



UNIVERSITÀ DEGLI STUDI
DI MILANO

Ingegneria del software

Lezione 4 eXtreme Programming (cont)

8. Proprietà collettiva

Egoless Programming (Weinberg 1971)

- Il codice non appartiene ad una persona sola
- Il codice non è senza proprietà dove tutti possono modificare tutto *senza preoccupazioni*
- Ci si deve sentire responsabilizzati sull'intero codice anche se non si conosce tutto alla stessa maniera

9. Integrazione continua

- Nell'ottica di avere feedback rapidi, l'integrazione va fatta più volte al giorno (cioè non solo test di unità ma anche di integrazione).
- La coppia quindi dopo avere risolto il suo problema in piccolo è **responsabile di risolvere** anche i problemi di integrazione.
- Azione rapida (molto più veloce dello sviluppo della feature) perché nel frattempo il sistema “non è cambiato molto”

10. Settimana di 40 ore

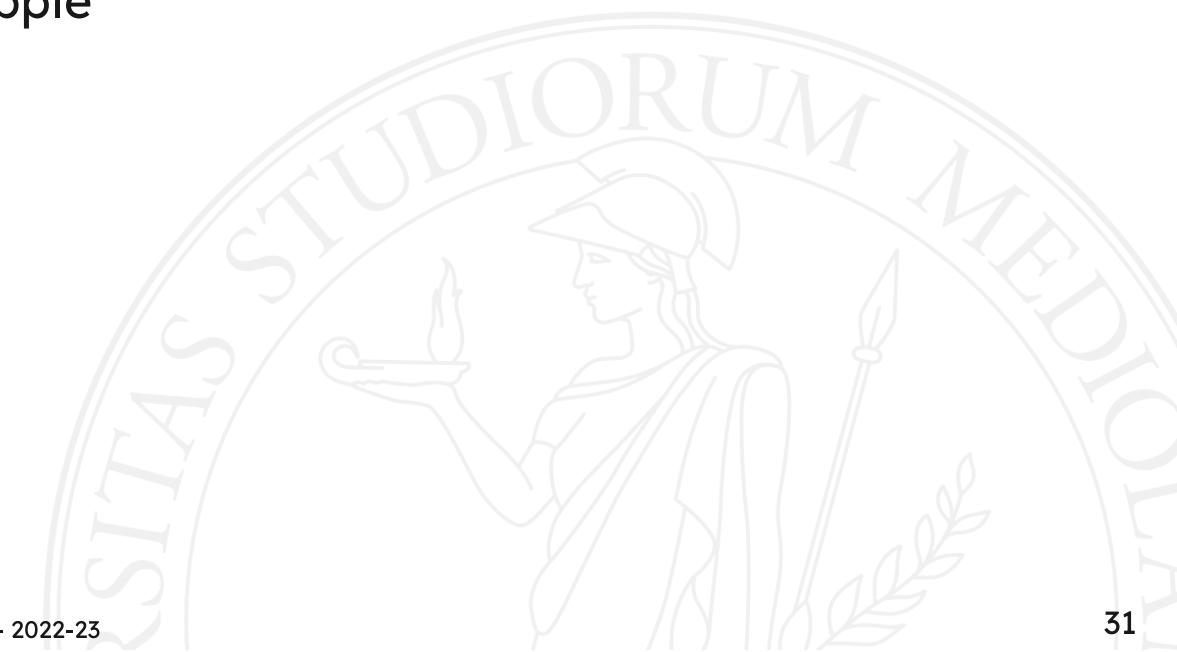
- Può sembrare strana una osservazione di questo genere, non tecnologica.
 - Problema di freschezza
 - soddisfazione di lavorare nel team
 - meno problemi familiari
 - minor probabilità di perdere dipendenti e *knowhow*

11. Cliente sul posto

- Visto spesso dai committenti come un problema: si toglie loro una risorsa
- Permette una fase di specifica leggera
- Fornisce in tempo reale scelte, test funzionali, etc.
- ma è rappresentativo di tutti gli stakeholder?

12. Standard di codifica

- Enfatizza la comunicazione attraverso il codice
- Aiuta
 - refactoring
 - programmazione a coppie
 - proprietà collettiva



Raggruppiamo per fasi

- Requirements:
 - Gli utenti fanno parte del (lavorano con il) team di sviluppo
 - Consegni incrementali e pianificazioni continue
- Design:
 - Una metafora come visione unificante di un progetto
 - Refactoring
 - Presumere la semplicità
- Code:
 - programmazione a coppie
 - proprietà collettiva
 - Integrazioni continue
 - standard di codifica
- Test:
 - testing di unità continuo (da scriversi prima del codice)
 - test funzionale scritto dagli utenti

E la documentazione?

- Non abbiamo mai parlato di documentazione in senso stretto:
 - delle schede di usi (scritte dagli utenti)
 - use stories (due o tre frasi in linguaggio naturale)
- Dove è allora la documentazione?
 - nel codice
 - gli standard di codifica permettono una maggiore leggibilità
 - ma soprattutto nei test di unità
- nelle persone
 - il cliente
 - il compagno di pair programming

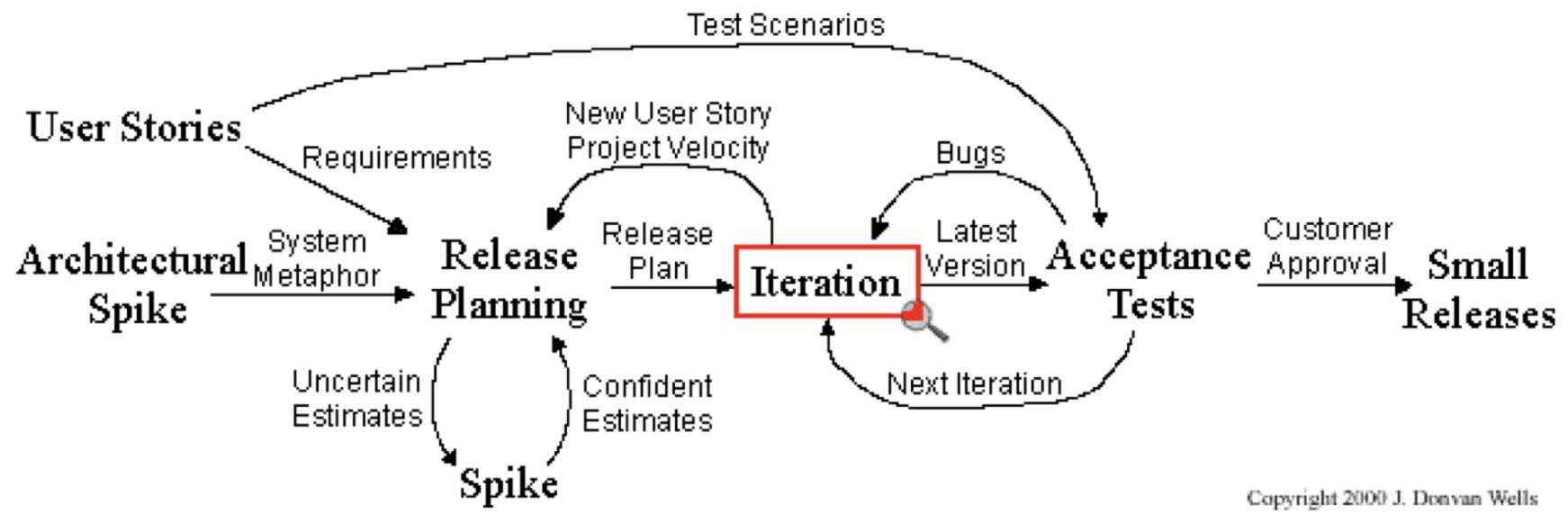
Quando non si può usare l'XP

- Ambienti che proibiscono l'uso anche solo di uno degli approcci usati
 - barriere tecnologiche che impediscono di testare ed avere un feedback in breve tempo
 - niente rapido feedback
 - barriere di tipo manageriale o burocratico
 - team troppo numerosi (numero ideale sulla decina)
 - necessità di documentazione per certificazioni basate su di essa
 - barriere di tipo fisico logistico
 - sistemazione degli spazi di lavoro che non permettono flessibilità nella formazione delle coppie di lavoro

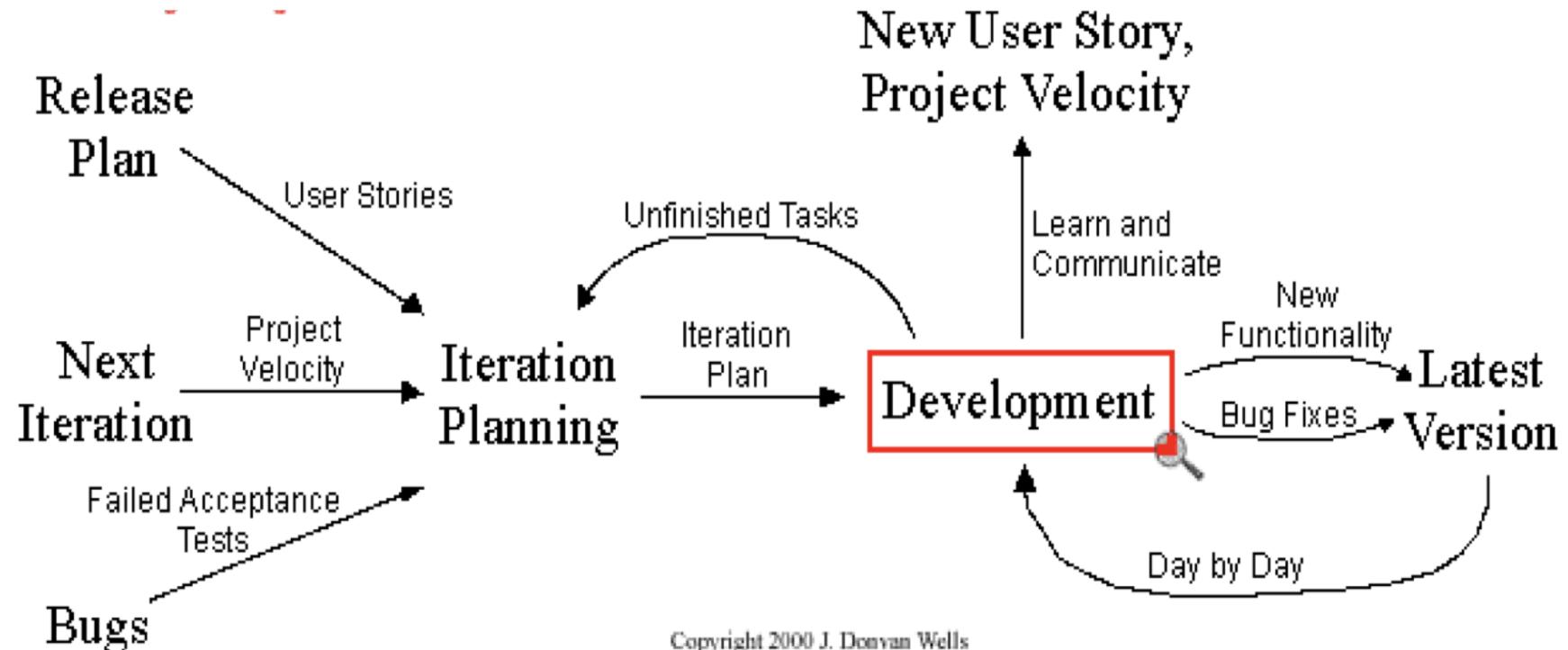
Alcune critiche

- Sottovalutazione up-front
- Sopravvalutazione User stories
- Mancata evidenziazione dipendenze tra user stories
- TDD può portare a visione troppo ristretta
- Cross functional teams

Se volessimo disporre in un grafo?

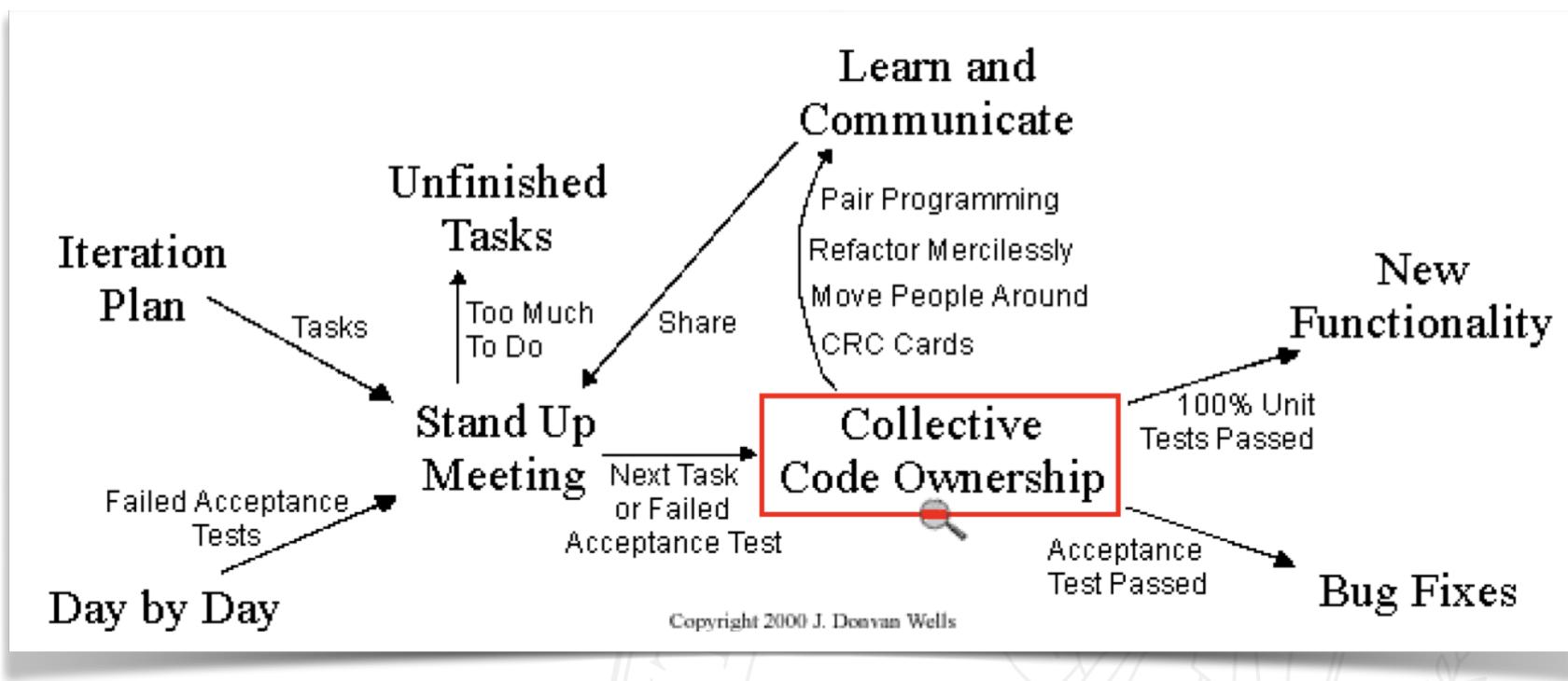


Se volessimo disporre in un grafo?

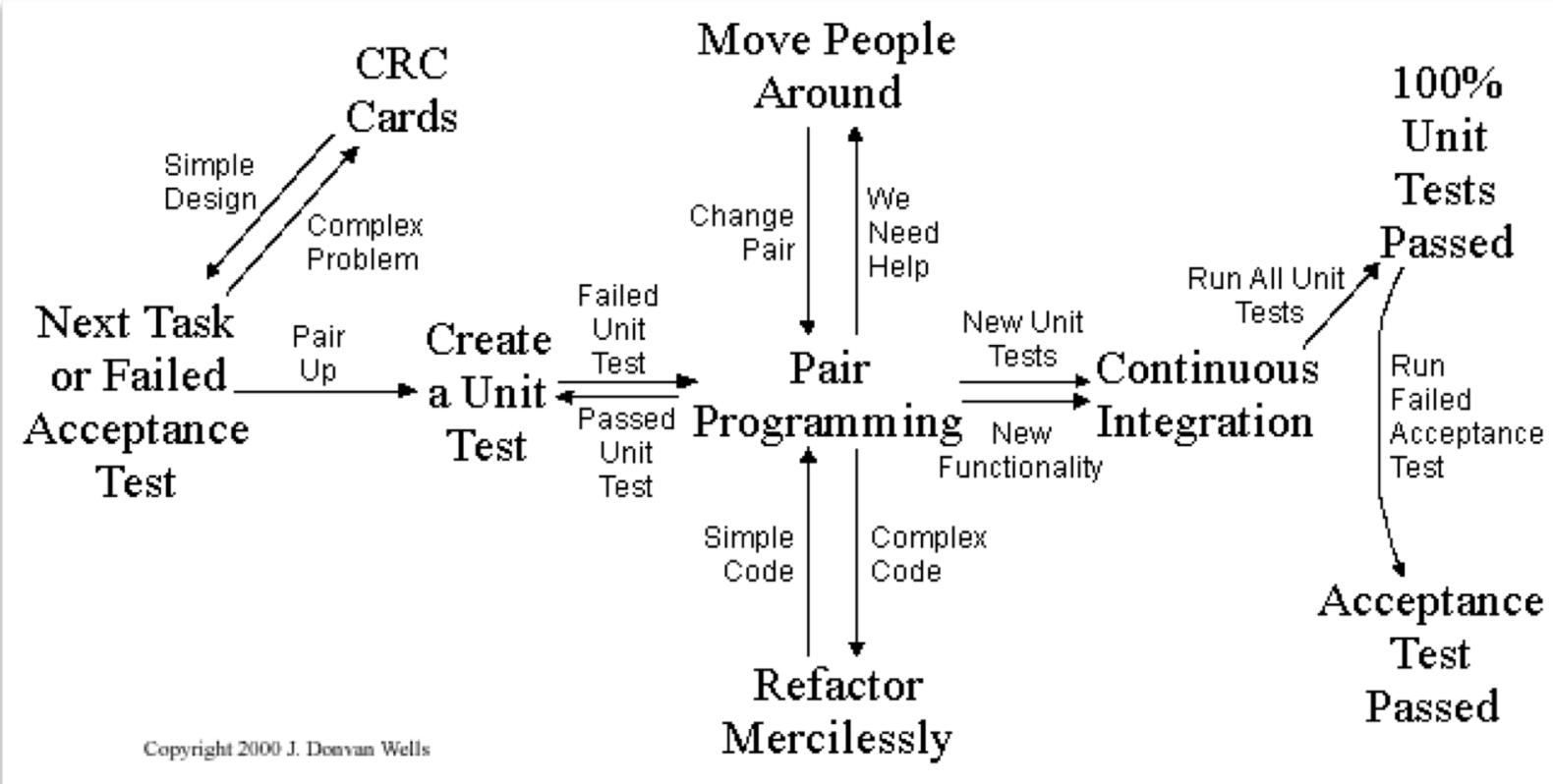


Copyright 2000 J. Donvan Wells

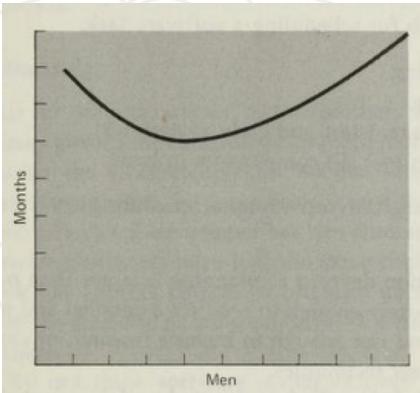
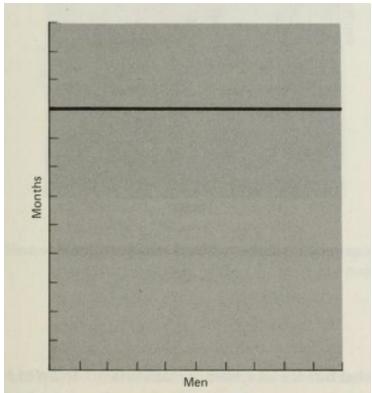
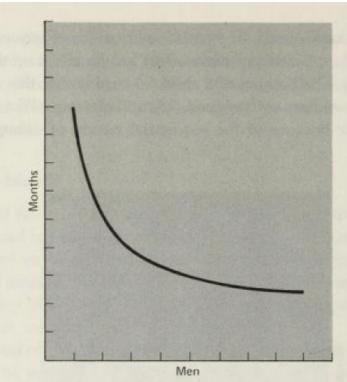
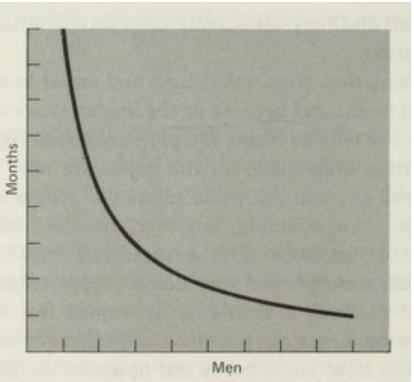
Se volessimo disporre in un grafo?



Se volessimo disporre in un grafo?



Mesi/uomo: Produttività, stime, costi, velocità



Materiale principale usato fino ad adesso

- Introduzione a ingegneria del software
 - [Ghezzi] Cap. 1 - Ingegneria del software: visione d'insieme
 - Qualità del software
 - [Ghezzi] Cap. 2 - Il software: natura e qualità
 - Modelli di ciclo di vita del software
 - [Ghezzi] Cap. 7 - Processo di produzione del software
 - fino a 7.4
 - [Bossavit]
 - Cap. 7 Who's afraid of the Big Bad Waterfall
 - Cap. 10 The cost of defects: an illustrated history
 - Principi della Ingegneria del sw
 - [Ghezzi] Cap. 3 - Principi dell'ingegneria del software
 - Agile e eXtreme Programming
 - [<http://www.extremeprogramming.org/rules.html>] per una presentazione amica
 - [Meyer] le varie pratiche, per una vista più "*critica*"
- [Ghezzi] Fondamenti di ingegneria del sw
 - [Meyer] Agile! The good, the hype, and the ugly
 - [Bossavit] The Leprechauns of sw engineering
- [Endres] A Handbook of Software and Systems Engineering
 - le *leggi* citate
 - link segnalati nelle slides di articoli e white papers
- [Meyer]
 - Cap. 2.2 The Top Seven Rhetorical Traps
 - [Bossavit]
 - Cap. 3 Why you should care about empirical results
 - Cap. 4 The messy workings of scientific discourse



UNIVERSITÀ DEGLI STUDI
DI MILANO

Open Source Process

Alcuni articoli

- **La cattedrale ed il bazaar** di *Eric Raymond*
 - <http://www.catb.org/~esr/writings/cathedral-bazaar/>
- **The Care and Feeding of FOSS** di *Craig A. James*
 - <https://web.archive.org/web/20081015010505/http://www.moonviewscientific.com/essays/carefeeding.html>
- **The Emerging Economic Paradigm Of Open Source** by *Bruce Perens*
 - <http://web.archive.org/web/20120724095330/http://perens.com/works/articles/EconomicParadigm.html>
- **An empirical study of open-source and closed-source software products** by *J.W. Paulson et al.*
 - <https://ieeexplore.ieee.org/abstract/document/1274044>

Cattedrale e bazaar

1. “Ogni buon lavoro software inizia dalla frenesia personale di uno sviluppatore”
[Raymond]
2. Chiede ad amici o colleghi cosa sanno sull’argomento. Alcuni hanno lo stesso problema o problemi simili, ma nessuno ha una soluzione.
3. Le persone interessate cominciano a scambiarsi pareri e conoscenze sull’argomento
4. Le persone interessate che intendono spendere delle risorse (a questo livello si tratta del proprio tempo libero) sulla soluzione del problema danno il via ad un progetto informale.
5. I membri del progetto lavorano al problema fino a che non raggiungono dei risultati presentabili.
6. Si rende noto il lavoro svolto dal gruppo e arrivano i primi suggerimenti esterni al gruppo
7. Interazione tra vecchi e nuovi membri del gruppo
8. Nuove informazioni vengono acquisite e si ritorna al punto 5

Alcune frasi dall'articolo

- “Se dai a tutti il codice sorgente, ognuno di essi diventa un tuo ingegnere”
- “Se ci sono abbastanza occhi [che cercano errori], gli errori diventano di poco conto”
- “Se tratti i tuoi beta-tester come se fossero la tua risorsa più importante, essi risponderanno diventando la tua risorsa più importante”
- “Quando hai perso interesse in un programma, l’ultimo tuo dovere è passarlo a un successore competente”

Confronto modelli

	Traditional	Agile	Open source
Documentation	Documentation is emphasized as a means of quality control and as a management tool.	Documentation is deemphasized.	All development artifacts are globally available, including code and information documentation.
Requirements	Business analysts translate users' needs into software requirements.	Users are part of the team.	The developers typically are the users.
Staffing model	Developers are assigned to a single project.	Developers are assigned to a single project.	Developers typically work on multiple projects at different levels of involvement.
Peer review	Peer review is widely accepted but rarely practiced.	Pair programming institutionalizes some peer review.	Peer review is a necessity and is practiced almost universally.
Release schedules	Large number of requirements bundled into fewer, infrequent releases.	Release early, release often.	Hierarchy of release types: "nightly," "development," and "stable."
Management	Teams are managed from above.	Teams are self-organized.	Individual contributors set their own paths.
Testing	Testing is handled by QA staff, following development activities.	Testing is part of development.	Testing and QA can be performed by all developers.
Distribution of work	Different parts of the codebase are assigned to different people.	Anyone can modify any part of the codebase.	Anyone can modify any part of the codebase, but only committers can make changes official.

Source: Forrester Research, Inc.

Care and Feeding of FOSS

1. **Invention:** Somebody has an idea, and makes it work
2. **Expansion and Innovation:** The world takes notice, and the technology begins to expand rapidly
3. **Consolidation:** A few projects edge into the lead, and their leads quickly cascade into dominance. The other projects are absorbed or go out of business
4. **Maturity:** The market is reduced to a handful of products. Innovation slows to a moderate pace. It is difficult or impossible for new suppliers to get into the market
5. **FOSS Domination:** With the slow pace of innovation of the Maturity phase, the FOSS community begins to slowly but inexorably erode the technical lead held by the commercial offerings
6. **The FOSS era:** In the end, the FOSS version dominates

The Emerging Economic Paradigm Of Open Source

- Il software, per una azienda, non è sempre un prodotto, ma piuttosto una **tecnologia abilitante** essenziale
- Dal punto di vista economico è importante distinguere i casi di **tecnologia differenziante o non differenziante**
 - il cliente si accorge degli effetti del software?
 - i competitor hanno accesso allo stesso software?

Differenziante
qualcosa che ci dà un vantaggio competitivo

Confronto

Paradigm	Efficiency	Failure Rate	Distributes Cost	Distributes Risk	Protects Customer Diff.	Protects Vendor Diff.	Required Market Size
Retail	less than 10%	50%	Late, sometime after sales start	No	No.	Yes.	Mass market
In-House and Contract	60% to 80%	50%	No	No	Yes.	Maybe.	1
Consortium and Non-Open-Source Collaboration	60% to 80%	~90%, too high.	Yes.	Yes	Maybe	Maybe	5 and up.
Open Source	60% to 100%	50%	Early, during development	Yes	No.	No.	5 and up.