instabili (es. Web)	Utile per gestire sistemi critici e con requisiti stabili
Dinamismo	Refactoring	Pianificazione dettagliata
Personale	Servono esperti dei metodi agili; con Scrum devono essere “certificati”	Servono esperti durante la pianificazione
Cultura	Piace a chi preferisce la libertà di fare	Piace a chi preferisce ruoli e procedure ben definiti


High
Low
Adapting
(Skill, Understanding)
Typical
Agile
Methodology
Typical
Rigorous
Methodology
Light
Heavy
Optimizing
(Process, Documentation)

La vera differenza



fonte: Extreme Programming Explained di Kent Beck

Valutazioni dell'approccio XP

- Dall' industria abbiamo supporto anedottico "forte"
  - "Produciamo sw quasi senza difetti in metà tempo"
- Studi empirici
  - Le coppie producono codice di miglior qualità
    - 15% in più di casi di test superati (ovvero 15% meno errori)
  - Le coppie completano i loro compiti con minor sforzo individuale
    - Sforzo solo 15% in più (non 100%)
  - Molti programmatori all'inizio sono riluttanti a lavorare in coppia
    - Ma poi le coppie apprezzano di più il loro lavoro (92%)
    - Le coppie hanno più fiducia nei loro prodotti (96%)

Cockburn&Williams, The Costs and Benefits of Pair Programming

Rischi del processo XP

- Il codice non viene testato completamente
- Il team produce pochi test utili per il cliente
- Il cliente non aiuta a testare il sistema
- I test “non funzionano” prima dell’integrazione
- I test sono troppo lenti
- Le storie sono troppo complicate
- Il sistema è troppo difettoso
- QA vuole il documento “Specifica dei requisiti”
- Il manager vuole la documentazione di sistema
- Il team è sovraccarico di compiti
- Il team deve affrontare scelte tecniche rischiose
- Alcuni membri (cowboy coders) ignorano il processo del team

Modelli orientati alla qualità

I modelli di processo orientati alla **qualità** si basano sulla misurazione sistematica di alcuni indicatori di qualità di processo

- PSP (personal) e TSP (team sw process)
- Capability Maturity Model (CMM)
- Cleanroom
- ISO 9000

Personal Software Process (PSP)

- Si basa sulla **misurazione delle attività di sviluppo** allo scopo di **migliorare** la **produttività** del singolo sviluppatore
- Alcuni parametri: Size, Effort, Quality, Schedule
- Famiglia di modelli di processo *in-the-small*
- I modelli di processo PSP scalano *in-the-many* (a livello di gruppo di persone) nei modelli Team Software Process (TSP)

www.processdash.com

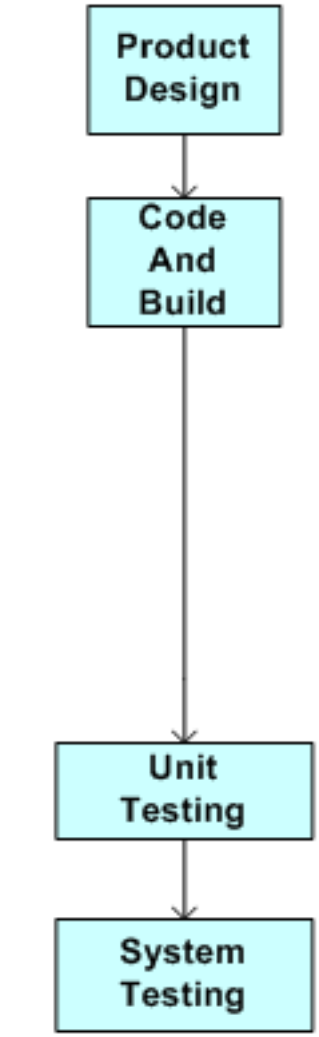
Principi di PSP

- Every engineer is different; to be most effective, engineers must plan their work and they must base their plans on their own personal data.
- To consistently improve their performance, engineers must personally use well-defined and measured processes.
- To produce quality products, engineers must feel personally responsible for the quality of their products. Superior products are not produced by mistake; engineers must strive to do quality work.
- It costs less to find and fix defects earlier in a process than later.
- It is more efficient to prevent defects than to find and fix them.
- The right way is always the fastest and cheapest way to do a job

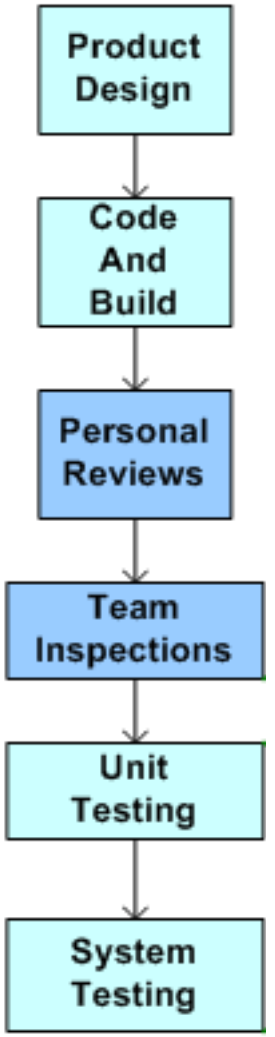
Indicatori di processo: esempi

- Produttività: numero di LOC per unità di tempo
- Riuso: numero di LOC utili riusate in diversi progetti
- Numero di difetti per rilascio
- Numero di difetti rimossi da un rilascio in una unità di tempo

Team Software Process



Today's Typical
SW Quality Process



The TSP
SW Quality Process

www.sei.cmu.edu/tsp/

ISO 9000

- ISO9000-3 è ISO9001 per le fabbriche del sw

ISO 9000	Quality management and quality assurance standards; guidelines for selection and use
ISO 9001	Quality systems model for quality assurance in design, development, production, installation, and servicing
ISO 9002	Quality systems model for quality assurance in production and installation
ISO 9003	Quality systems model for quality assurance in final inspection and test
ISO 9004	Quality management and quality systems elements. Guidelines

Capability Maturity Model for Software (SW-CMM©)

Livello	**Caratteristiche**	**Principali problemi**
Ottimizzante	Migliorare il feedback al processo	Identificare indicatori di processo
Gestito	(Quantitativo) Processo misurato	Raccolta automatica di dati di processo per analizzarlo e modificarlo
Definito	(Qualitativo) Processo definito e istituzionalizzato	Misurazione del processo Analisi del processo Piani di qualità quantitativi
Ripetibile	(Intuitivo) Processo dipendente dagli individui	Creare un Gruppo di Processo Identificare un'architettura di processo Introdurri metodi e strumenti di Sw Eng
Iniziale	Ad hoc/ Caotico No stima dei costi, pianificazione, o gestione	Project management Pianificazione del progetto Controllo di qualità del software

http://www.sei.cmu.edu/cmm/

Lo sviluppo del software Open Source

- Ridistribuzione libera del sorgente
- Codice sorgente sempre incluso nella distribuzione
- Altri software derivabili con licenza
- Protezione dell'integrità del sorgente dell'autore
- Nessuna discriminazione contro persone o gruppi
- Nessuna discriminazione contro campi applicativi
- Distribuzione della licenza, che si applica a tutti
- Licenza non specifica di prodotto
- Licenza che non vincola altri prodotti sw
- Licenza tecnologicamente neutrale

`www.opensource.org/docs/definition.php`

La filosofia Open Source

- Ogni buon prodotto sw inizia da un problema personale
- I bravi programmatori sanno cosa scrivere. I migliori sanno cosa riscrivere
- Quando hai perso interesse in un programma che hai costruito, è tuo dovere passare le consegne ad un successore competente
- Trattare gli utenti come sviluppatori è la strada migliore per debugging efficace e rapidi miglioramenti del codice
- Distribuisci presto e spesso; e presta ascolto agli utenti
- Stabilita una base di betatester e cosviluppatori sufficientemente ampia, ogni problema verrà presto individuato e qualcuno troverà la soluzione adeguata

E. Raymond, The Cathedral and the Bazaar, O'Reilly, 1997

Famosi Progetti Open Source

- Linux, OpenBSD, FreeBSD, NetBSD
- Progetto Apache. HTTP server, ANT, Tomcat
- PHP, Perl
- Progetto GNU. GCC, GDB, EMACS, GNOME
- JBOSS
- Vim
- Postfix, SpamAssassin, Clam AntiVirus
- …e molti altri

Alcuni repository di prodotti software openvsource:

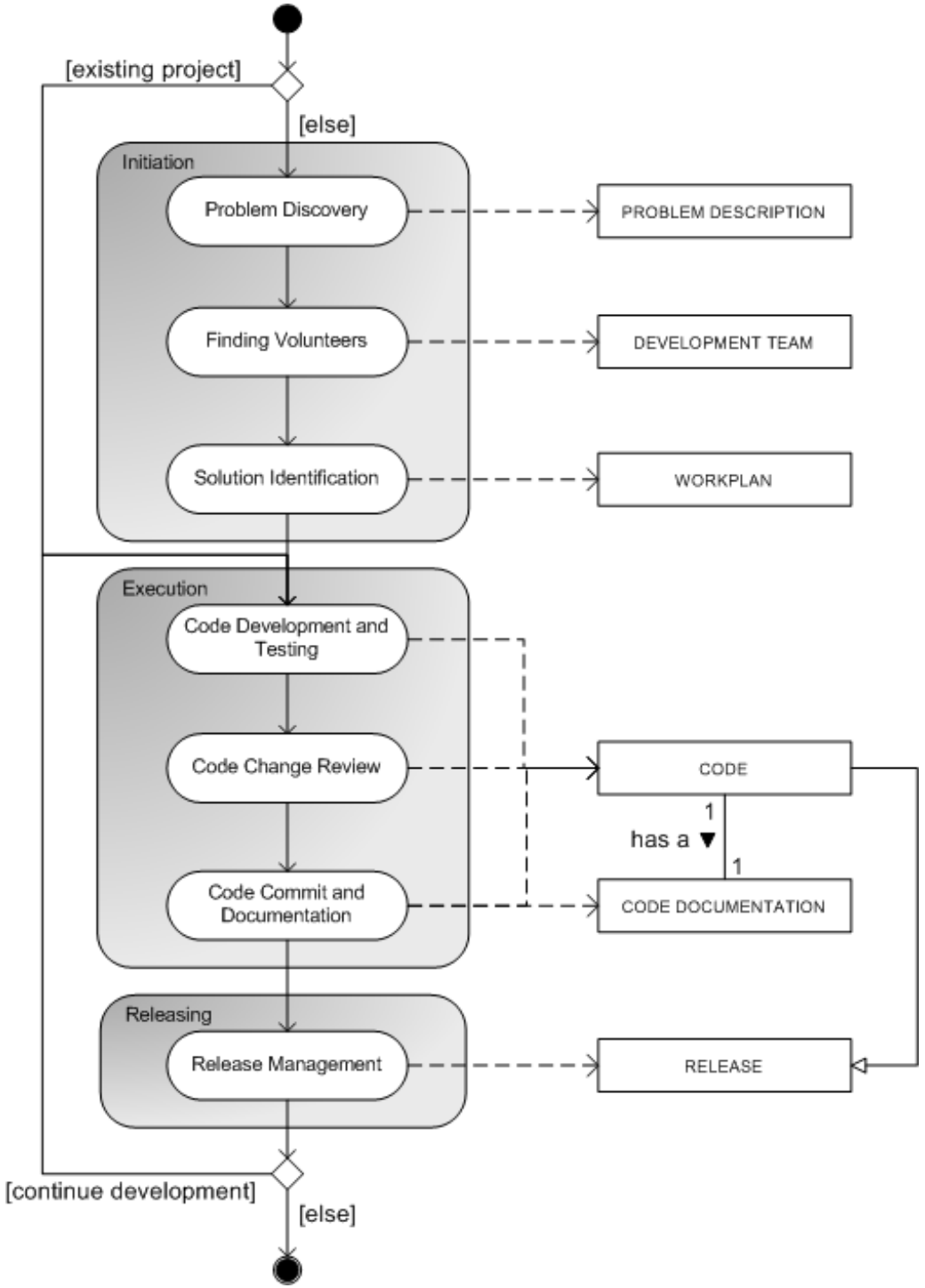
```
https://github.com/
http://sourceforge.net/
```

Il processo di sviluppo del sw open source

- Definizione del problema (problem discovery)
- Ricerca di volontari
- Identificazione della soluzione
- Sviluppo del codice e testing
- Code change review
- Code commit and documentation
- Release management

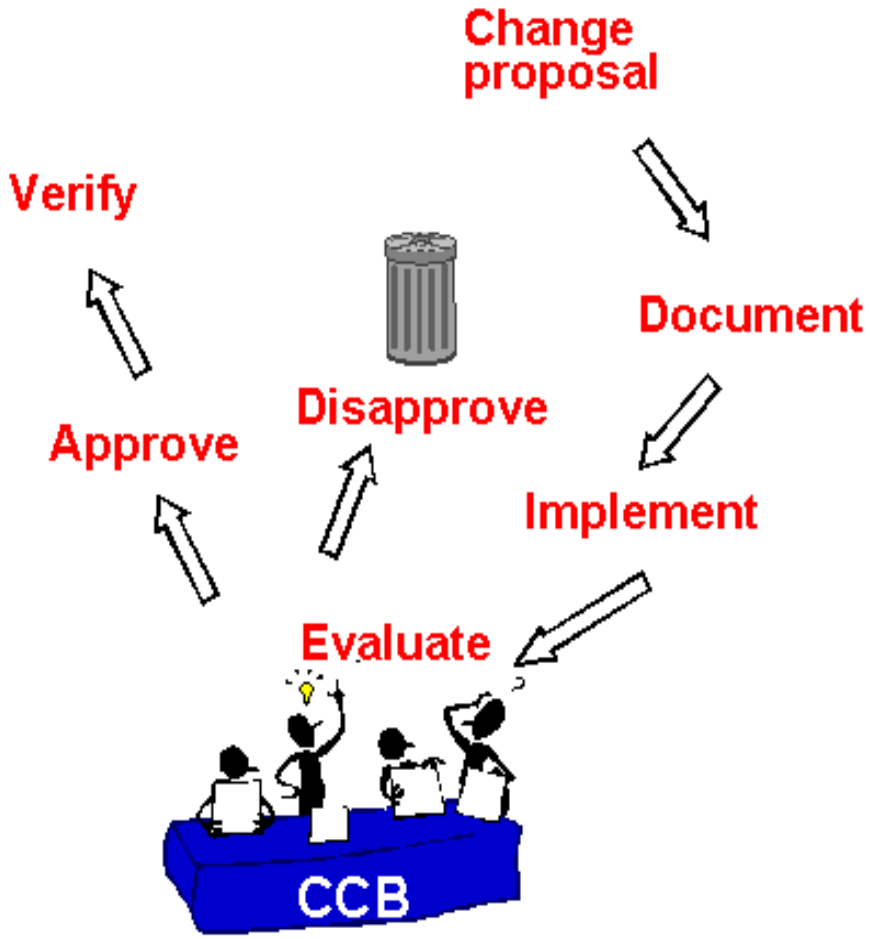
A Framework for creating hybrid-open source software communities. Srinarayan Sharma et. al. Info Systems (2002)

Processo open source



Change control board per OSS

- Il processo è ”pubblico”
- Le implementazioni sono controllate da un board (CCB) che revisiona e testa il codice proposto
- modifiche moderate
- build frequenti
- proprietà collettiva
- ”no maintenance”



Change process for OSS

Reputation-driven

- I sistemi OS sono quasi sempre costuiti da piccoli gruppi di sviluppatori volontari
- Siccome non sono pagati, qual è l'incentivo?
- "*The 'utility function' Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers*."
  Raymond, La cattedrale e il bazaar, 1999

Conclusioni: modelli pianificati

- Miglioramento del processo
- Maturità dell'organizzazione
- Specialisti di qualità del processo
- Gestione dei rischi
- Verifica separata dalla validazione
- Uso sistematico dell'architettura

Conclusioni: modelli agili

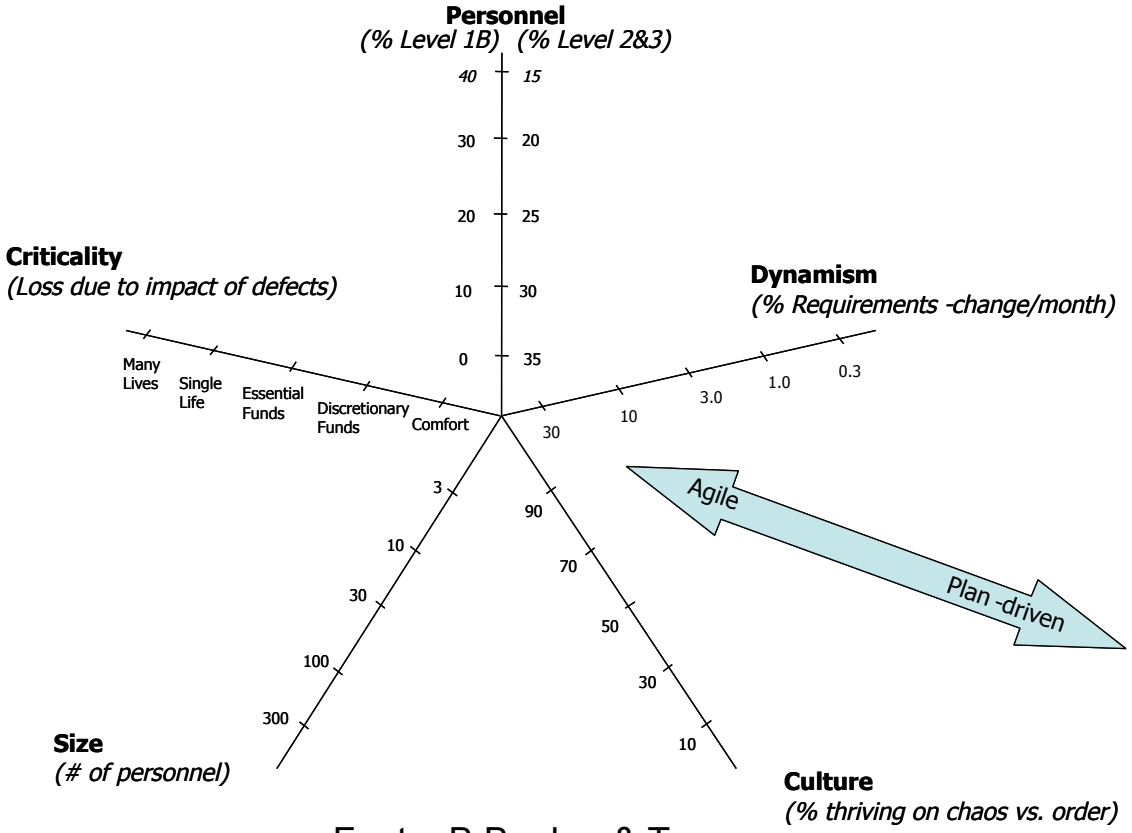
- Accettano il cambiamento
- Sviluppo guidato dai test
- Delivery rapide e frequenti
- Progetto semplice
- Refactoring
- Analisi retrospettiva
- Conoscenze tacite

Confrontare i modelli

CMM	RUP	XP
Cos'è importante?		
Stabilità	Adattabilità	Flessibilità
Ripetibilità	Precisione	Agilità
Perché standardizzare?		
Predicibilità	Componentistica	Velocità
Efficienza	Integrazione	Semplicità
La qualità come viene fuori?		
Rimozione difetti	API predefinite	Integrazione continua
Prevenzione difetti	Integrazione semplice	Presenza cliente
Progetto per qualità	Gestione dei rischi	

Agile vs pianificati: fattori per la scelta

Dimensione, Criticalità, Dinamismo, Personale, Cultura

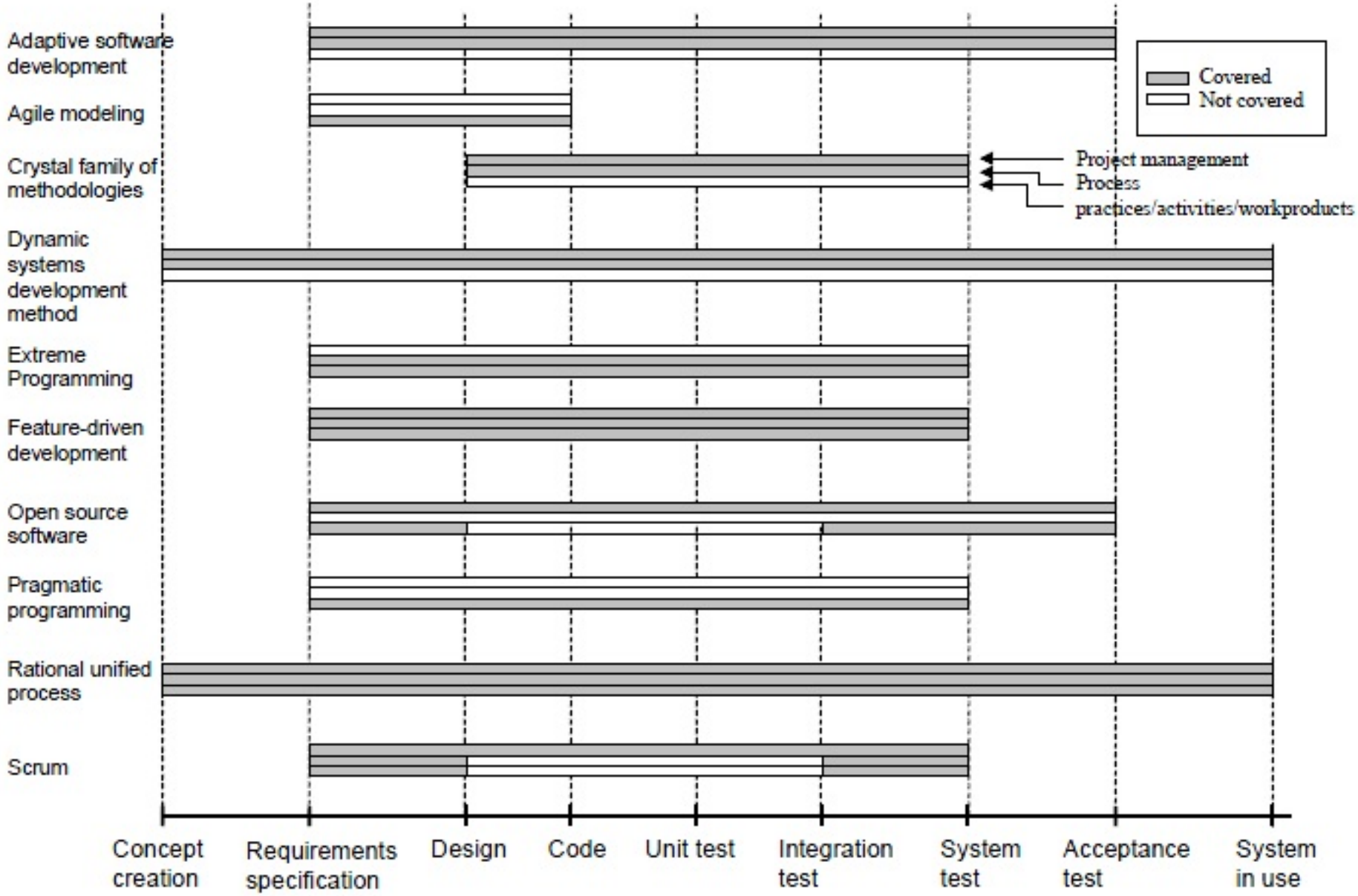


Fonte: B.Boehm & Turner

Livelli di personale (Boehm e Turner)

Livello	Criterio
-1	Non collaborativo
1B	Buon lavoratore poca esperienza necessita di guida
1A	Buon lavoratore esperienza limitata accetta guida
2	Funziona bene in piccoli team in progetti con precedenti
3	Funziona bene in grandi team in progetti senza precedenti

Confrontare i modelli



Conclusioni

- No “Silver bullets” nel processo di sviluppo
- Processi waterfall: pianificati, rigidi
- Processi iterativi: pianificati, flessibili
- Processi agili: non pianificati, test-driven
- Processi per la qualità: misurati
- Processi open-source: reputation-driven

- Ogni organizzazione dovrebbe esplorare più modelli e scegliere quelli più efficaci

Letture raccomandate

- Bohem e Turner, Using Risk to Balance Agile and Plan-Driven Methods, *IEEE Computer*, June 2003
- E. Raymond, La cattedrale e il bazaar, 1997 http://it.wikisource.org/wiki/La_cattedrale_e_il_bazaar

Riferimenti

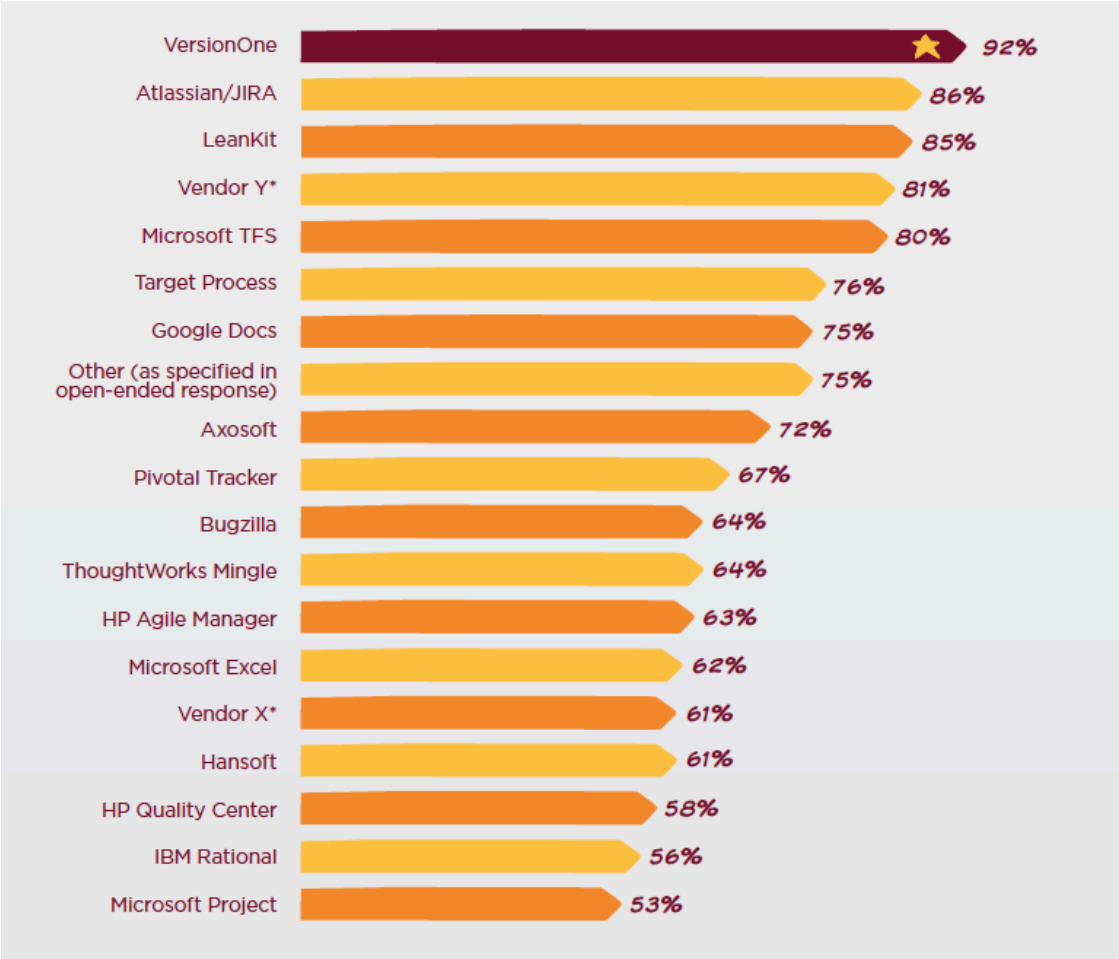
- Capitolo 9 del SWEBOK: “Software engineering process”
- Highsmith, *Agile Software Development Ecosystems*, AW 2003
- Larman, *Agile and Iterative Development: A Manager's Guide*, 2003
- Cockburn & Williams, The costs and benefits of pair programming, 2000
- Beck e altri, Manifesto for agile development, 2001 `agilemanifesto.org`

Siti

- Extreme Programming www.extremeprogramming.org
- Scrum www.scrum.org

Strumenti

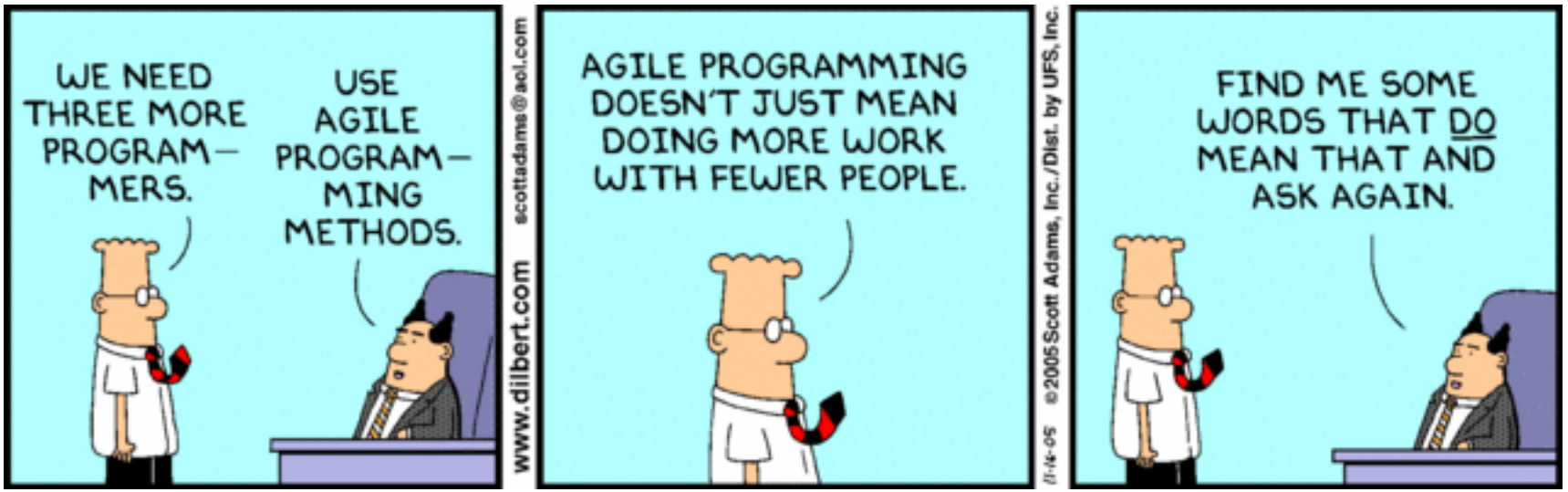
- Ant, XDoclet, JUnit, Cactus, Maven
- www.atlassian.com/software/jira
- agiletrack.net
- www.planningpoker.com
- www.versionone.com/product/agile-collaboration-tools/



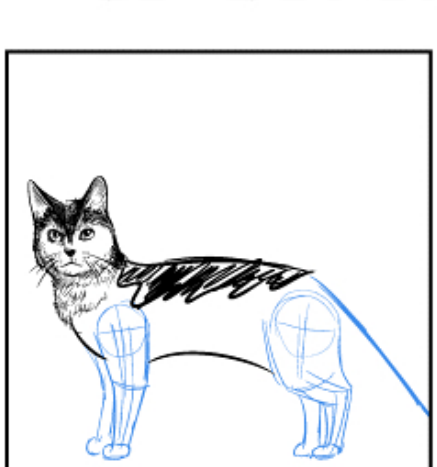
Pubblicazioni di ricerca

- International Conference on Software and System Process
- Journal of Software Maintenance and Evolution: research and practice

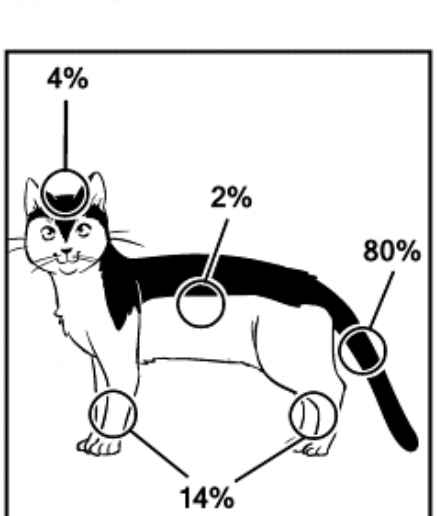
Domande?


WE NEED THREE MORE PROGRAM-MERS.
USE AGILE PROGRAM-MING METHODS.
www.dilbert.com scottadams@aol.com
AGILE PROGRAMMING DOESN'T JUST MEAN DOING MORE WORK WITH FEWER PEOPLE.
11-4-05 ©2005 Scott Adams, Inc./Dist. by UFS, Inc.
FIND ME SOME WORDS THAT DO MEAN THAT AND ASK AGAIN.

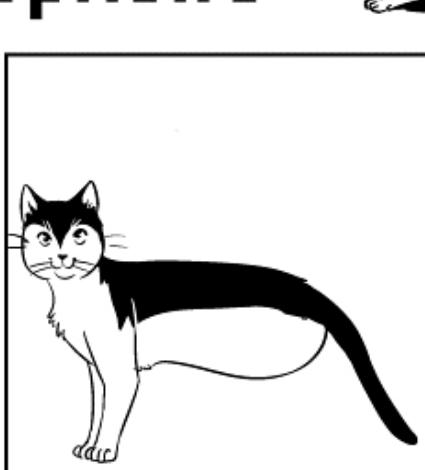
Richard's guide to software development



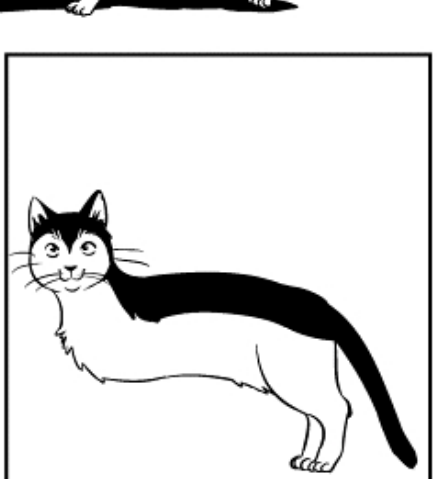
HOW THE SOFTWARE IS DESIGNED.



HOW MUCH TIME HAS TO BE SPENT ON EACH PART.



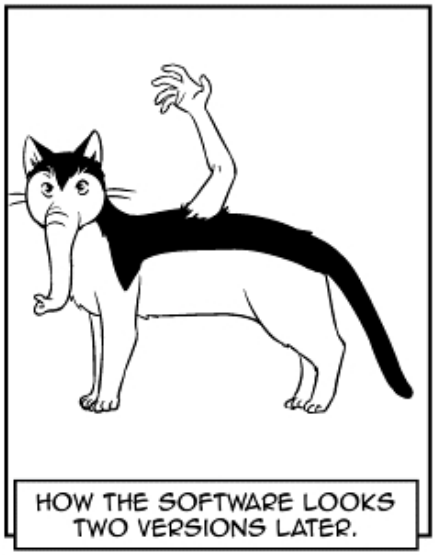
HOW THE SOFTWARE LOOKS BEFORE THE BETA TEST.



HOW THE SOFTWARE LOOKS AFTER THE BETA TEST.

HOW THE SOFTWARE IS ADVERTISED.

WHAT THE CUSTOMER REALLY WANTED.



HOW THE SOFTWARE LOOKS TWO VERSIONS LATER.