

La produzione del software



Corso di Ingegneria del Software
CdL Informatica Università di Bologna

Obiettivi di questa lezione

- Fare software
- La nozione di **prodotto** software
- Varietà di tipi di prodotti software
- L'industria del software
- Lo sviluppo del software

Discussione

- Cos'è il software?
- Chi crea il software?
- Cos'è un prodotto sw? Cos'è un servizio sw?
- Esistono tipi diversi di software?



Alan Turing

sugli ingegneri del software



Roughly speaking those who work in connection with the Automatic Computing Engine will be divided into its masters and its servants. Its masters will plan out instruction tables for it, thinking up deeper and deeper ways of using it. Its servants will feed it with cards as it calls for them. They will put right any parts that go wrong...As time goes on the calculator itself will take over the functions both of masters and of servants...

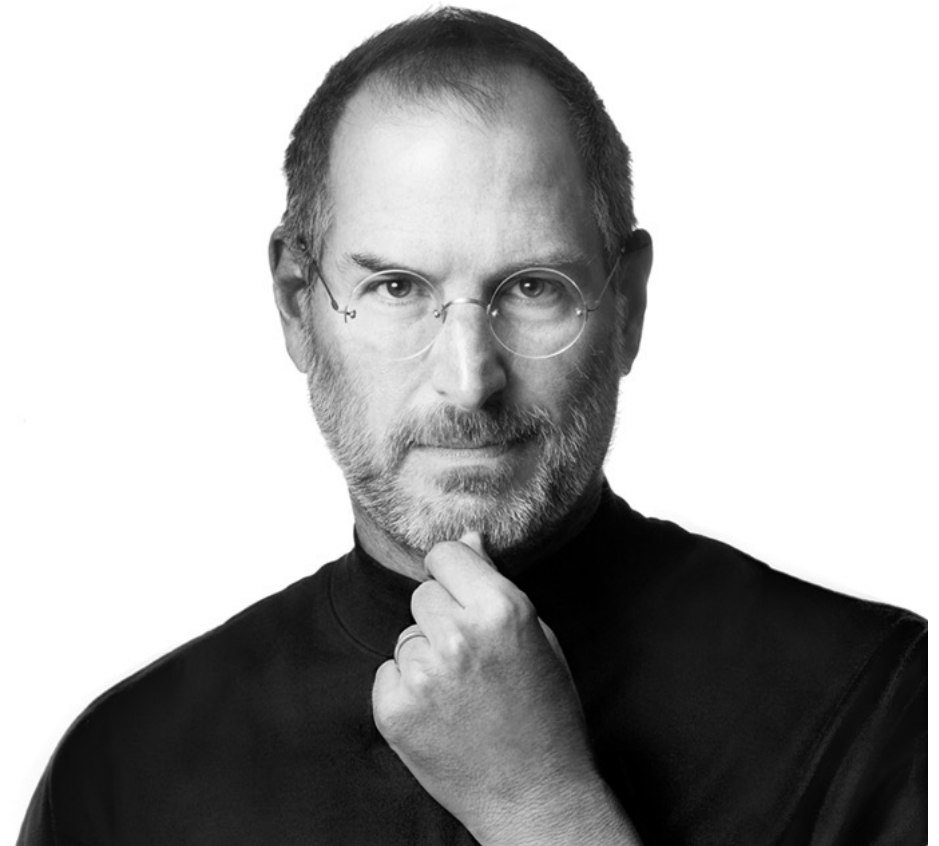
The masters are liable to get replaced because as soon as any technique becomes at all stereotyped it becomes possible to devise a system of instruction tables which will enable the electronic computer to do it for itself.

It may happen however that the masters will refuse to do this. They may be unwilling to let their jobs be stolen from them in this way. In that case they would surround the whole of their work with mystery...

(da una sua conferenza del 1947)

Steve Jobs sul software

"The problem is, in hardware you can't build a computer that's twice as good as anyone else's anymore. Too many people know how to do it. You're lucky if you can do one that's one and a third times better or one and a half times better. And then it's only six months before everybody else catches up. But you can do it in software." (1994)

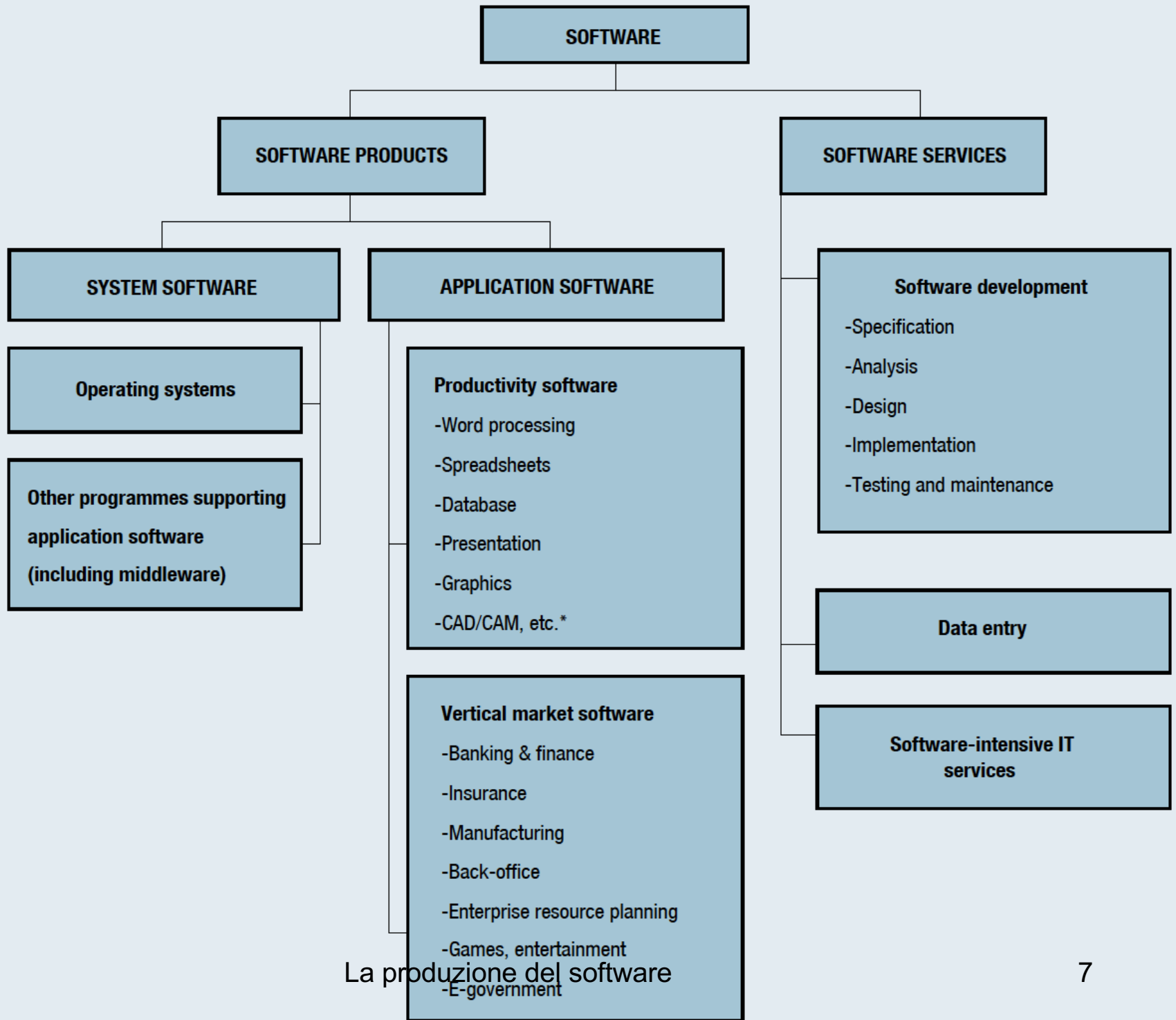


Una pubblicità RadioShack del 1991

Tutti i dispositivi mostrati sono oggi contenuti in uno smartphone

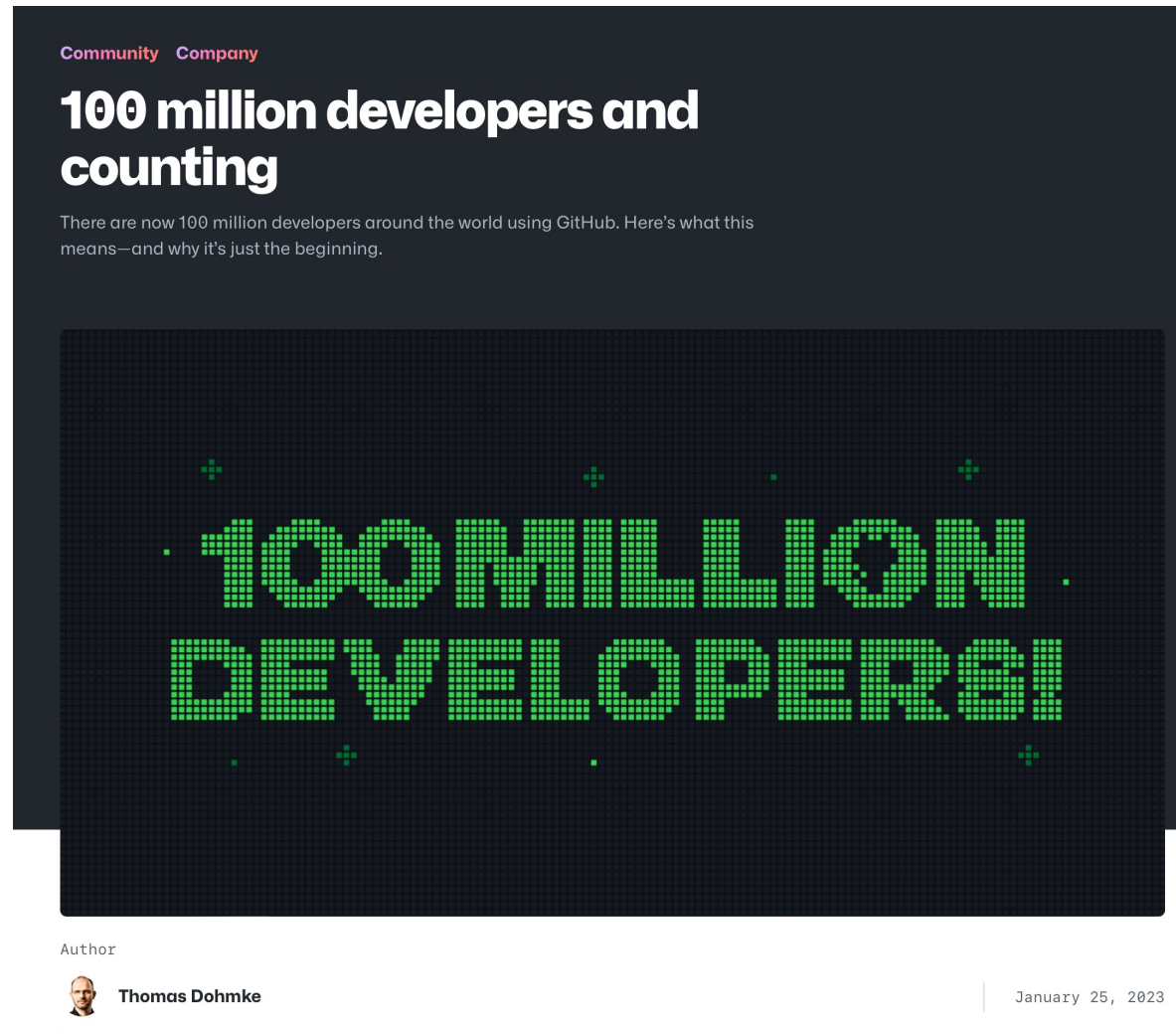
[illegible]

Tipi di software



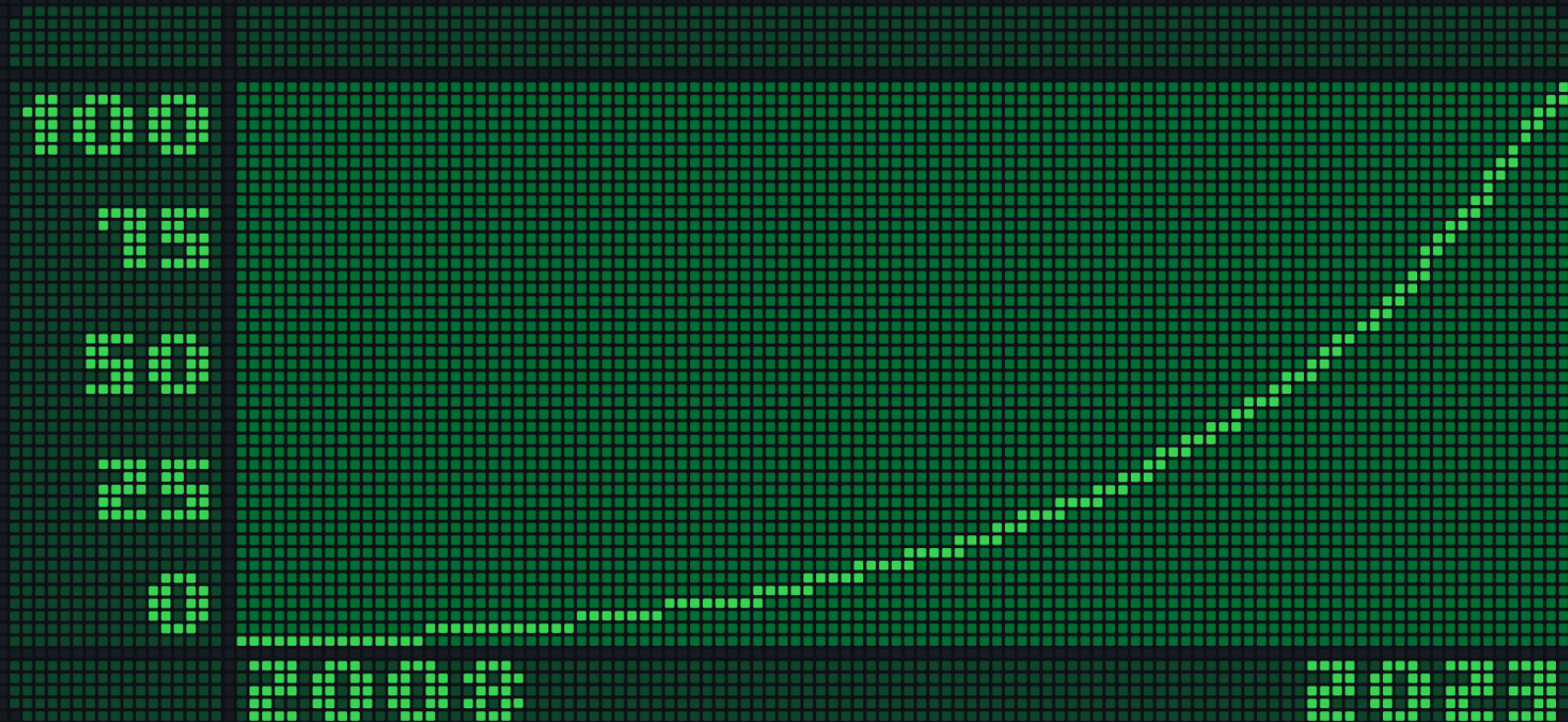
La produzione del software

100 milioni di sviluppatori!

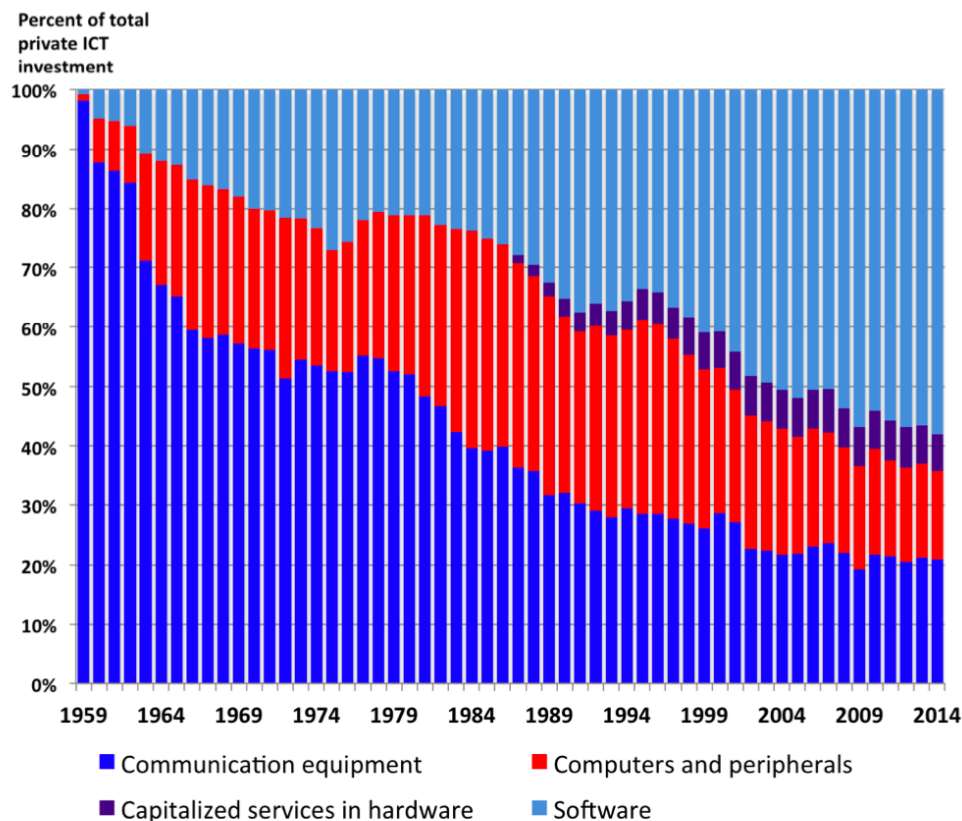


+ GITHUB USERS +

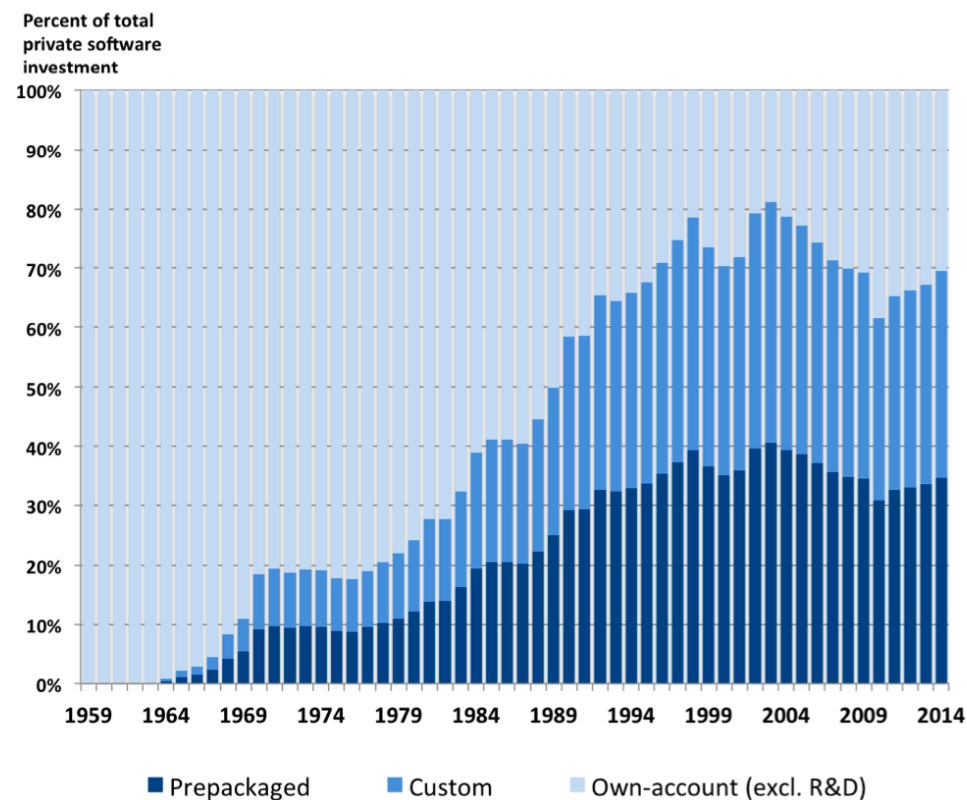
(IN MILLIONS)



Il software domina gli investimenti ICT



(a) ICT investment component shares



(b) Software investment component shares

Fonte: Byrne & Corrado, ICT Prices and ICT Services: What do they tell us about productivity and technology? 2017

Quanto costa produrre un'app?

- <http://howmuchtomakeanapp.com>

Siti per
sviluppatori
freelance
(esempio
www.freelancer.com)

PROJECT DESCRIPTION	BUDGET
Expert Android Application Developer hello, I am looking some expert honest android application for develop a custom android application... Like this here is the demo [login to view URL] You have to watch this demo more then 20 time befor ... See more Skills: Android iPhone Java Mobile App Development PHP	€636 - €1273 EUR Bid now
Custom Adapter Module in SAP PI Custom Adapter Module in SAP PI, Custom Adapter Module in SAP PI Skills: Java SAP	€15 EUR Bid now
software developed for linux I need you to develop some software for me. I would like this software to be developed for Linux using Java. Skills: Java Linux Software Architecture	€212 - €636 EUR Bid now
CREATE P2P CRYPTO EXCHANGE WEBSITE CAN U CREATE P2P CRYPTO EXCHANGE WEBSITE Skills: C Programming Java PHP Software Architecture Website Design	€18 - €146 EUR Bid now
Android Proof of Concept (PoC) detect if u are close a area (Place) or in the place Only GPS/Maps Experts (not genral apps) Ure android phone, should detect when u are 1km of and address (area of a place). Also when u are at 100 meters, and when u are in the place. [login to view URL] ... See more Skills: Android Google Earth GPS Java Kotlin	€8 - €25 EUR Bid now

La produzione del Software

Which Country Has the Best Developers?

Ranked by Average Score Across All HackerRank Challenges

Rank	Country	Score Index	Rank	Country	Score Index
1	China	100.0	26	Netherlands	78.9
2	Russia	99.9	27	Chile	78.4
3	Poland	98.0	28	United States	78.0
4	Switzerland	97.9	29	United Kingdom	77.7
5	Hungary	93.9	30	Turkey	77.5
6	Japan	92.1	31	India	76.0
7	Taiwan	91.2	32	Ireland	75.9
8	France	91.2	33	Mexico	75.7
9	Czech Republic	90.7	34	Denmark	75.6
10	Italy	90.2	35	Israel	74.8
11	Ukraine	88.7	36	Norway	74.6
12	Bulgaria	87.2	37	Portugal	74.2
13	Singapore	87.1	38	Brazil	73.4
14	Germany	84.3	39	Argentina	72.1
15	Finland	84.3	40	Indonesia	71.8
16	Belgium	84.1	41	New Zealand	71.6
17	Hong Kong	83.6	42	Egypt	69.3
18	Spain	83.4	43	South Africa	68.3
19	Australia	83.2	44	Bangladesh	67.8
20	Romania	81.9	45	Colombia	66.0
21	Canada	81.7	46	Philippines	63.8
22	South Korea	81.7	47	Malaysia	61.8
23	Vietnam	81.1	48	Nigeria	61.3
24	Greece	80.8	49	Sri Lanka	60.4
25	Sweden	79.9	50	Pakistan	57.4

La produzione del software



15

(nuove?) categorie di software

- Apps e software ecosystems
- Servizi software
- Nuovi strumenti di sviluppo
- Social software
- Scraping/mining big data
- Embedded software, IoT
- ...

Ecosistemi software



La produzione del software

Ecosistemi software

- Gli ecosistemi software sono mercati, in cui si vendono prodotti (es. AppStore o PlayStore) o componenti e servizi (es. Amazon Elastic Computing)
- La caratteristica principale è quella di una collezione di prodotti software, su piattaforma definita da un'azienda, che vengono sviluppati ed evolvono nello stesso ambiente
- Es. Appstore (al 2016): 100 miliardi di download, utili oltre 40miliardi\$; 20 “grandi” sviluppatori incassano il 50% degli utili.

Software as a service

Google: 2 miliardi di linee di codice

25.000 sviluppatori

45.000 commit al giorno

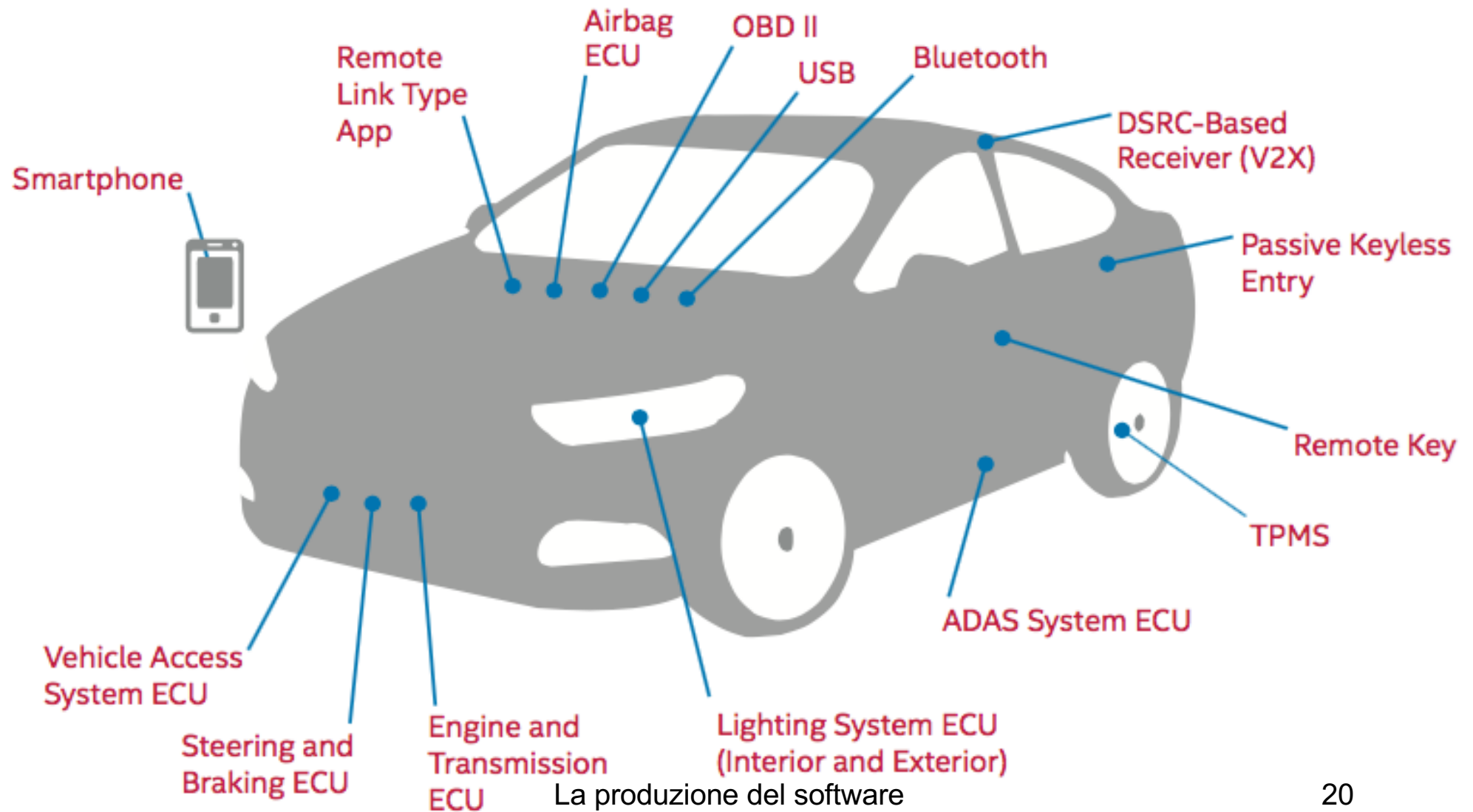
Chrome: 17.4 milioni di linee di codice

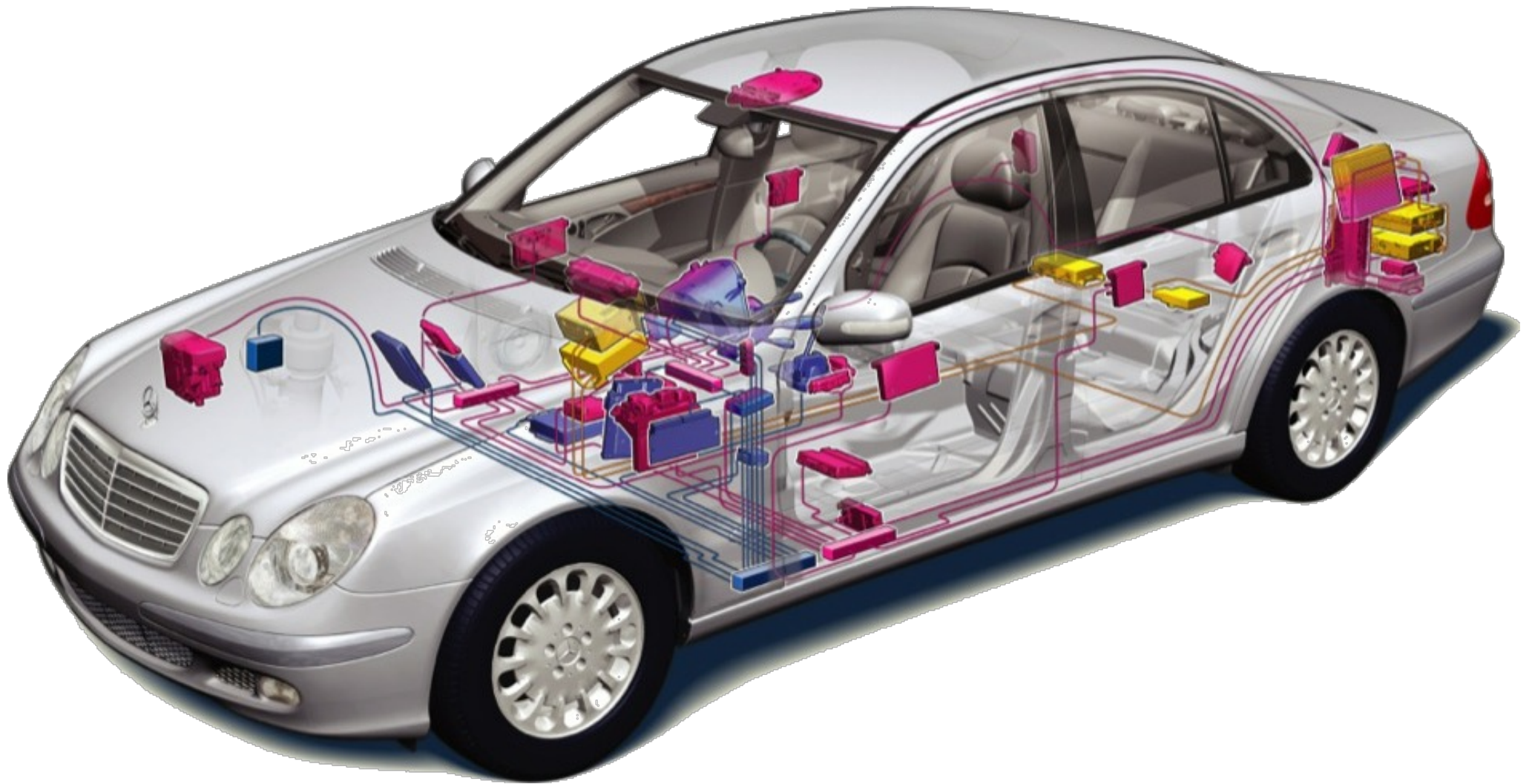
3.700 sviluppatori

380 commit al giorno



Software embedded (“nascosto”)





Social software (Web 2.0)

- Software che supporta la “conversazione” di comunità di utenti. Es. Facebook, X (Twitter), LinkedIn, Instagram, Pinterest, T2, ecc.
- Wiki, chat, forum, blog, ecc

Software libero (non gratis!)

0. A program can be run for any purpose
1. A program can be studied and changed to adapt it to new needs
2. A program can be freely distributed
3. A program can be freely improved and these improvements can be freely distributed

Free Software Foundation

La produzione del software

**FREE AS IN
FREEDOM**
RICHARD STALLMAN'S
CRUSADE FOR FREE SOFTWARE



Richard Stallman
FSF founder

Application Programming Interface (API)

- Amazon API: commercio sw driven
- PhilipsHue API: illuminazione sw driven
- Facebook API: social network sw driven
- GoogleMaps API: maps sw driven
- Stanford API
- CNN API
- Walmart API

Aspetti economici dei prodotti sw

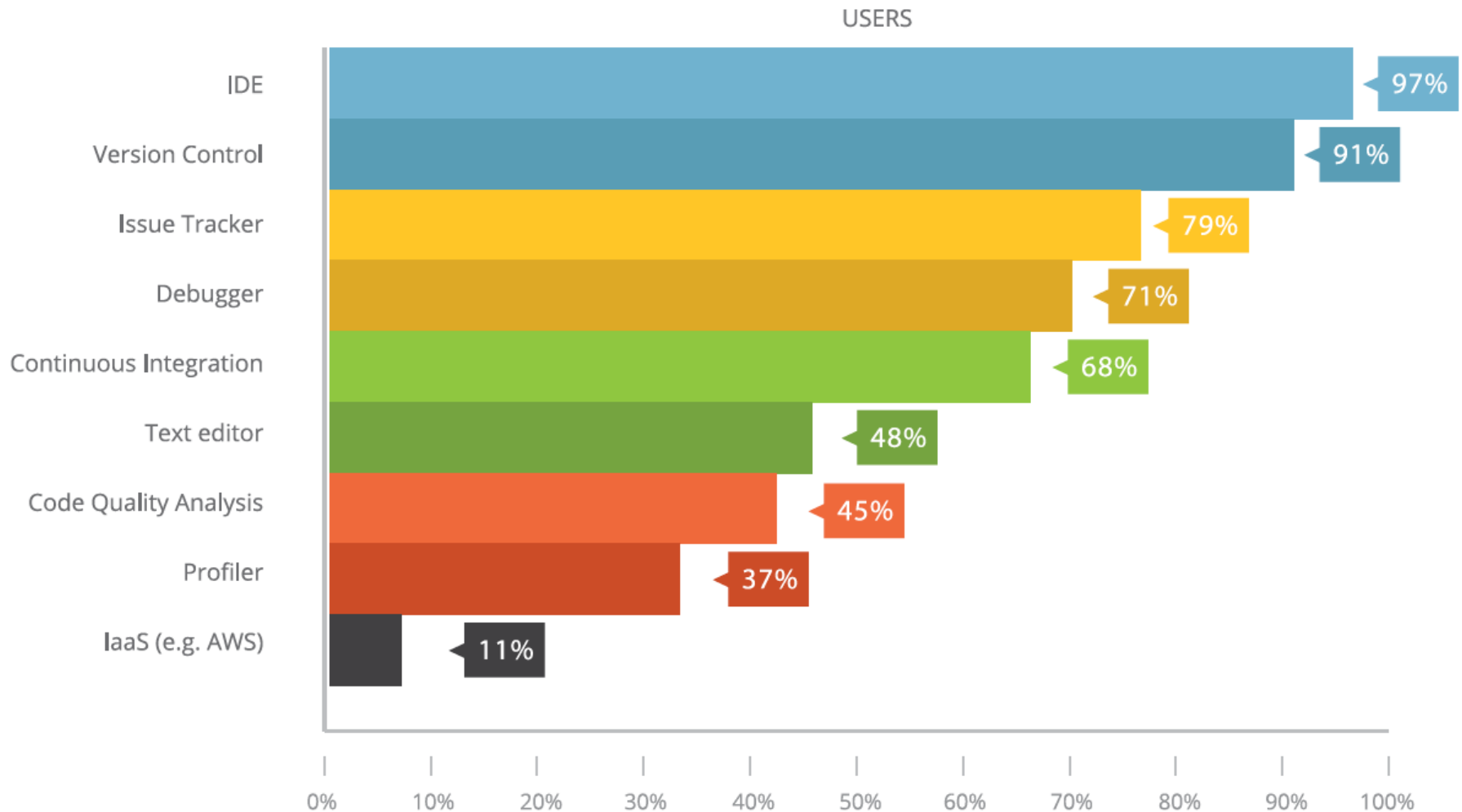
- Ambito d'uso (dimensione in righe)
- Piano di sviluppo (durata)
- Sforzo di sviluppo (costo)
- Produttività del team
- Qualità (difetti)

Videogiochi

- Sforzo tipico: 100 ÷ 500 anni/persona
- Team: di solito 50 ÷ 100 persone
(Assassin Creeds 2009: 450 persone)
- Vendere un milione di copie è ok ma non eccellente

<http://www.informationisbeautiful.net/visualizations/million-lines-of-code/>

Gli strumenti software statistiche d'uso



I sw sono sempre più grandi e costosi

It would cost over \$1 billion to develop REDHat 7.1 GNU/Linux distribution by conventional proprietary means in the U.S. (in year 2000 U.S. dollars).

Compare this to the \$600 million estimate for Red Hat Linux version 6.2 (which had been released about one year earlier).

Red Hat Linux 7.1 includes over 30 million physical source lines of code (SLOC), compared to well over 17 million SLOC in version 6.2.

Using the COCOMO cost model, this system is estimated to have required about 8,000 person-years of development time (as compared to 4,500 person-years to develop version 6.2).

Red Hat Linux 7.1 represents over a 60% increase in size, effort, and traditional development costs over Red Hat Linux 6.2. This is due to an increased number of mature and maturing open source / free software programs available worldwide (D.Wheeler, 2002)

Alcune cifre

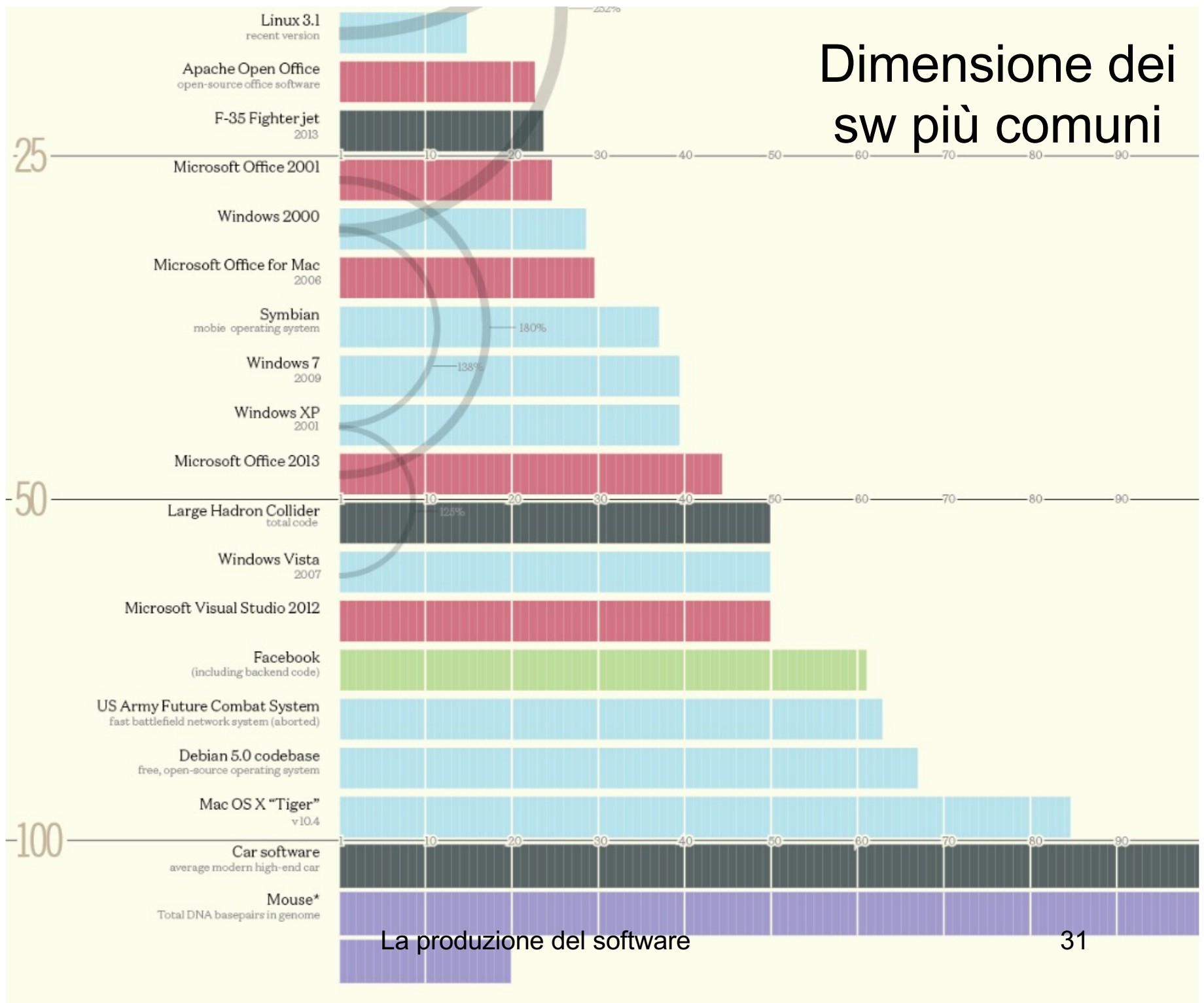
Prodotto	SLOC (righe di codice sorgente)
NASA Space Shuttle Flight Control	430K(shuttle) + 1.4M (ground)
Sun Solaris 1998-2000	7-8M
Microsoft Windows 3.1 (1992)	3M
Microsoft Windows 95	14M
Microsoft Windows 98	18M
Microsoft Windows NT (1992)	4M
Microsoft Windows NT5.0 (1998)	20M
RedHatLinux 6.2 (2000)	17M
MacOS 10.4 (2005)	86M
Linux kernel 4.2 (2016)	20.2M
Debian 7.0 (2012)	41.9M

Software sizes

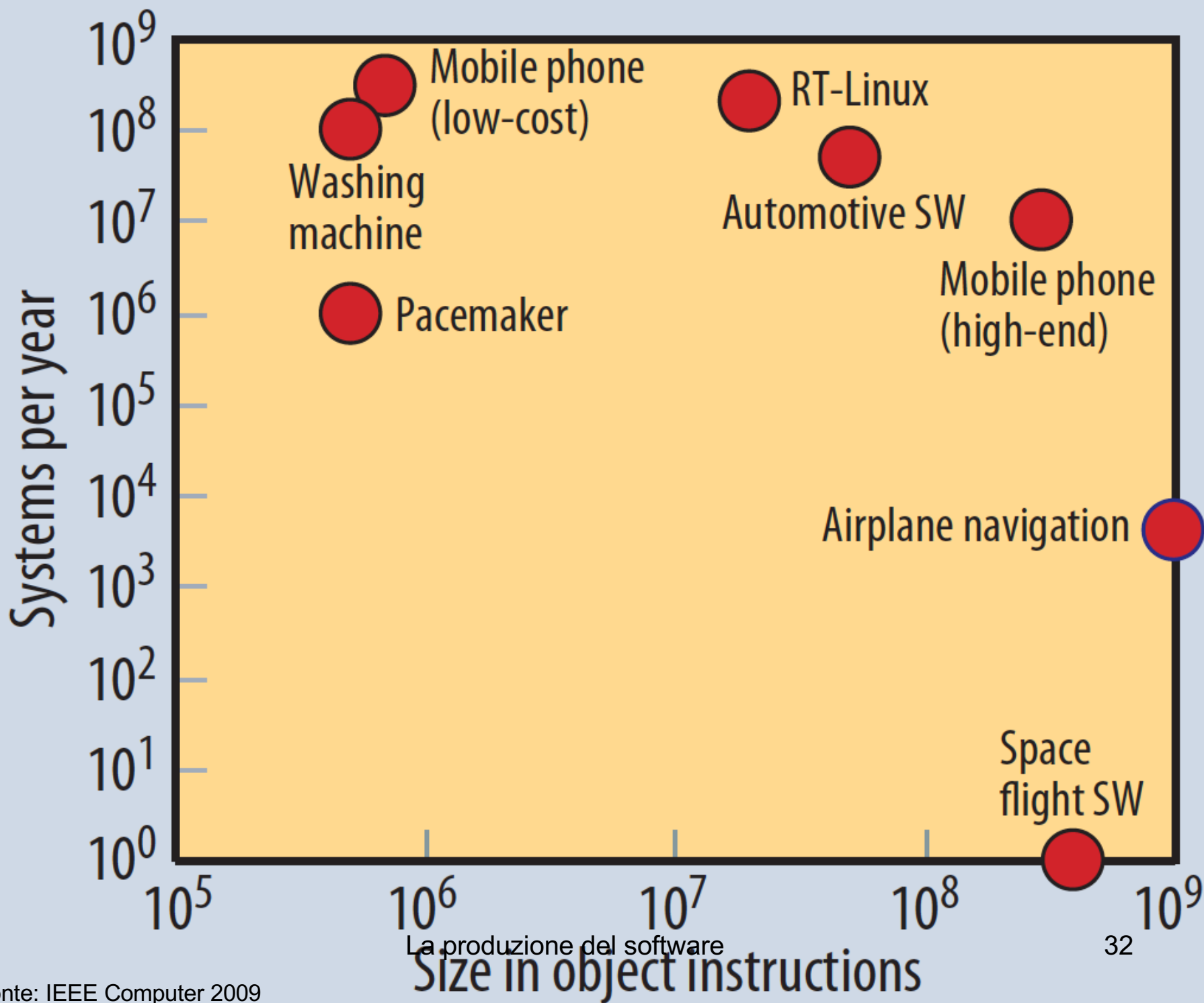
Category	Programmers	Duration	Size (Lines of Code)
Extremely Large	> 200	> 6 yrs.	>1,000,000
Very Large	20 - 200	3 - 6 yrs.	100,000 - 1,000,000
Large	5 - 20	2 - 3 yrs.	20,000 - 100,000
Medium	2 - 5	6 mo. – 2 yrs.	3,000 – 20,000

https://www.researchgate.net/publication/259359557_Software_Archaeology_and_the_Preservation_of_Code-based_Digital_Art/figures?lo=1

Dimensione dei sw più comuni



Software embedded



Software

- **Prodotto** invisibile, intangibile, facilmente duplicabile ma costosissimo: opera dell'ingegno protetta dalle leggi
- **Componente** di un sistema di elaborazione: può essere di larga diffusione (*off the shelf*) o *commissionato* da un singolo committente
- **Macchina astratta**; offre funzioni utili per qualche scopo, ha un'*architettura* (fatta di componenti e connettori)
- **Servizio**; ha un'*interfaccia* e si basa su una *infrastruttura*

Il sw è un prodotto industriale

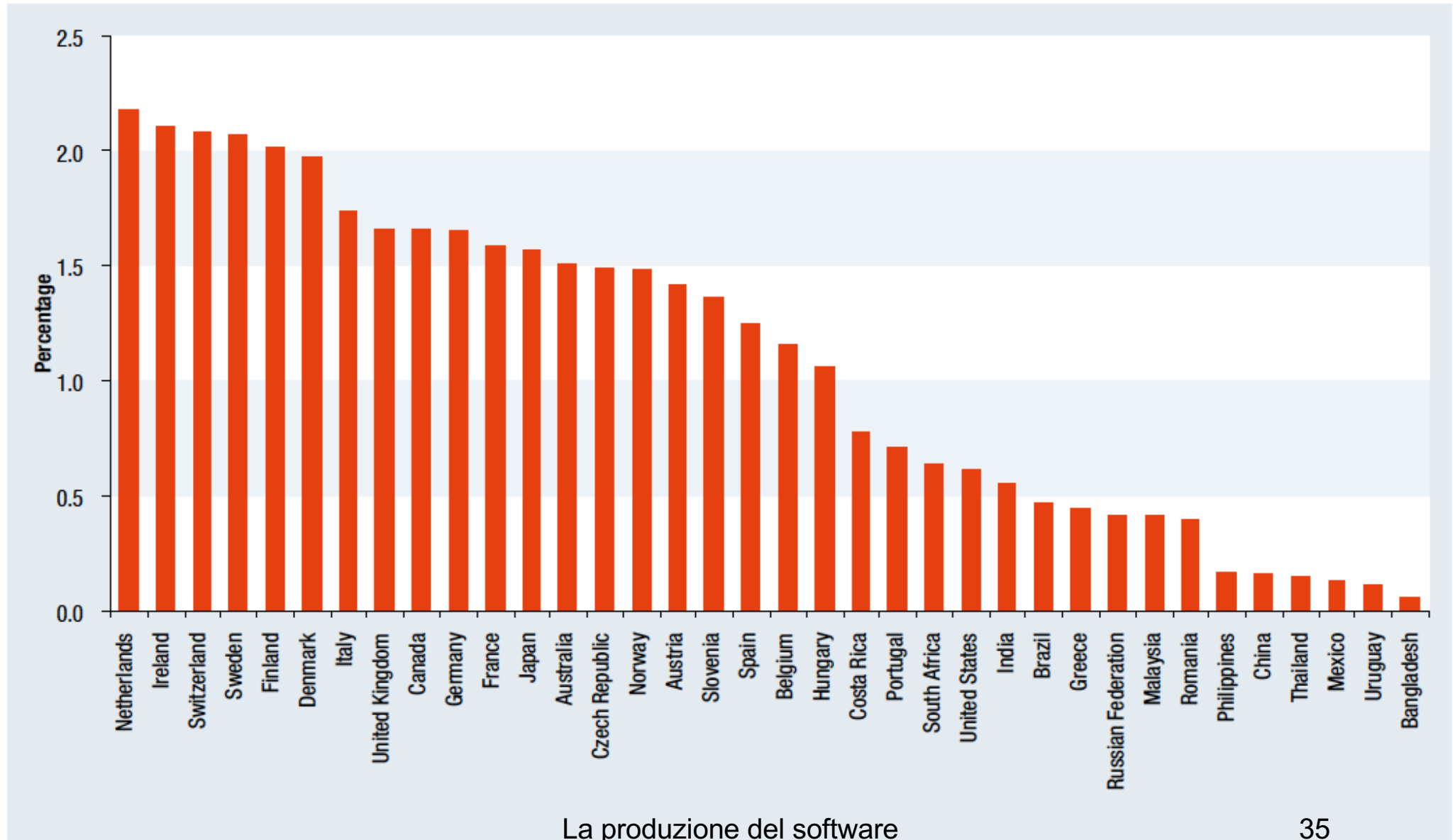
Il software è sempre il *prodotto di un processo di sviluppo*, che inizia con un'idea e termina quando il software viene ritirato

L'industria mondiale del sw cresce a tassi dal 5 al 10% annuo

Il costo di sviluppo di un prodotto software tende a crescere in proporzione al *quadrato* delle sue dimensioni

Quanti producono software

(occupati a produrre sw o servizi, in percentuale sul totale degli occupati)























Source: UNCTAD, based on international and national sources. Total employment data sourced from WITSA.

I grandi produttori di software in Europa

(2015, source Truffle Report)

La produzione del software

Rank	Company	Country of HQ location	Public	Software +Services 2014 (m€)	Total revenue 2014 (m€)	R&D employees 2014
1	SAP	DE		17 243.9	17 560.0	18908
2	Dassault Systemes	FR		2 078.6	2 346.7	5562
3	Sage	UK		1 539.5	1 620.5	1169
4	Hexagon	SE		1 442.3	2 622.4	3430
5	Wincor Nixdorf	DE		1 367.0	2 471.0	750
6	Asseco Group	PL		1 193.3	1 487.6	3696
7	Software AG	DE		835.6	857.8	968
8	DATEV	DE		790.7	843.5	1355
9	Wolters Kluwer	NL		740.2	3 660.0	2222
10	Misys	UK		639.5	639.5	1100
11	Micro Focus	UK		627.9	627.9	897
12	SWIFT	BE		596.8	628.0	493
13	Unit4	NL		516.0	516.0	1380
14	Visma	NO		464.7	851.7	730
15	Cegedim	FR		460.6	911.5	959
16	Sopra Steria	FR		445.4	3 370.0	800
17	Qlik	SE		418.9	418.9	358
18	Avaloq	CH		411.6	411.6	300
19	Swisslog	CH		411.4	551.3	146
20	Northgate Information Solutions	UK		373.7	826.0	600
21	Fiducia & GAD	DE		369.3	458.4	220
22	Compugroup Holding	DE		368.8	515.1	1426
23	Murex	FR		368.0	368.0	386
24	Temenos	CH		352.6	352.6	441
25	Fidessa	UK		341.0	341.0	437
26	Gemalto	NL		333.9	2 465.0	1105
27	IFS	SE		333.4	333.4	333
28	ESET	SK		328.8	328.8	382
29	Zucchetti	IT		310.3	358.0	900
30	Schneider Electric	FR		304.1	1 713.0	565
31	Reply	IT		287.0	632.2	390

Microsoft Says Its Software 'Ecosystem' Employs 15 Million

- » [E-Mail](#)
- » [Print](#)
- » [Discuss](#)
- » [Write To Editor](#)
- » [Digg](#)
- » [Slashdot](#)
- » [News Stories](#)

IDC research, paid for by Microsoft, also found that the company's partners earn \$7.79 for every dollar earned by Microsoft.

By [Paul McDougall](#)
[InformationWeek](#)

ottobre 19, 2007 01:38 PM

At a time when its business practices are under close scrutiny in the United States, Europe, and other parts of the world, Microsoft (NSDQ: [MSFT](#)) is touting a new study that says it's responsible for the creation of almost 15 million jobs globally.

IT work involving Microsoft and its [network](#) of partners kept 14.7 million workers employed worldwide, according to the study -- excerpts of which Microsoft made available on Friday.

The jobs range from software programming to system [integration](#) and help desk support.

The study was conducted by market researchers at IDC and paid for by Microsoft. IDC said the IT industry as a whole kept 35.2 million workers employed in 2007. "Software provides a disproportionate contribution to a vibrant IT economy," said John Gantz, chief research officer at IDC, in a statement.

IDC also found that Microsoft's partners earn \$7.79 for every dollar earned by Microsoft, and that the economic activity spurred by the Microsoft ecosystem will produce \$514 billion in tax revenue for governments worldwide in 2007.

I salari d'ingresso dei big players (2016)

Azienda	Stipendio annuo medio in \$ - junior	Bonus medio \$	Totale \$
Amazon	109.000	22.000	131.000
Apple	104.000	16.000	120.000
Google	86.000	20.000	106.000
Cisco	67.000	1.000	68.000
Oracle	67.000	-	67.000
Microsoft	58.000	9.000	67.000
Telefonica	45.000	4.000	49.000
Orange	48.000	-	48.000
IBM	48.000	-	48.000
SAP	44.000	4.000	48.000

Discussione

Come nasce il software?



Prodotti, sistemi, servizi

- Prodotti **generici** (OTS: off the shelf)
 - Prodotti creati da qualche produttore di software e venduti sul mercato a più (tanti) clienti
 - Es.: videogioco
- Sistemi **commissionati** (“customizzati”)
 - Sistemi commissionati da un cliente specifico e sviluppati apposta da un qualche fornitore
 - Es.: portale dell’Università
- Servizi **in perpetuo sviluppo**
 - Sistemi che offrono servizi 24/7 in continuo cambiamento
 - Es. Facebook, Amazon

Requisiti e feature del software

- **Requisito** software: funzione o qualità controllabile (testabile) che deve possedere l'implementazione di un prodotto software. È importante per il **cliente**
- **Feature** software: insieme di funzioni che permettono di usare un prodotto software in un servizio o prodotto. È importante per il **fornitore**

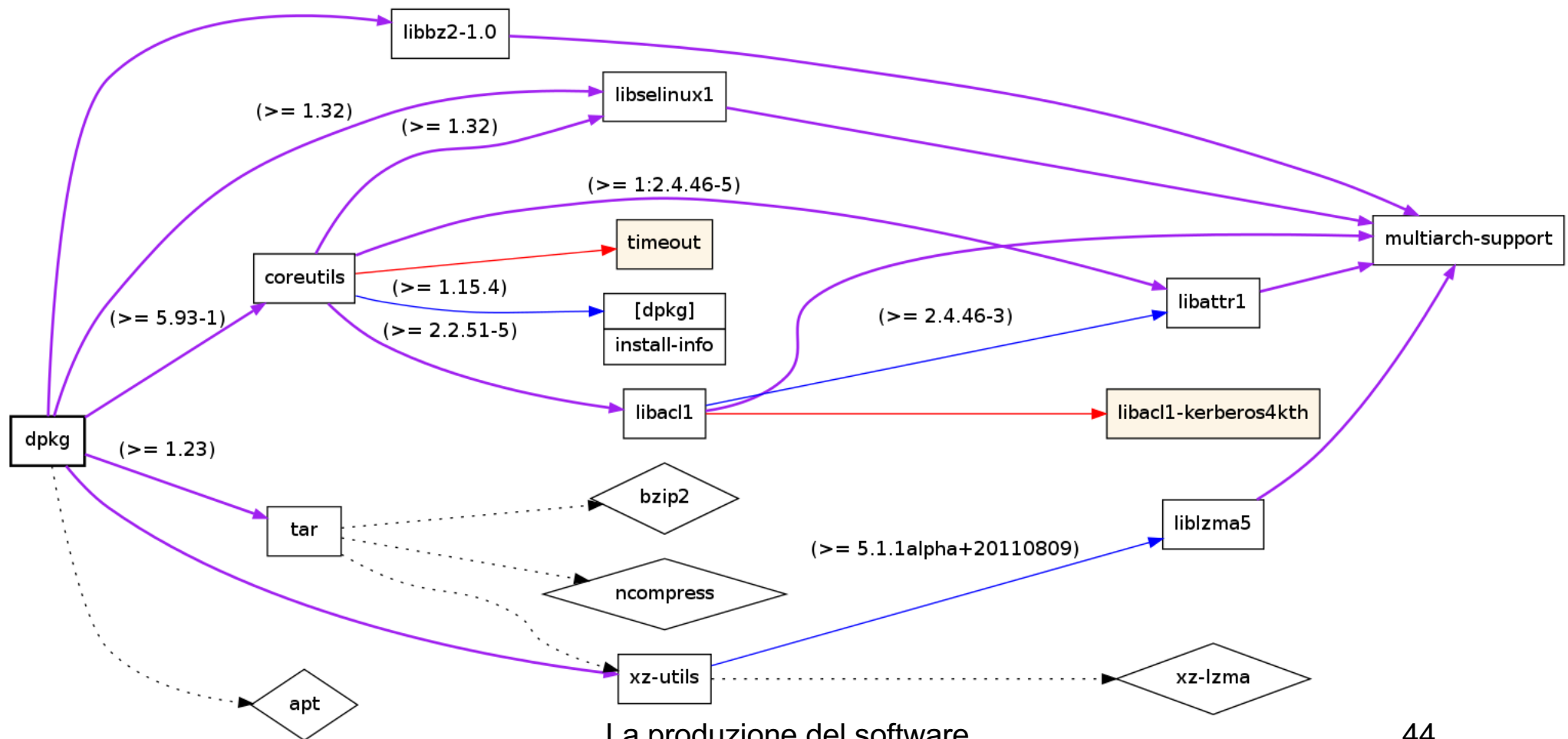
Esempio

- Feature: carrello per negozio elettronico
- Requisiti di un servizio di commercio elettronico: l'utente deve poter registrarsi, aggiungere o togliere elementi al carrello, specificare indirizzi alternativi, pagare

Dipendenze

- Ogni prodotto sw dipende da altri prodotti sw, che a loro volta dipendono da altri sw
- Associamo a ciascun prodotto o sistema software un **grafo di dipendenze**
- I nodi del grafo delle dipendenze sono pacchetti software (es librerie) in diverse versioni

Esempio



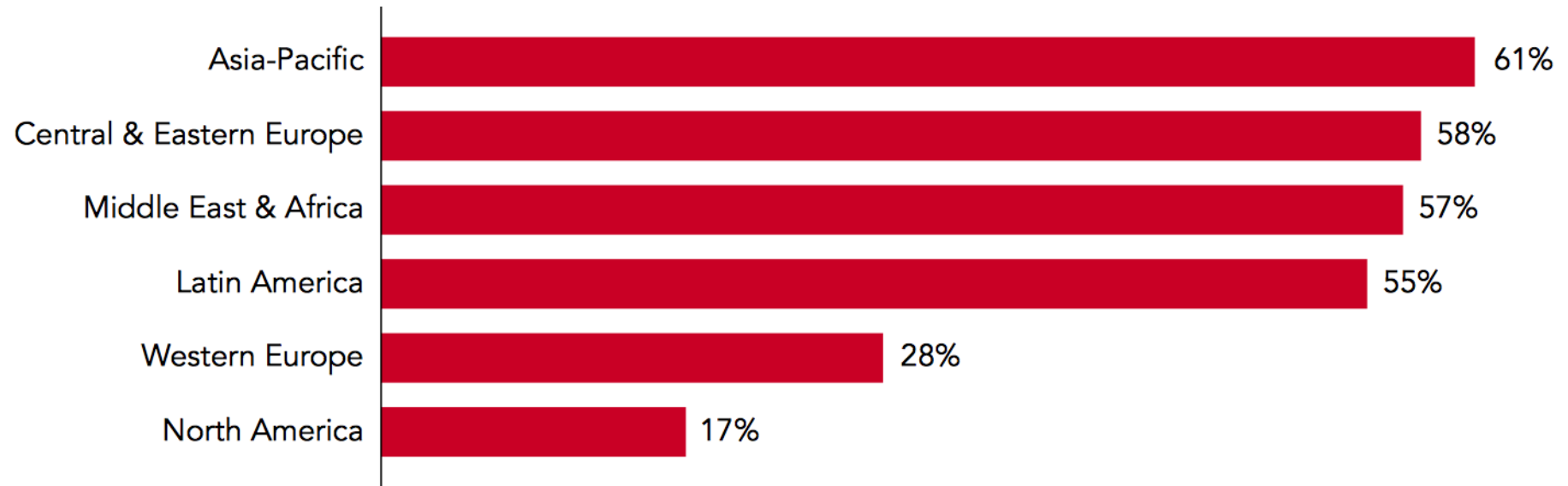
La produzione del software

<http://collab-maint.alioth.debian.org/debtree/>

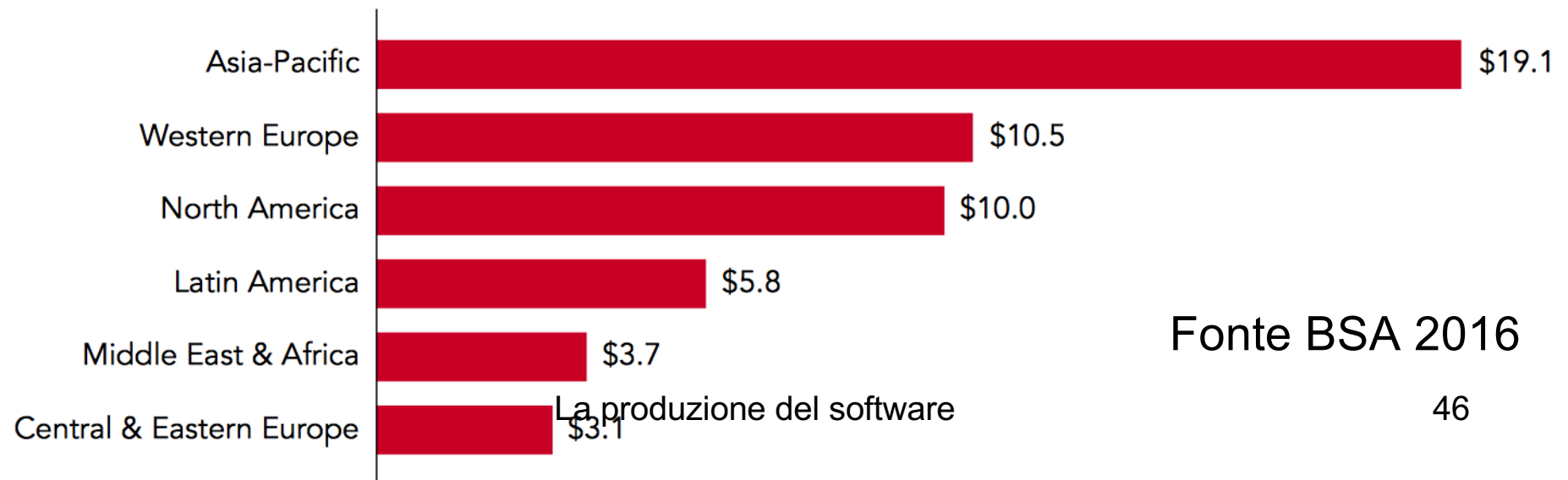
Il software è un prodotto speciale

- È invisibile e intangibile
- Ogni prodotto ha molte dipendenze
- È facilmente duplicabile e distribuibile su rete
- In Europa non è brevettabile (ma protetto)
- Il software di consumo non è garantito
- Viene acquisito su **licenza**
 - Proprietaria (normale, shareware, freeware)
 - Public domain
 - Open source

Average Rate of Unlicensed Software Use



Commercial Value of Unlicensed Software Use (in Billions)



Fonte BSA 2016

Protezione legale del sw

- **Protezione dell'autore:** Il software è un'opera dell'ingegno: chi lo produce è un autore che ha diritto ad un compenso

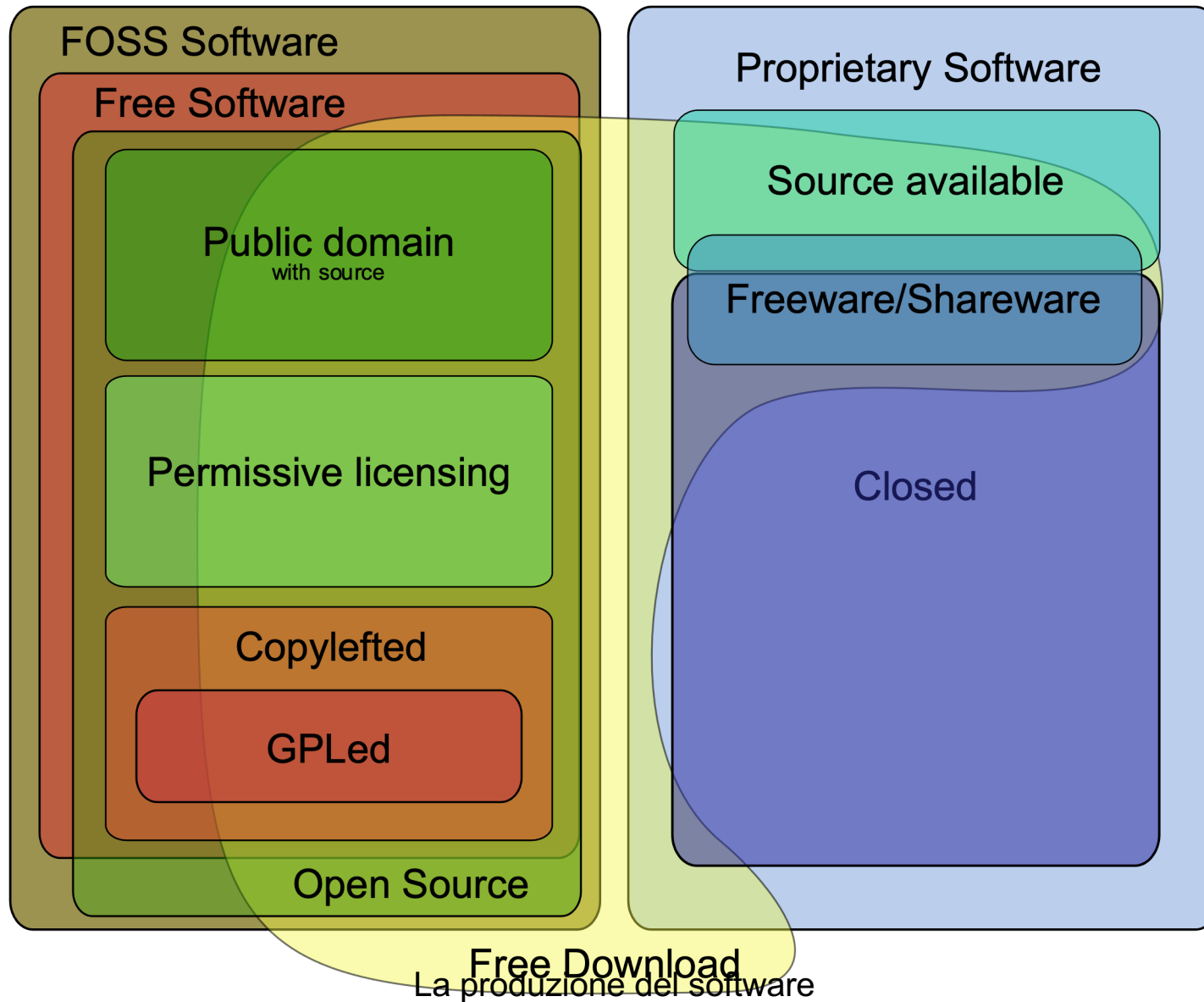
Copiare software abusivamente è **illegale** (anche se non lo si fa per profitto) e in Italia costituisce un reato penale:

La legge italiana 248/2000 punisce col carcere da 6 mesi a 3 anni chi duplica abusivamente software

- Per informazioni sulla brevettabilità del software negli USA:

<http://www.softwarepatent.com>

Le licenze software



SIAE: il pubblico registro sw

- Possono essere registrati i sw che rispettino requisiti di originalità e creatività tali da poter essere identificati come opere dell'ingegno.
- è possibile registrare tutti gli atti che trasferiscono in tutto o in parte diritti di utilizzazione economica relativi a programmi per i quali sia già avvenuta la registrazione
- Per registrare un programma, il richiedente deve trasmettere a SIAE una "dichiarazione" e una "descrizione" oltre, naturalmente, ad un esemplare del programma da depositare registrato su supporto digitale non riscrivibile

Decompilare un sw

- Si può decompilare un software?
- Per esempio, chi l'ha comprato ed ha solo il codice oggetto può decompilarlo per correggere un difetto?

<https://www.lexology.com/library/detail.aspx?g=363a5a27-c2eb-4163-be9d-928784b6a90f>

La garanzia del software

Protezione del compratore:

Quale protezione ha il compratore da difetti del prodotto?

Nel software di consumo in teoria NON c'è alcuna garanzia.

Il software viene venduto “così com'è”, e se ci sono difetti il fabbricante non se ne fa carico:

lo dice il contratto che si visualizza quando si usa per la prima volta un'applicazione

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is **licensed, not sold**.

1. GRANT OF LICENSE. The SOFTWARE PRODUCT is licensed as follows:

* Installation and Use. Microsoft grants you the right to **install and use copies** of the SOFTWARE PRODUCT on *your computers running validly licensed copies* of the operating system for which the SOFTWARE PRODUCT was designed [e.g., Windows(r) 95; Windows NT(r), Windows 3.x, Macintosh, etc.].

* Backup Copies. **You may also make copies** of the SOFTWARE PRODUCT as may be necessary for backup and archival purposes.

* Components. Certain software components of the SOFTWARE PRODUCT are subject to the following additional provisions:

2. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

* Distribution. You may not distribute copies of the SOFTWARE PRODUCT to **third parties**.

* **Prohibition on Reverse Engineering, Decompilation, and Disassembly.**

4. COPYRIGHT. All title, including but not limited to copyrights, in and to the SOFTWARE PRODUCT and any copies thereof are owned by Microsoft or its suppliers. All rights not expressly granted are reserved by Microsoft.

8. NO WARRANTIES. To the maximum extent permitted by applicable law, Microsoft and its suppliers provide the SOFTWARE PRODUCT and any (if any) Support Services related to the SOFTWARE PRODUCT **AS IS AND WITH ALL FAULTS**, and hereby disclaim all warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties or conditions of merchantability, of fitness for a particular purpose, of lack of viruses, of accuracy or completeness of responses, of results, and of lack of negligence or lack of workmanlike effort, all with regard to the SOFTWARE PRODUCT, and the provision of or failure to provide Support Services.

ALSO, THERE IS **NO WARRANTY** OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE PRODUCT.

THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING BUT NOT LIMITED TO THE WARRANTY OF TITLE AND QUANTITY. THE SOFTWARE IS PROVIDED AS TO THE QUALITY OF OR ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE PRODUCT AND SUPPORT SERVICES, IF ANY, **REMAINS WITH YOU**.

La produzione del software

Garanzie sul software

- La **verifica** garantisce l'aderenza ad una specifica
- La **validazione** garantisce l'accettazione da parte del cliente
- La **certificazione** garantisce l'aderenza a specifiche definite dalla legge

NB: il sw commerciale di solito viene venduto **senza** garanzie (“*as is*”)

I rischi

- Rischi di sviluppo dei sistemi software
- I difetti nel software operativo
- Rischi di esercizio dei sistemi software

Rischi di esercizio

Luglio 2015: **Fiat richiama 1,4 milioni di Jeep**

- Nuova svolta dopo il famoso caso in cui alcuni hacker avevano violato il sistema multimediale UConnetc di una Jeep Cherokee: FCA, dopo aver rilasciato immediatamente un aggiornamento di software ha deciso di richiamare in officina per un controllo gratuito 1,4 milioni di vetture per sistemare definitivamente il sistema informatico di bordo
- “Tutti i settori sono potenziali bersagli di un hacker e l'industria automobilistica non ha fatto eccezione”, ha dichiarato Gualberto Ranieri, capo della comunicazione FCA mercati Nafta spiegando poi che “non c'è stato un solo incidente nel mondo reale in cui è stato coinvolto qualsiasi veicolo FCA a seguito di un'intrusione pirata nei suoi software”.

Alcune qualità del software

- **Robustezza**: sw capace di sopportare errori durante l'esecuzione
- **Sostenibilità**: software duraturo capace di essere modificato economicamente
- **Riproducibilità**: controllo delle versioni

Alcuni dati

Numero di difetti (fault) rilevati durante l'esercizio

- I peggiori sistemi militari: 55 faults/KLoC
- I migliori sistemi militari: 5 faults/KLoC
- Prodotti ottenuti con sviluppo agile (XP): 1.4 faults/KLoC
- Apache web server (open source): 0.5 faults/KLoC
- NASA Space shuttle: 0.1 faults/KLoC

www.easterbrook.ca/steve/?p=1366

Domande di autotest

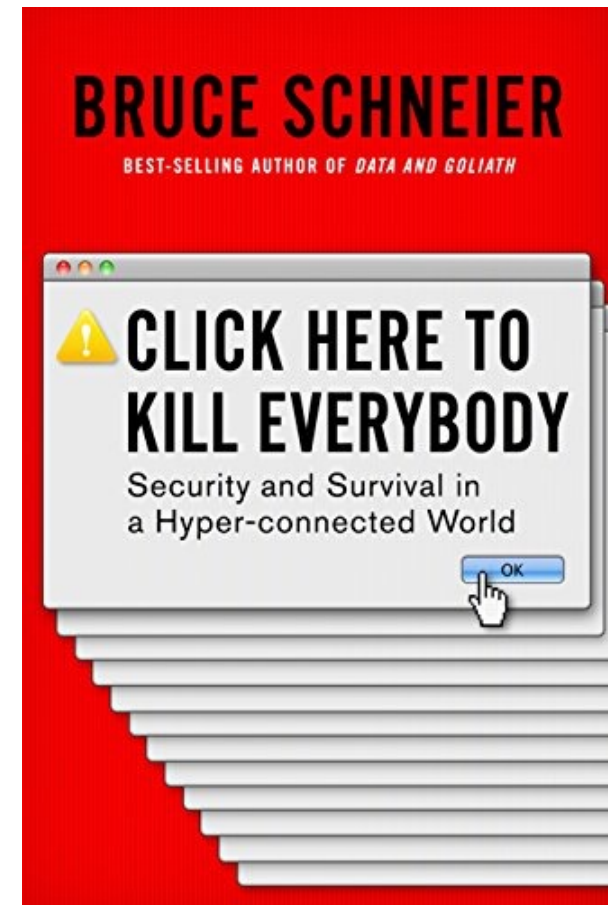
- Cos'è il software? Quanti tipi di sw esistono?
- Perché è costoso produrlo?
- Quali sono i rischi dell'uso del software?
E quelli dello sviluppo?
- Cos'è un difetto software?
- Quanto software può produrre in un anno
 - Una persona? Un'organizzazione? Una nazione?

Libri sul fare software

Oram e Wilson, *Making Software*,
O'Reilly, 2011

Martin, *Clean code. Guida per
diventare bravi artigiani nello sviluppo
agile di software*, 2018

Schneier, *Click here to kill everybody*,
2018



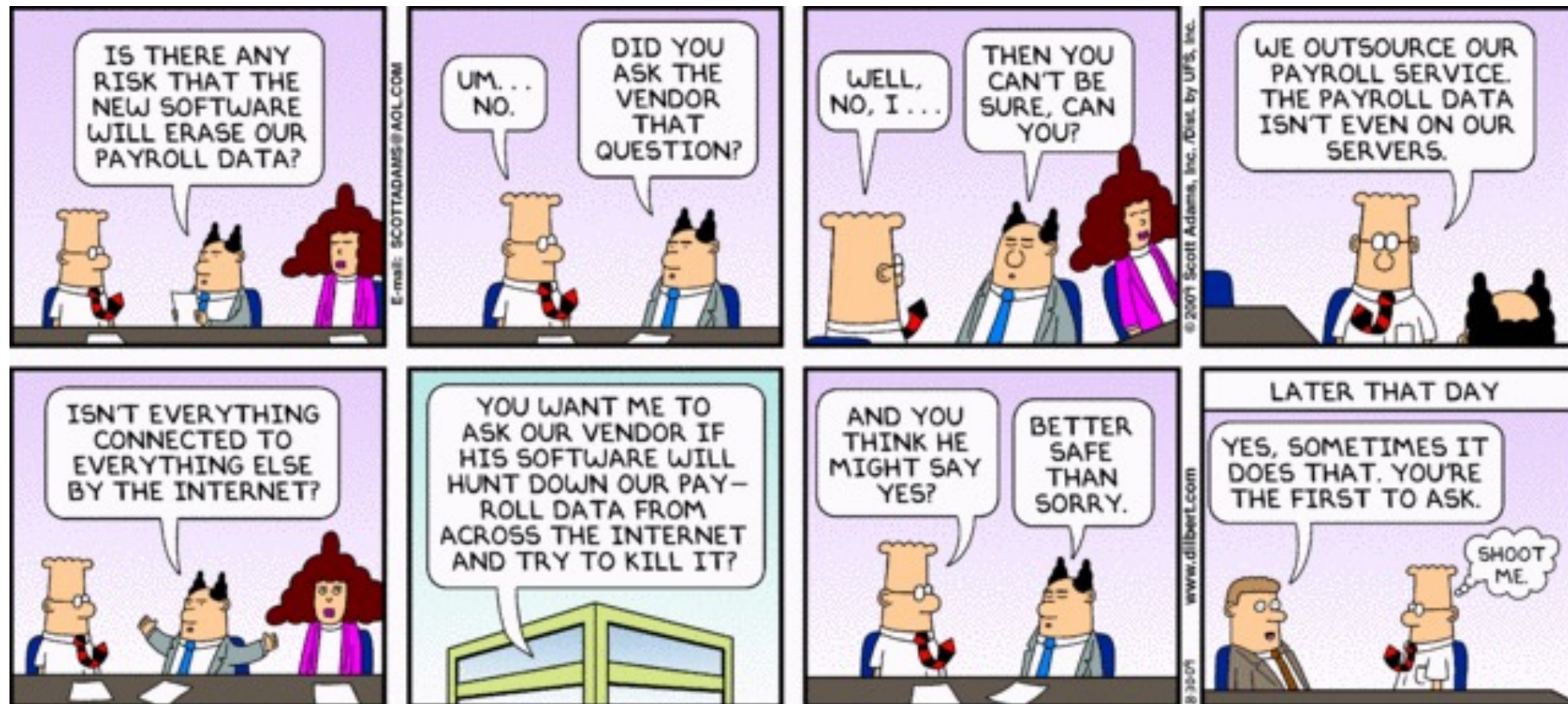
Blog e altro

- `www.joelonsoftware.com`
- `stackoverflow.com`
- `www.freelancer.com`
- `best-practice-software-engineering.blogspot.com`

Gruppi linkedin

- Software developer
- Software testing and quality assurance
- Software as a service

Domande?



Ingegneria del Software



Corso di Ingegneria del Software
CdL Informatica Università di Bologna

Obiettivi di questa lezione

- Cos'è l'ingegneria del software?
- Il ciclo di vita del software
- Il processo di sviluppo del software
- Miti e leggende della produzione sw

From programming to development

Cosa è più difficile:

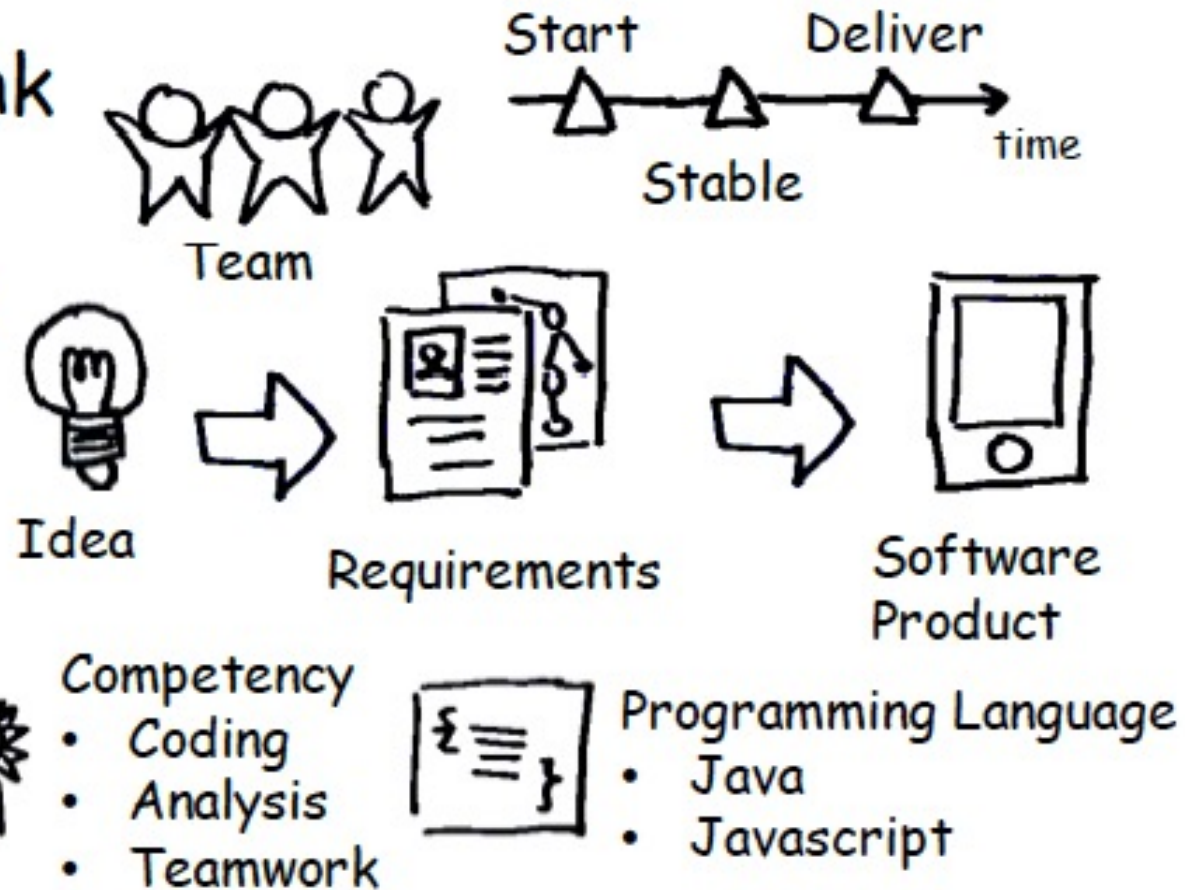
scrivere software, oppure

leggerlo (per es. per modificarlo)?

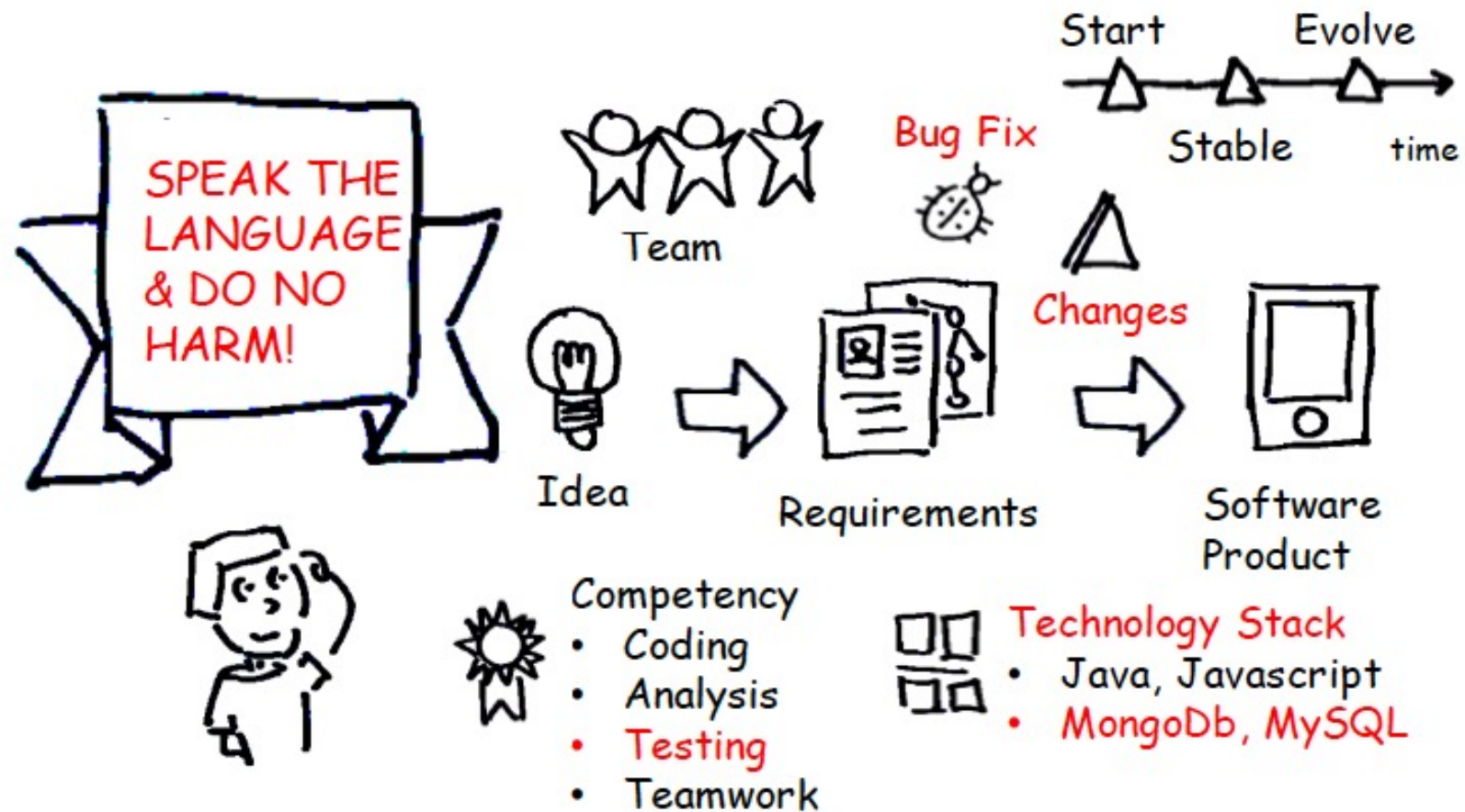
Come si **sviluppa** il software?

Punto di vista: studente

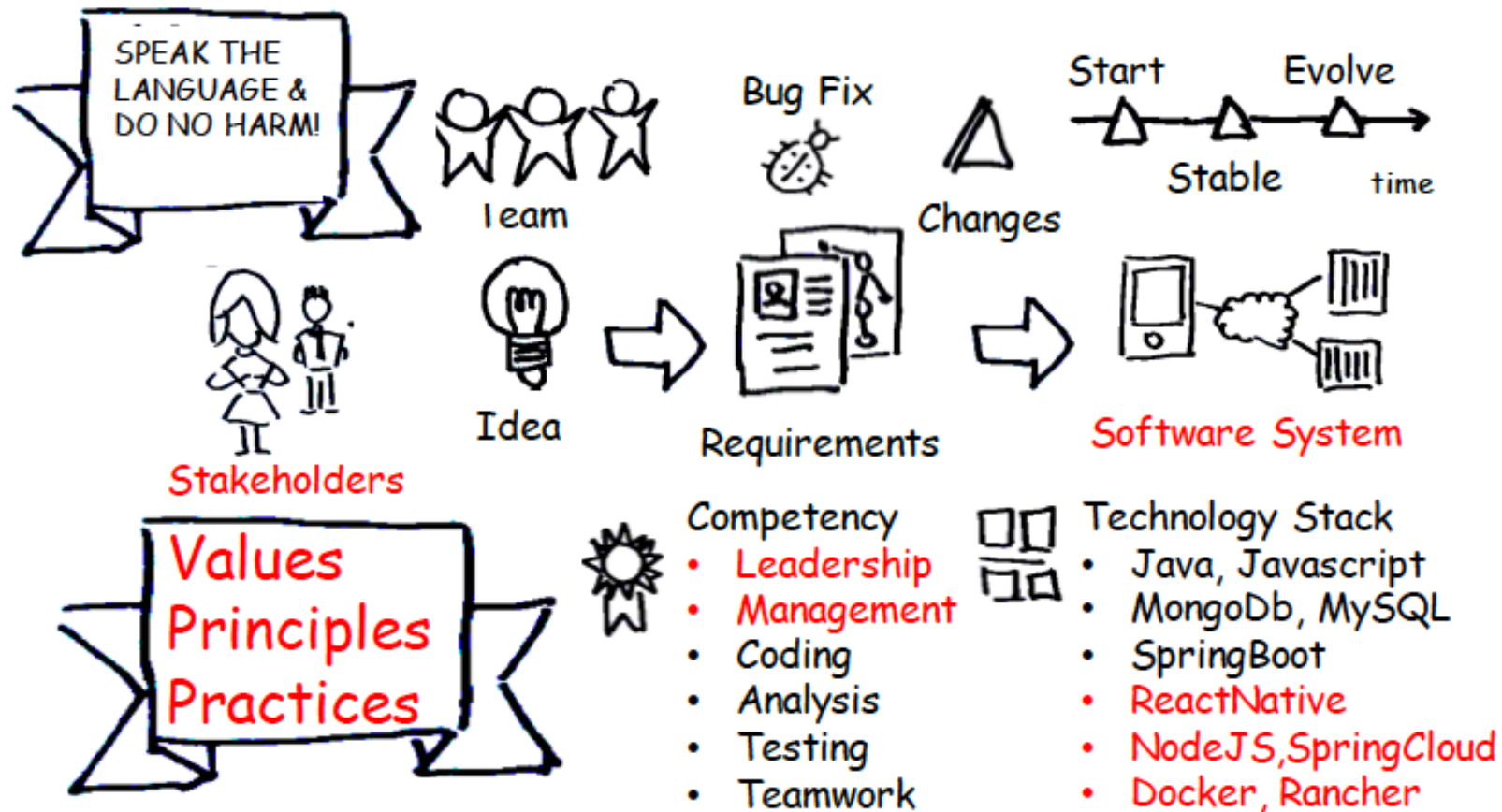
What I think
software
engineering
is about



Punto di vista: tesista in azienda



Punto di vista: professionista



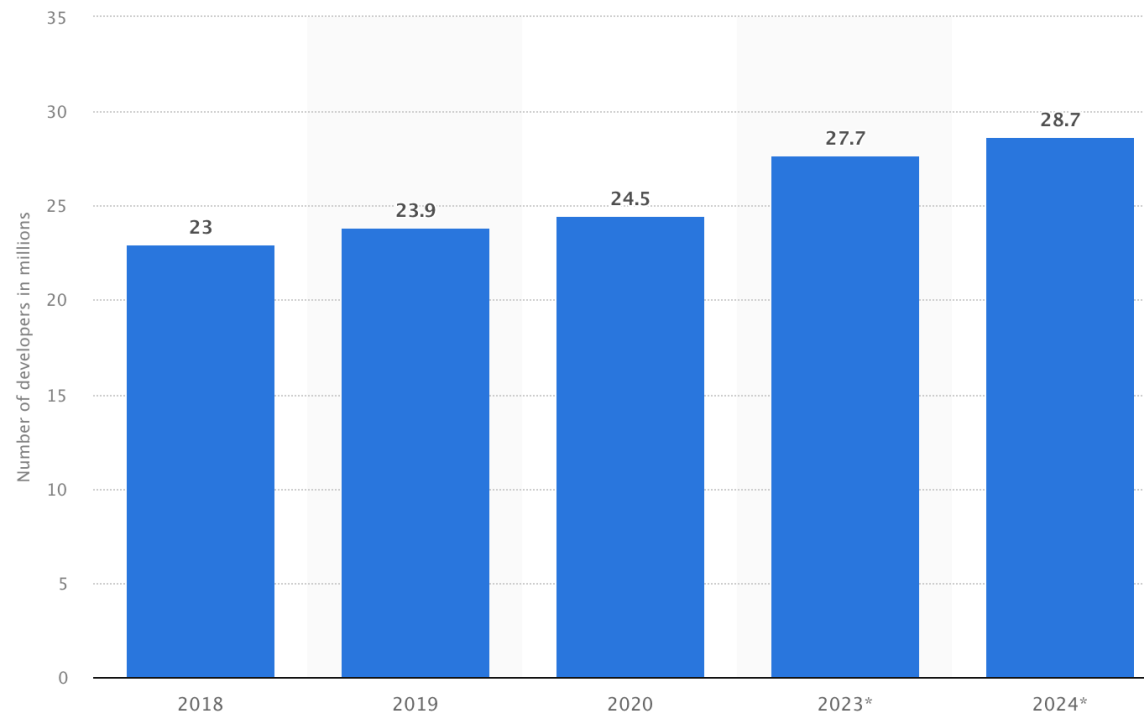
Ingegneria del software

- L' Ingegneria del Sw (Software Engineering) è una *disciplina metodologica*, cioè studia i *metodi* di produzione, le *teorie* alla base dei metodi, e gli *strumenti* di sviluppo e misura della *qualità* dei sistemi software
- È anche una *disciplina empirica*, cioè basata sull'esperienza e sulla storia dei progetti passati
- NB: manca una TEORIA dell'ingegneria del sw

La professione

Technology & Telecommunications › Software

Number of software developers worldwide in 2018 to 2024
(in millions)



[Additional Information](#)

© Statista 2022

[Show source](#)

Ingegneria del software

La professione

- Nei paesi anglosassoni “*software engineer*” è una **professione** riconosciuta
- Oltre metà di tutti gli ingegneri USA sono “sw engineers”

Fonte: https://en.wikipedia.org/wiki/Software_engineering_demographics#United_States

<http://computer-careers-review.toptenreviews.com/software-engineer-review.html> dati al 2015

<https://evansdata.com/reports/viewRelease.php?reportID=9> dati al 2018

Produttività

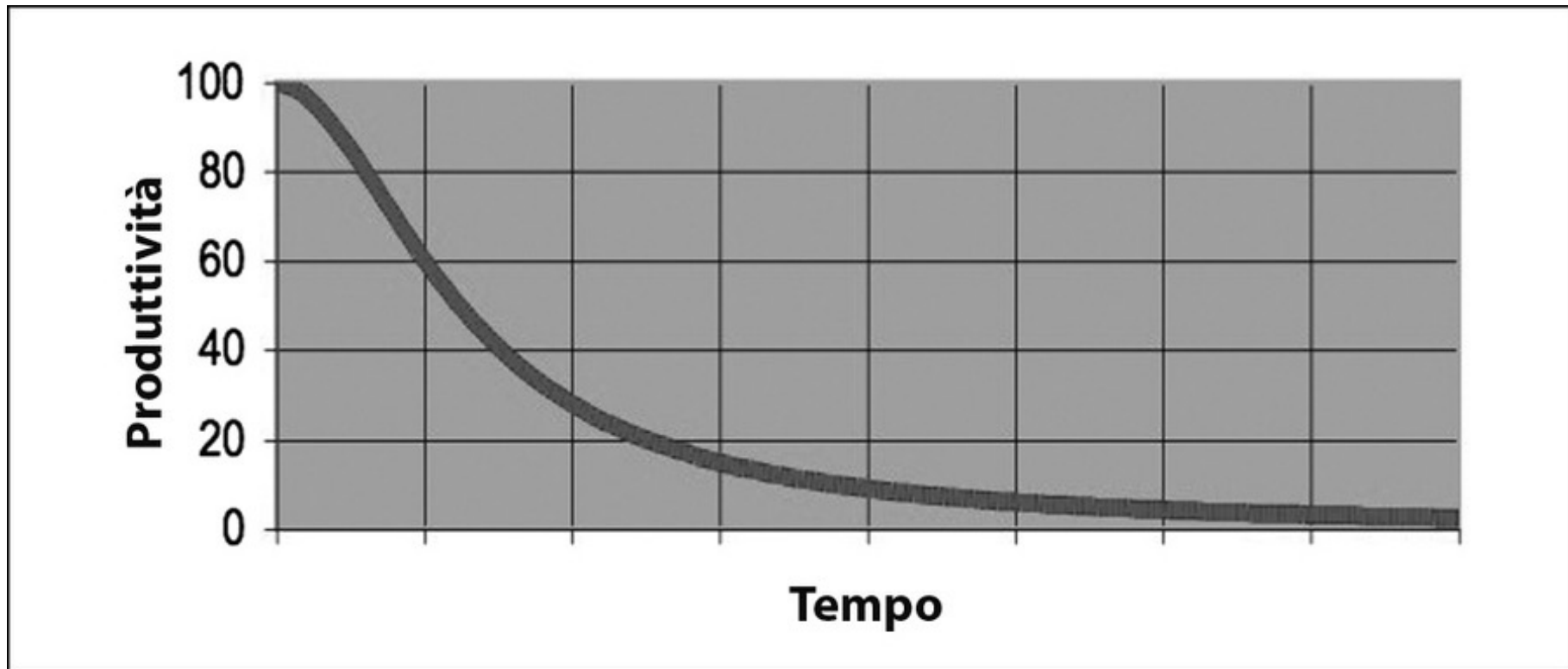
- La produttività è il rapporto tra la **quantità** di beni o servizi prodotti ed il **costo del lavoro** necessario a produrli
 - Output/Input.

Esempio: produco 1000 caramelle al costo di 100 euro.
La produttività è: 10 caramelle per ogni euro speso

Produttività degli sviluppatori

- La produttività dello sviluppo software è il rapporto tra **software prodotto** (misurato in LoC: Lines of Code) e il **costo dello sforzo (effort, misurato in giorni/persona)** di produrlo
 - **LoC/effort** (esempio: 50 LoC per giorno/persona)

Produttività nel software



Misurare la produttività



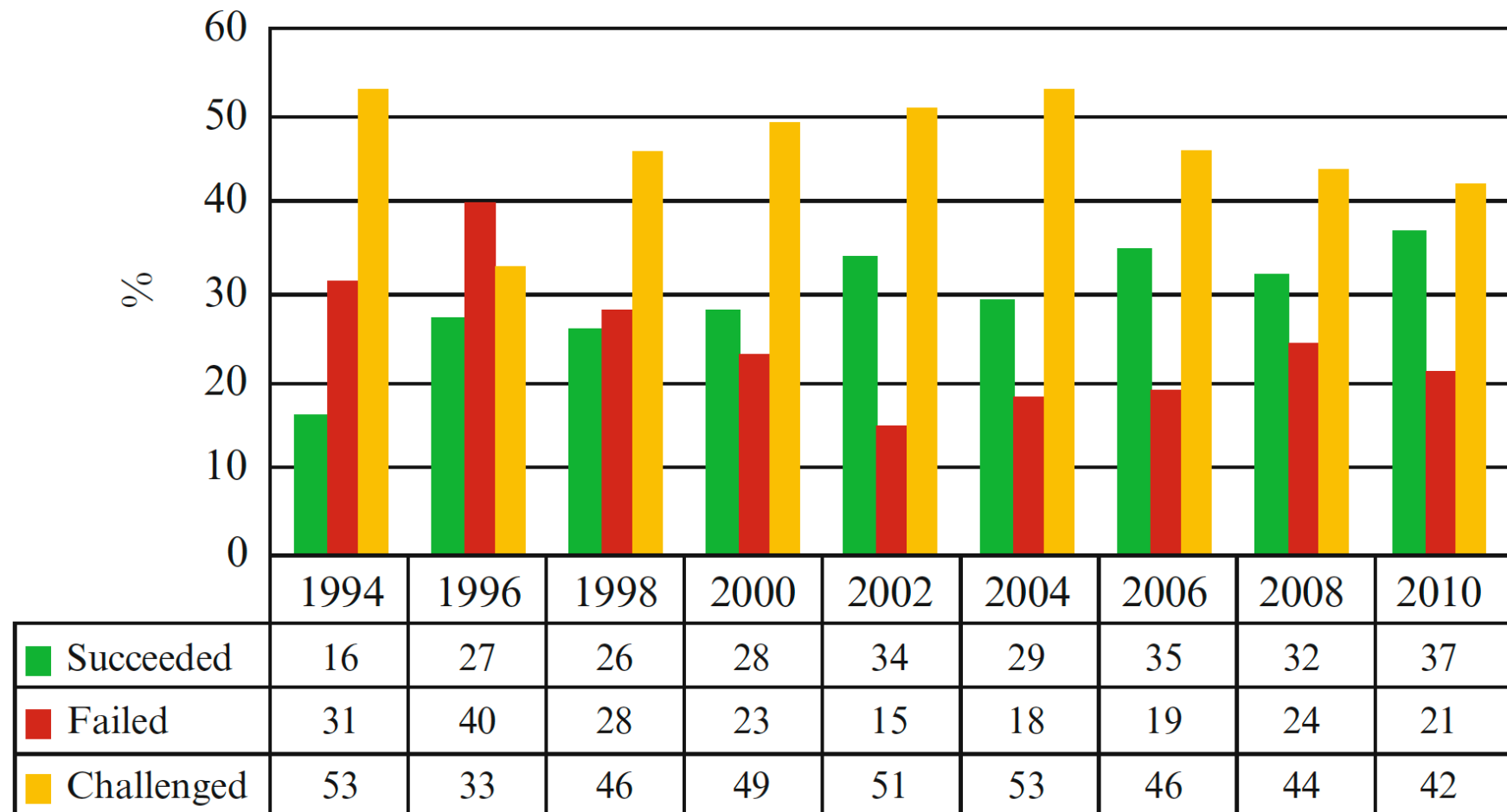
La produttività dell'industria sw è bassa

- Da un'analisi di 13.522 progetti di costruzione sw:
 - 66% di tutti i progetti **falliscono** (non hanno risultato utile)
 - 82% dei progetti superano i tempi previsti
 - 48% dei progetti producono sistemi senza le funzioni richieste dai clienti
 - 55 miliardi \$ di spreco considerando solo i progetti USA

Standish Report 2003

Standish CHAOS reports

Standish figures 1994-2010



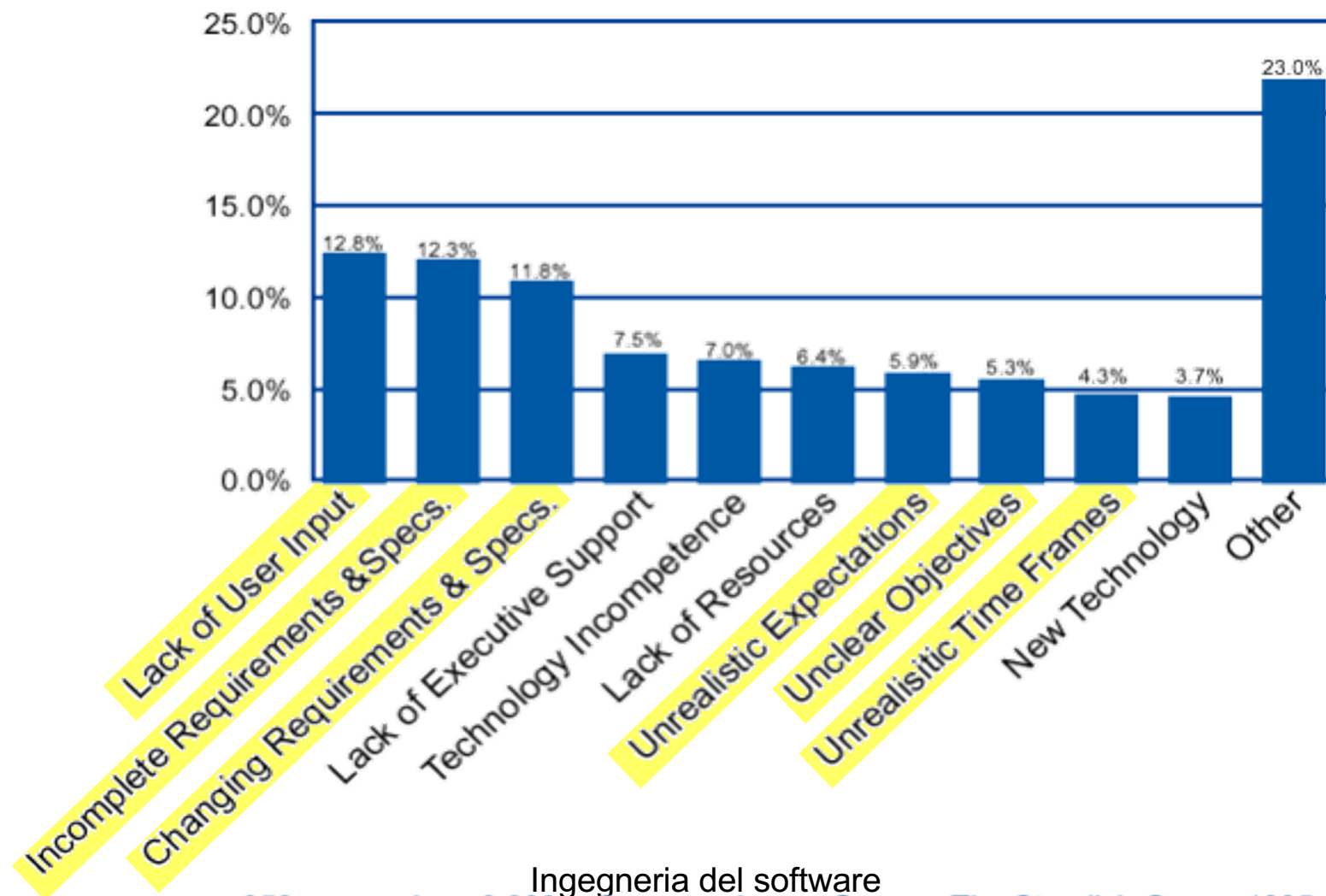
Agile vs waterfall (CHAOS 2015)

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011-2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000

Perché falliscono i progetti sw



Ingegneria del software
352 companies - 8,000 software projects. Source: The Standish Group, 1995

Perché falliscono i progetti sw: i rischi

Quali sono i rischi principali di chi sviluppa software?

- Mancanza di feedback da parte del cliente/utente
- Turnover dello staff e in particolare del team di sviluppo
- Realizzare funzioni non richieste
- Ritardi nella consegna
- Superare il budget di progetto
- Realizzare un sistema inusabile
- Realizzare un sistema incapace di funzionare insieme con altri sistemi esistenti

Turnover dello staff

Durate media degli impieghi (2017):

Facebook 2.02 anni

Google 1.90 anni

Oracle 1.89 anni

Apple 1.85 anni

Amazon 1.84 anni

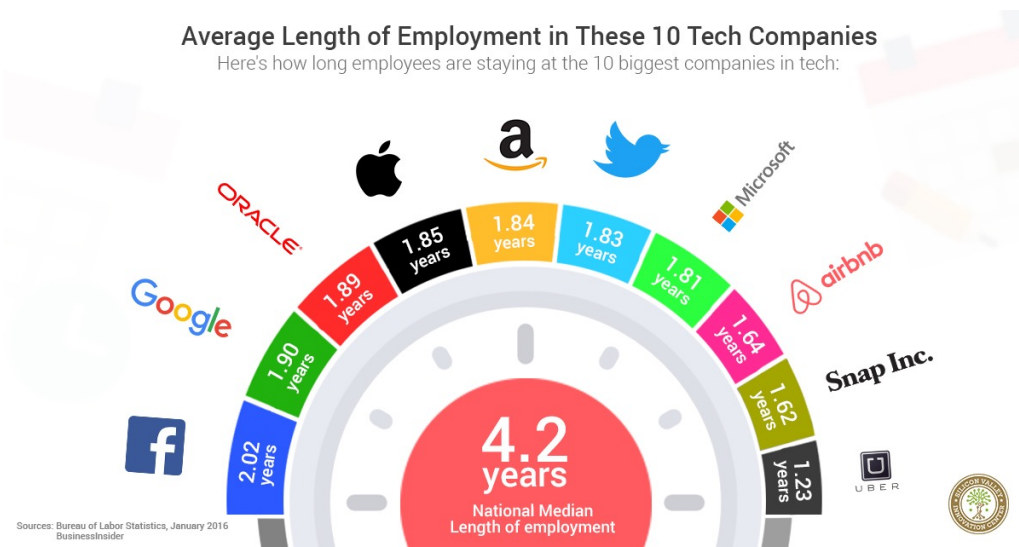
Twitter 1.83 anni

Microsoft 1.81 anni

AirBnb 1.64 anni

Snap Inc. 1.62 anni

Uber: 1.23 anni



Fonte: http://www.businessinsider.com/employee-retention-rate-top-tech-companies-2017-8?IR=T&utm_content=buffer5cb9&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer

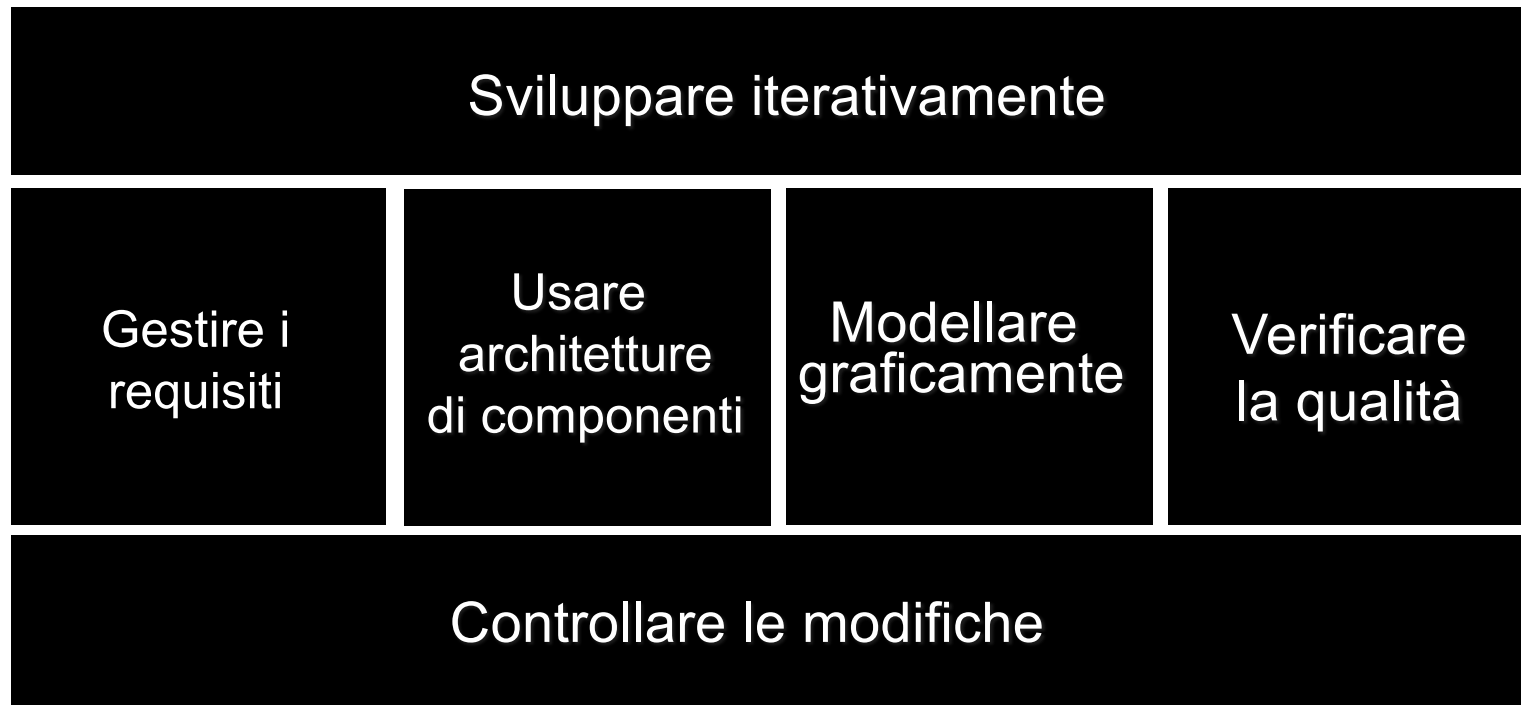
Discussione

Come si costruisce un prodotto software?

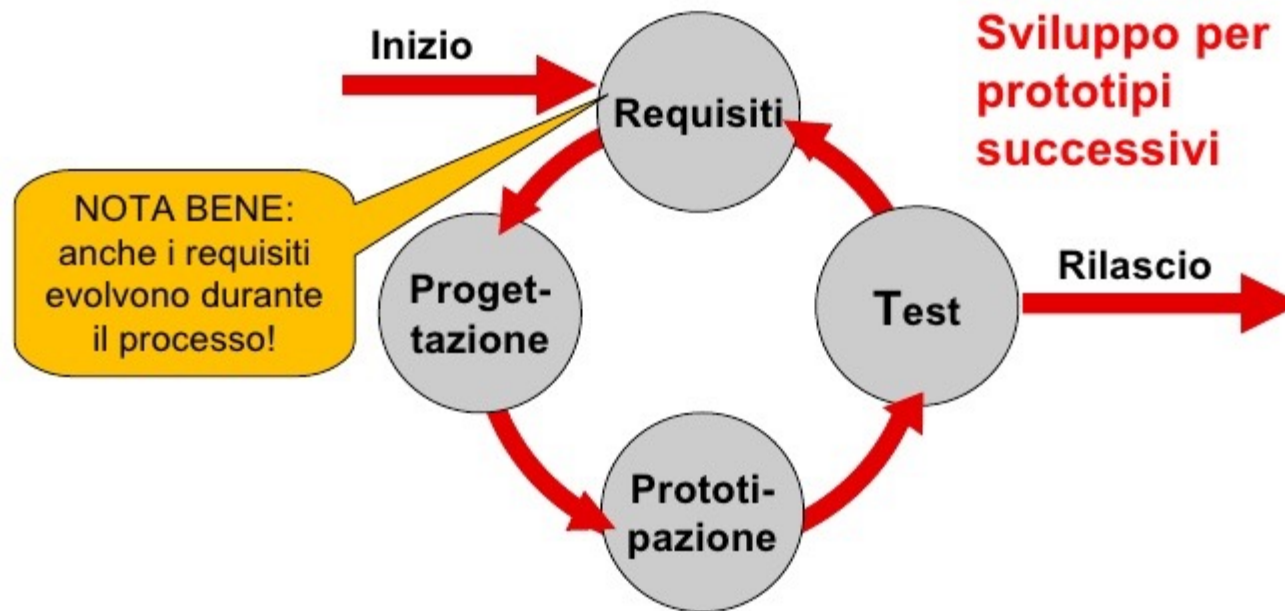
Come si misura?



Principi guida dello sviluppo software



Sviluppare iterativamente



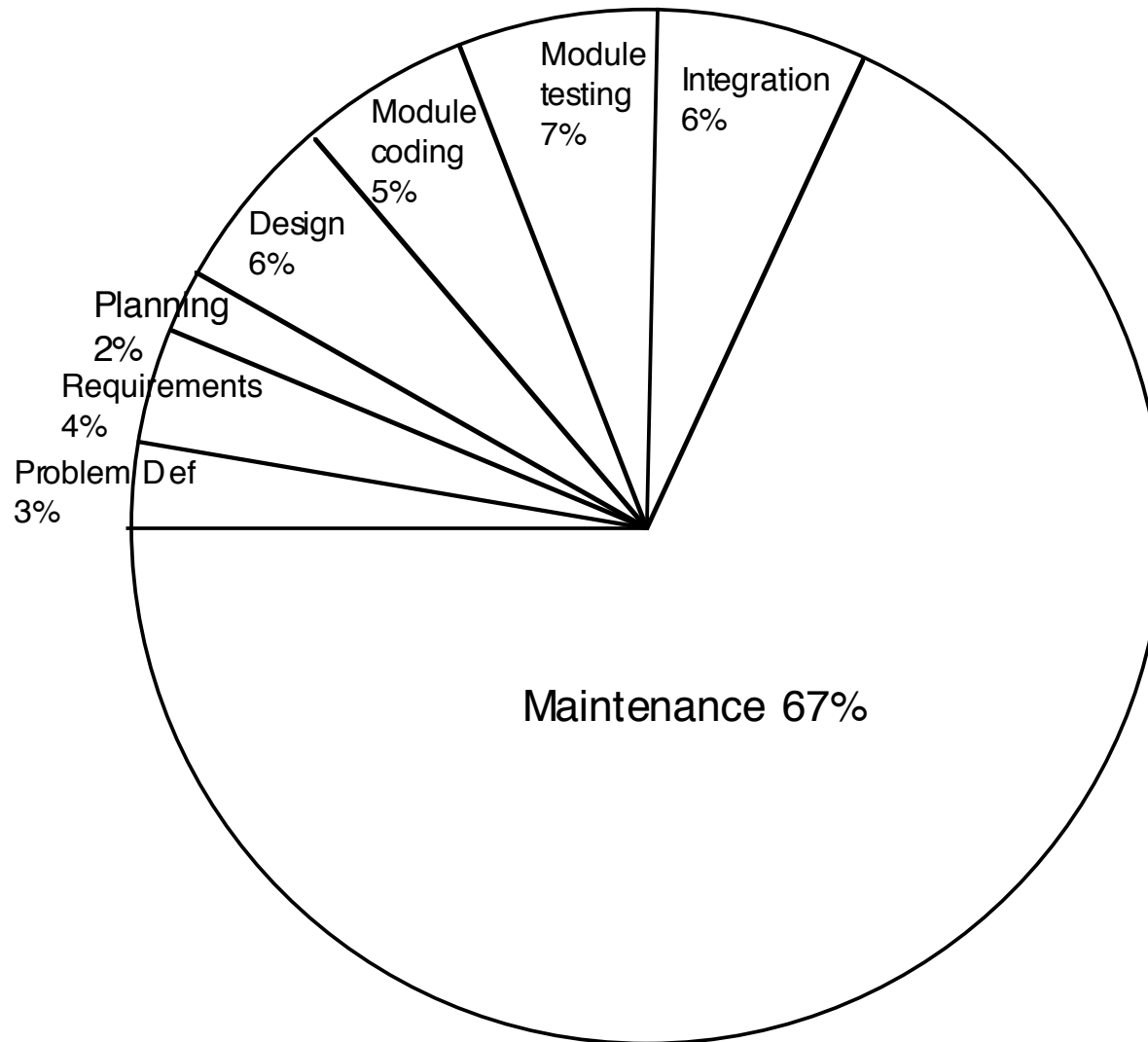
Il ciclo di vita del software

- Requisiti: analisi e specifica
- La progettazione: modellazione dell'architettura e dei singoli componenti
- La codifica ed il debugging
- Il testing e la verifica
- Il deployment (= la messa in opera)
- La manutenzione

I costi del software

- A causa dell' impatto dei rischi, i costi software spesso **dominano** i costi di produzione di un sistema; in particolare, i costi sw sono spesso maggiori dei costi dell' hardware sottostante
- **È più costoso mantenere il software che svilupparlo**: nel caso di sistemi con vita duratura, i costi di manutenzione sono un multiplo dei costi di sviluppo (es.: 3 volte)
- L' ingegneria del software si preoccupa di produrre software con costi “accettabili”

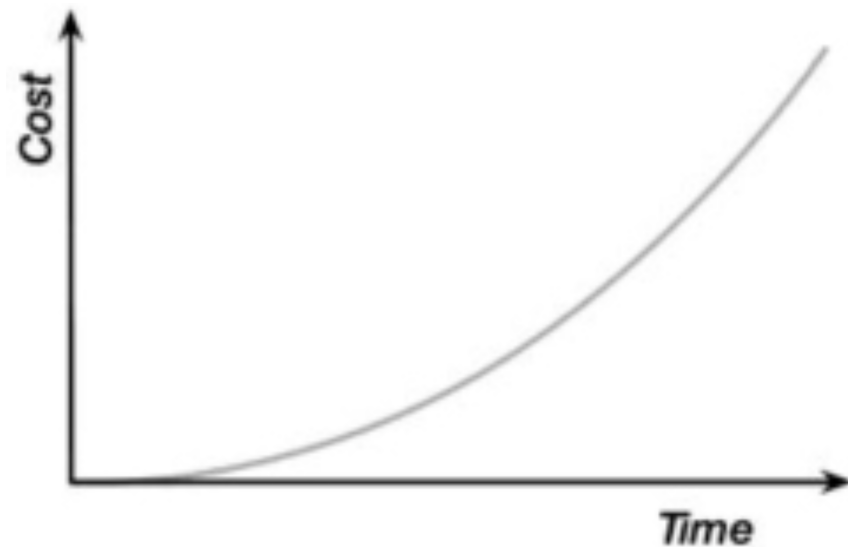
Costi di Sviluppo (Boehm citato da Schach)



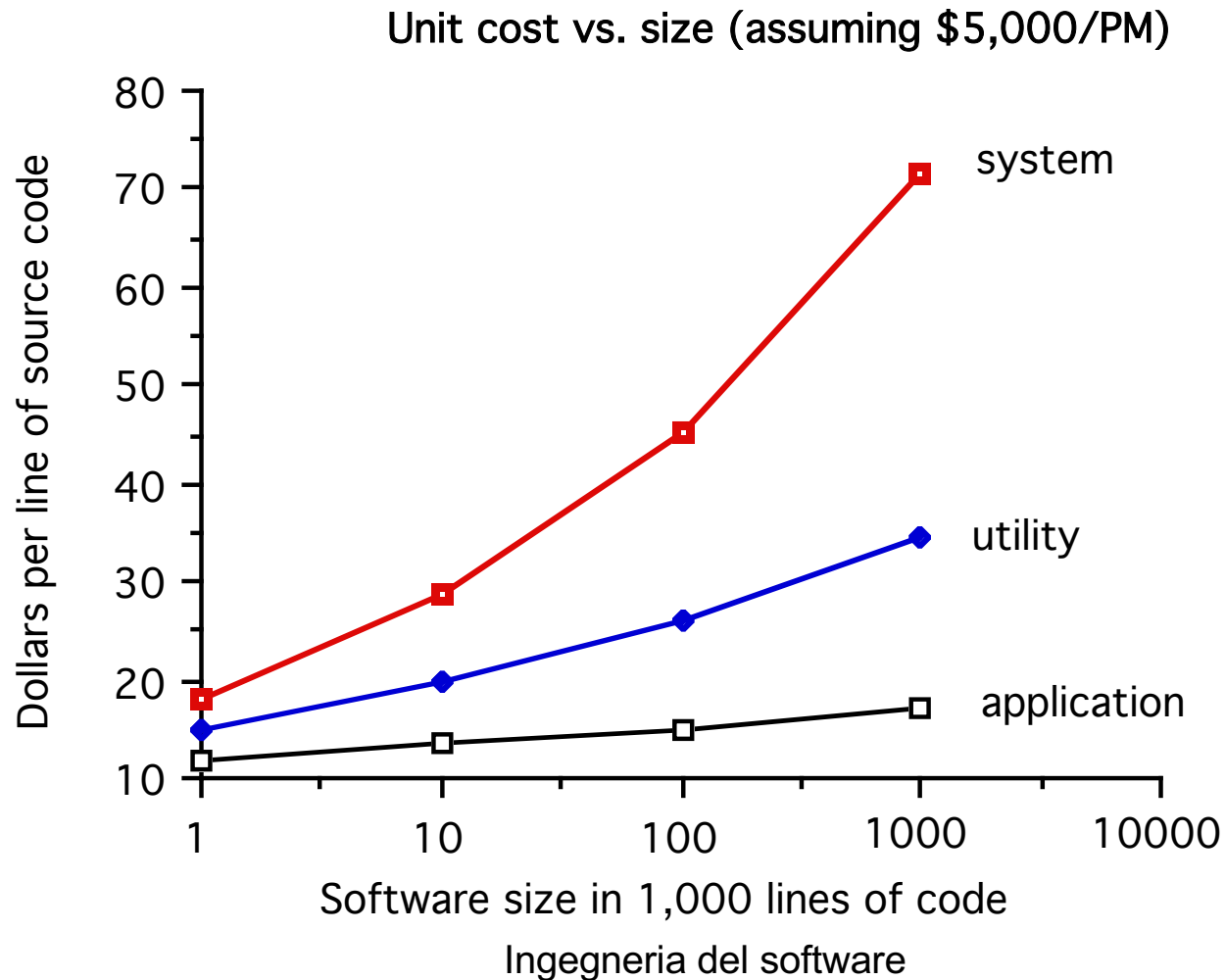
Il costo del cambiamento

Se accettiamo questo grafico, che mostra che ogni cambiamento nel sw è tanto più costoso quanto più tardi avviene, allora occorre prendere tutte le decisioni importanti il più presto possibile all'inizio o nelle fasi iniziali dello sviluppo (modello waterfall)

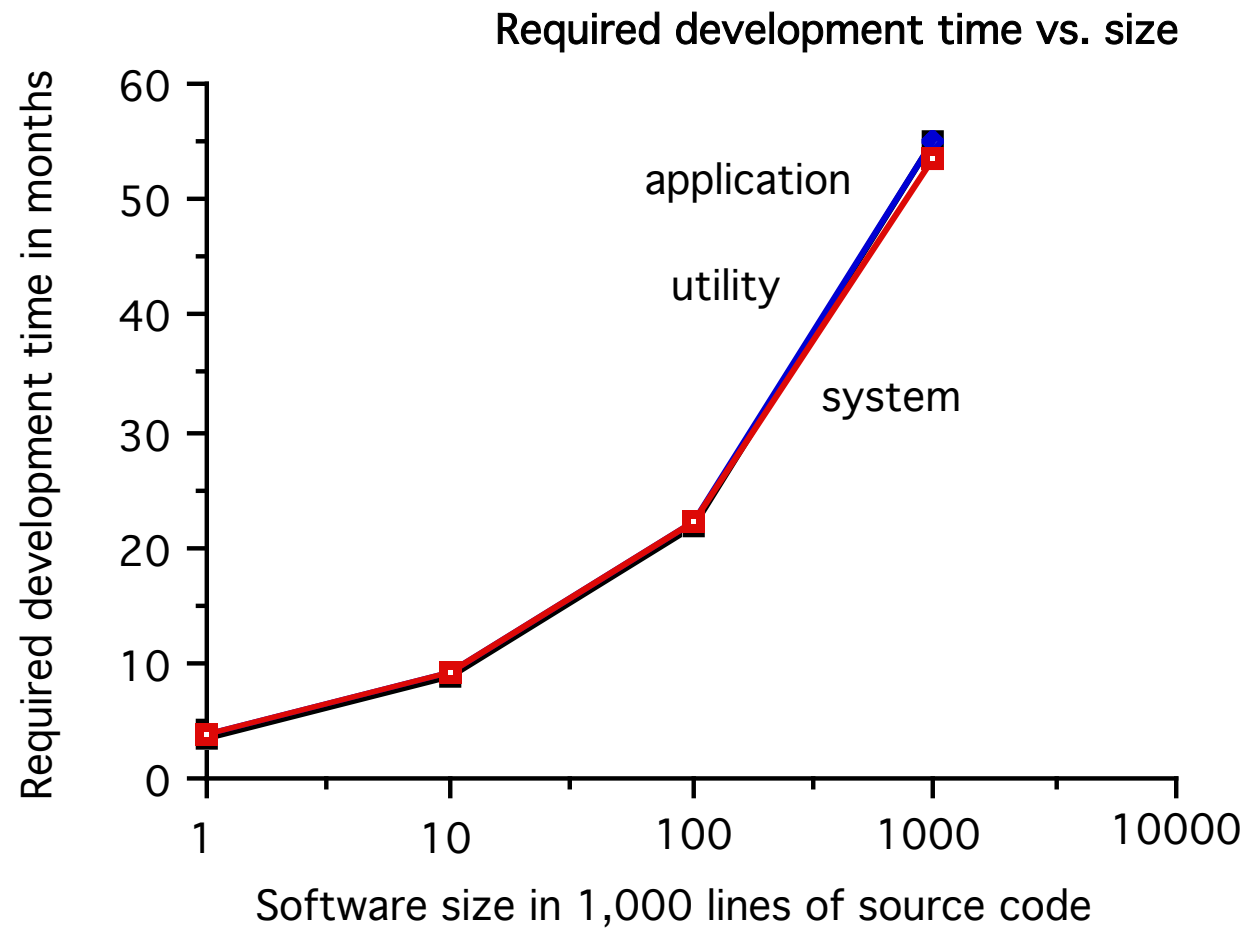
Ma è davvero così?



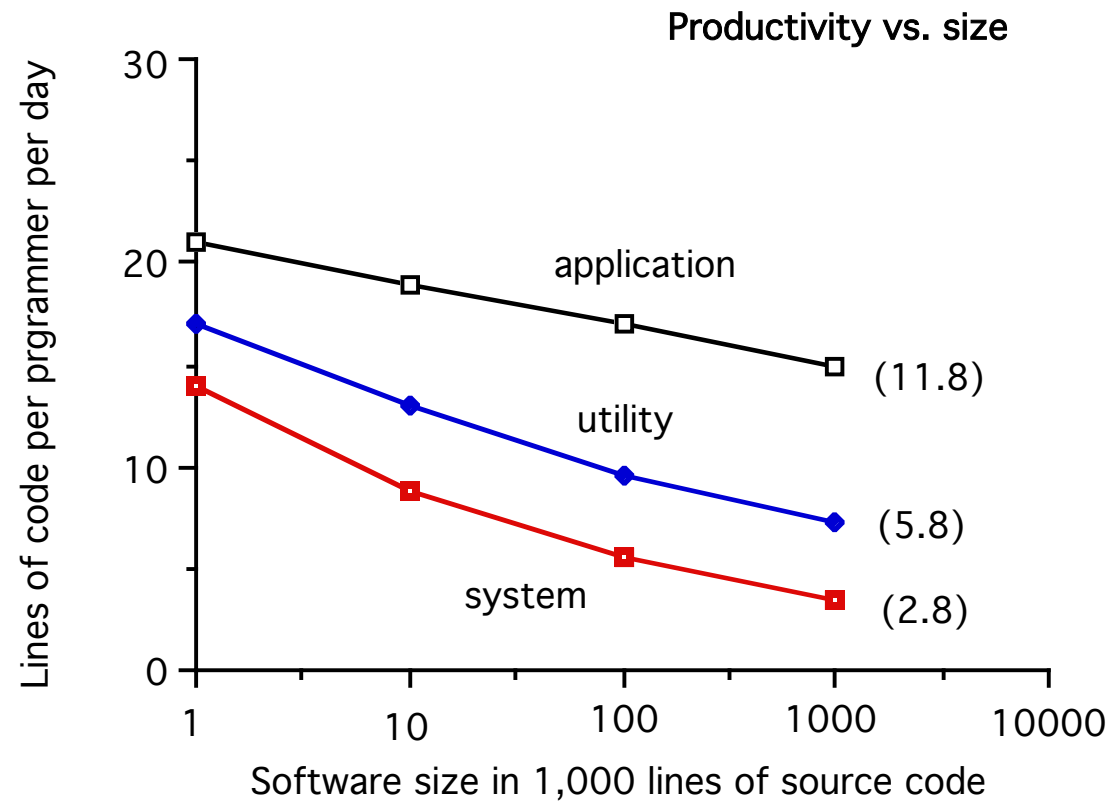
Costo per linea di codice



Durata



Produttività



Fare la cosa giusta

Di tutte le funzionalità di un'applicazione sw:

- Il 7% è usato continuamente
- Il 13% è usato spesso
- Il 16% è usato saltuariamente
- Il 19% è usato raramente
- Il 45% non è mai usato

Fonte: Standish Group, Chaos report 2002

La manutenzione

- Tutti i prodotti hanno bisogno di **manutenzione** a causa del **cambiamento**
- I tipi principali di manutenzione:
 - **Perfettiva o preventiva** (65%): migliorare il prodotto
 - **Adattiva** (18%): rispondere a modifiche ambientali
 - **Correttiva** (17%): correggere errori trovati dopo la consegna

Il mondo cambia continuamente
La manutenzione è “normale”

Attributi dei prodotti software

- Attributi **esterni** (visibili all'utente)
 - Costo (e tipo di licenza)
 - Prestazioni
 - Garanzia
- Attributi **interni** (visibili ai progettisti)
 - Dimensione (*size*)
 - Sforzo di produzione (*effort*)
 - Durata della produzione (dall'inizio alla consegna)
 - Mantenibilità
 - Modularità

Gli standard per lo sviluppo del software

Standard principali software engineering IEEE

- IEEE 610 Standard glossary sw engineering
- IEEE 828 Sw configuration management
- IEEE 829 Sw test documentation
- IEEE 830 Recommended practice for sw Requirements Specifications
- IEEE 1008 Sw unit testing
- IEEE 1219 Sw maintenance
- IEEE 1471 Recommended practice for sw Architectural Descriptions
- IEEE 1517 Sw reuse processes

II SWEBOK: sw engineering Body of Knowledge

Knowledge areas SWEBOK 3.0

Table I.1. The 15 SWEBOK KAs

Software Requirements

Software Design

Software Construction

Software Testing

Software Maintenance

Software Configuration Management

Software Engineering Management

Software Engineering Process

Software Engineering Models and Methods

Software Quality

Software Engineering Professional Practice

Software Engineering Economics

Computing Foundations

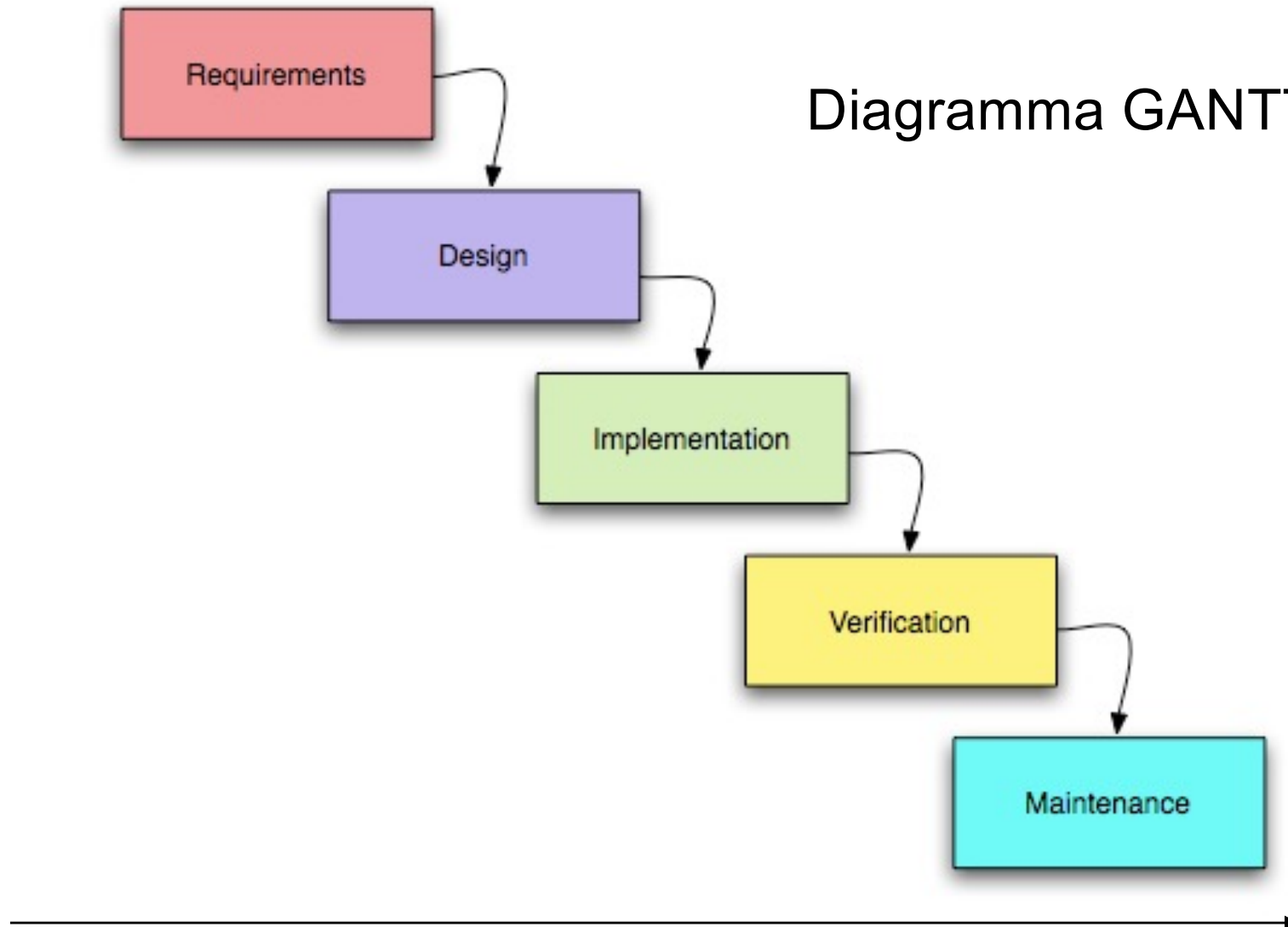
Mathematical Foundations

Engineering Foundations

Le attività di sviluppo

- Le attività di sviluppo del software differiscono in funzione dell'organizzazione che sviluppa e del sw da produrre, ma di solito includono:
 - **Specifica** delle funzionalità richieste (requisiti)
 - **Progetto** della struttura modulare e delle interfacce
 - **Implementazione**: codifica moduli e integrazione
 - **Verifica e validazione**
 - **Evoluzione** e manutenzione
- Per poterle gestire vanno **esplicitamente** modellate

Diagramma GANTT



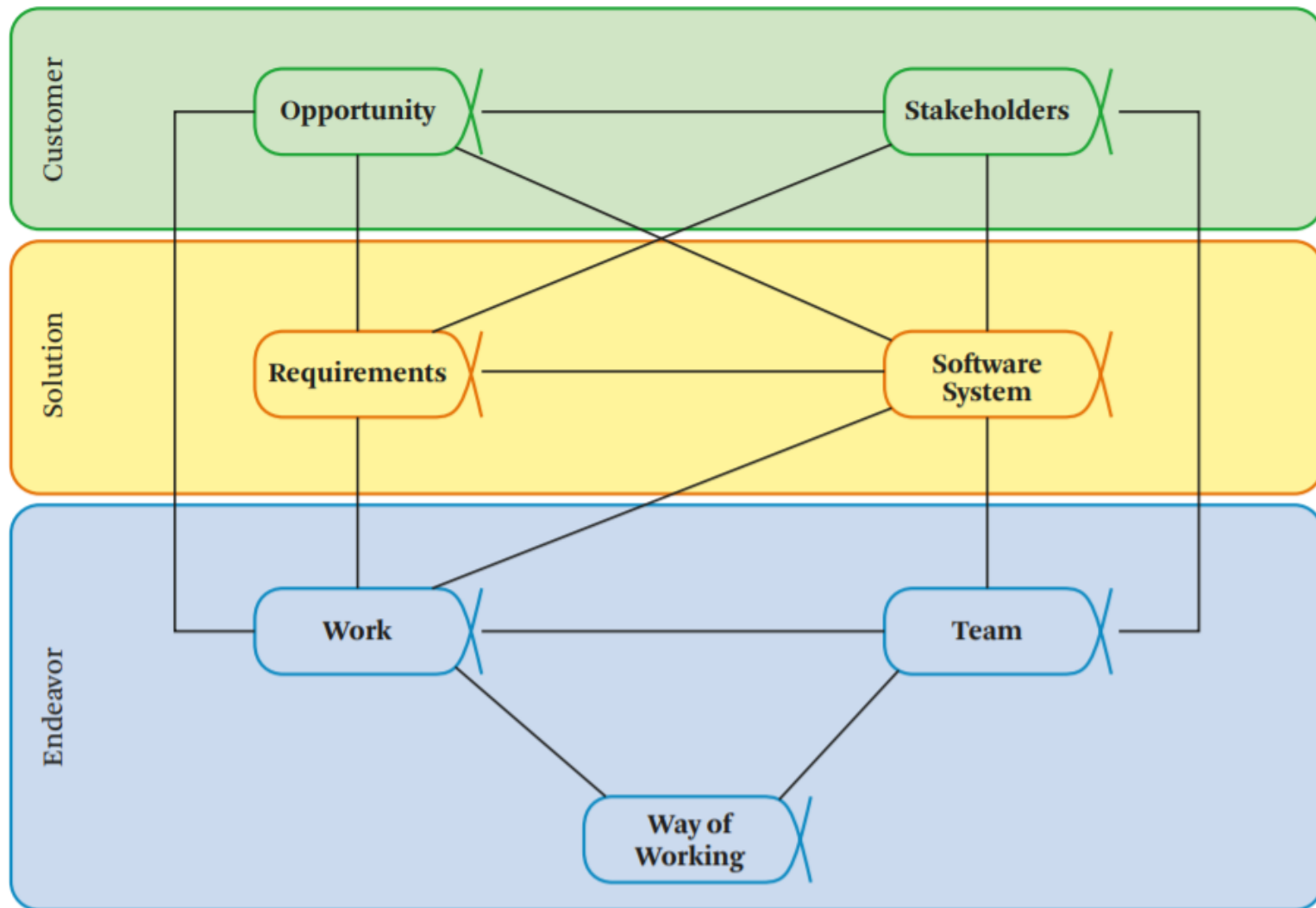
Il processo di sviluppo del sw

- **Processo software**: insieme dei ruoli, delle attività e dei documenti necessari per creare un sistema software

Esempi: ruoli attività documenti

- Esempi di **ruoli**: stakeholder, progettista, sviluppatore, tester, manutentore, ecc.
- Esempi di **attività**: programmare, testare, fare una riunione, fare una demo, documentare
- Esempi di **documenti**: codice sorgente, codice eseguibile, specifica, commenti, risultati di test, ecc.

Essence: verso una teoria dell'ingegneria del sw



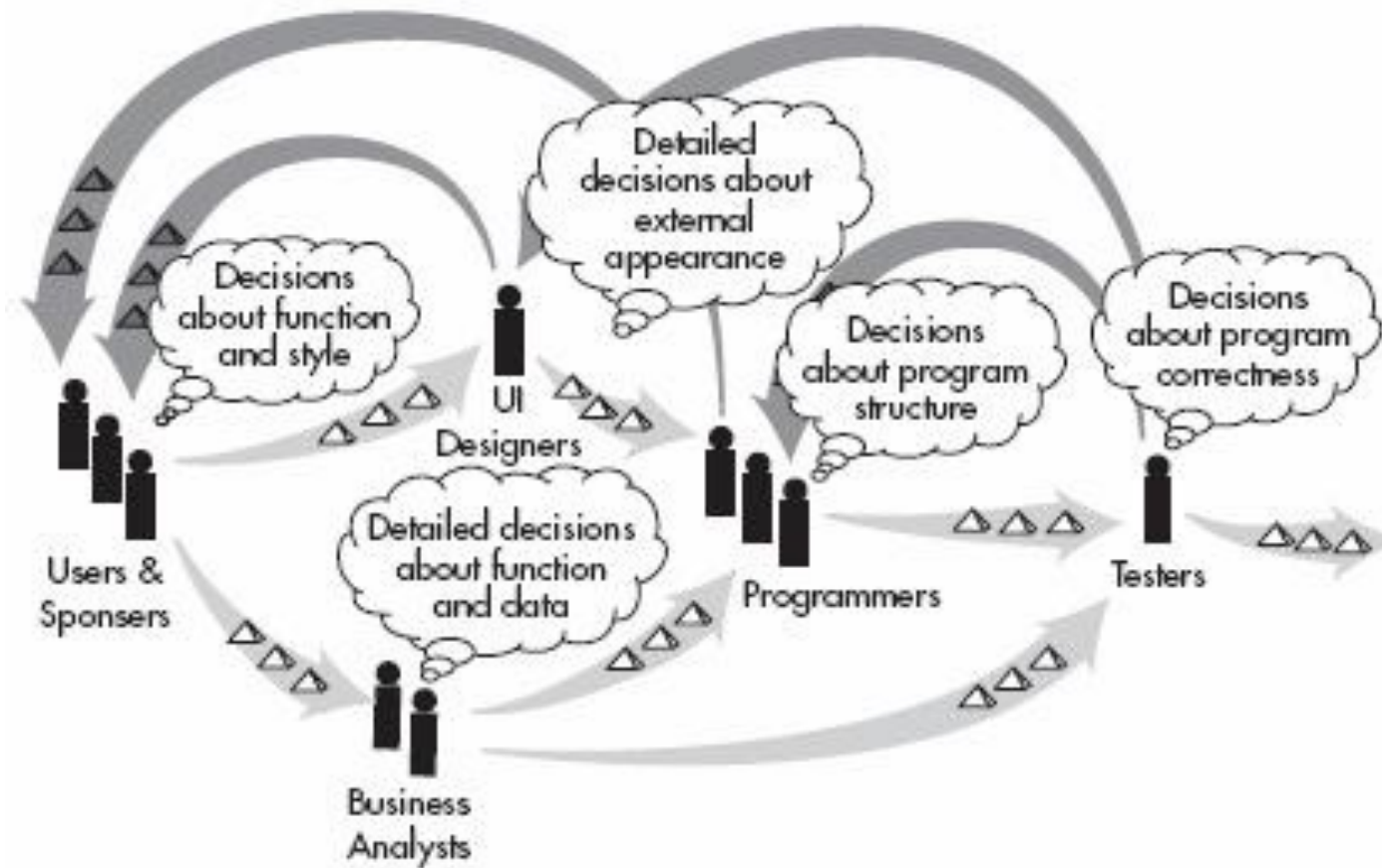
Parti interessate (stakeholders)

Tipi di stakeholders

- Progettisti professionisti
- Management
- Personale tecnico
- Decisori
- Utenti
- Finanziatori
- ...

Ad ogni stakeholder corrisponde almeno uno
specifico **punto di vista** (view) e varie
decisioni

Decisioni degli stakeholders

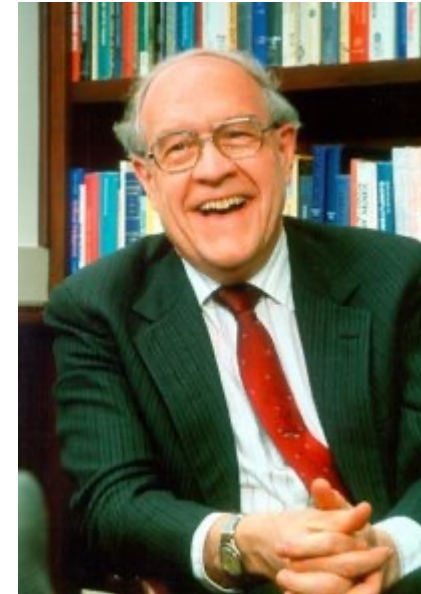


Miti e leggende dell'ingegneria del sw

- Il "silver bullet" è il proiettile d'argento che uccide i lupi mannari
- "*Trovare un silver bullet*" è sinonimo di "trovare una soluzione finale" ad un problema
- Costruire software è difficile: qual è il silver bullet dell'ingegneria del sw?

Fred Brooks

- Fred Brooks, premio Turing 1999, fu progettista del sistema operativo IBM 360, usato anche per la missione Apollo (cioè sulla Luna!)
- Dalle sue esperienze trasse spunto per scrivere il libro “The Mytical Man Month” e l’articolo “No silver bullet”



Miti e leggende

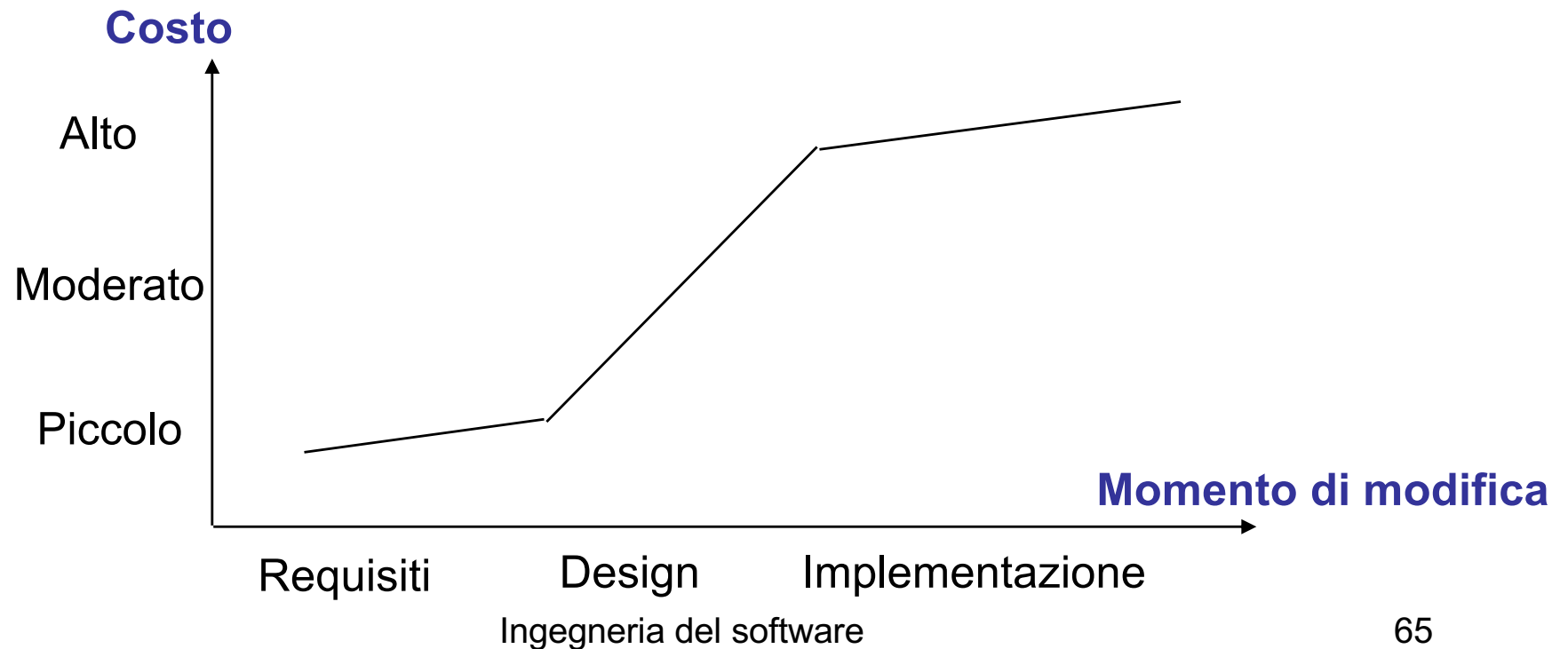
- Se il progetto ritarda, **possiamo aggiungere programmatori e rispettare la consegna**
 - **Legge di Brooks: “Aggiungere personale ad un progetto in ritardo lo fa ritardare ancor di più”**

Miti e leggende

- Per cominciare a scrivere un programma, basta un'idea generica dei suoi obiettivi - **ai dettagli si pensa dopo**
 - **La cattiva definizione della specifica dei requisiti è la maggior causa di fallimenti progettuali**

Miti e leggende

- Se i requisiti di un progetto cambiano, non è un problema tenerne conto perché il **software è flessibile**



Sommario

- Produrre software è costoso
- La produttività dell'industria del sw è bassa
 - Le consegne sono spesso in ritardo
 - I costi software spesso sfiorano il budget
 - La documentazione è inadeguata
 - Il software è spesso difficile da usare
- Soluzione: migliorare il processo software

Domande di autotest

- Quali sono le fasi tipiche del ciclo di vita di un sistema software? E quelle dello sviluppo?
- Qual è la fase solitamente più costosa?
- In quale fase dello sviluppo è più pericoloso commettere un errore?
- In quale fase dello sviluppo è più semplice correggere un errore?
- Cos'è un processo di sviluppo del software?
- Quali sono i tipici documenti prodotti durante un processo software?
- Cos'è la legge di Brooks?

Lettura consigliata

F.Brooks, No Silver Bullets, *IEEE Computer*,
20:4, 1987

Riferimenti

- *Software Engineering Body of Knowledge*, IEEE, 2014
- F.Brooks, *The Mythical Man Month*, AddisonWesley, 1995
- M.Cusumano, *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know in Good Times and Bad*, Free Press, 2004
- IEEE/EIA 12207.0, "Standard for Information Technology – Software Life Cycle Processes"

Principali pubblicazioni scientifiche

- IEEE Transactions on Software Engineering
- ACM Transactions on Software Engineering and Methodology
- Int. Conference on Software Engineering

Pubblicazioni di ricerca sul sw engineering

Riviste:

- IEEE Transactions on sw engineering
- ACM Transactions on software engineering and methodology
- IEEE Software
- Empirical Software Engineering
- Automated Software Engineering
- Journal of Object Technology
- ACM SIGSOFT

Conferenze:

- International Conference of Software Engineering
- Fundamentals of Software engineering
- SPLASH
- International Conference on Software and System Process

Siti utili

- `www.sigsoft.org/seworld`
- www.computer.org/web/swebok
- `https://dokumen.tips/reader/f/guide-to-the-software-engineering-body-of-knowledge-swebok-v3`
- `swebokwiki.org/Main_Page`
- `essence.ivarjacobson.com/services/what-essence`

Domande?



Modelli di processo per lo sviluppo del software: i modelli pianificati (lineari o iterativi)



Corso di Ingegneria del Software
CdL Informatica Università di Bologna

Obiettivi di questa lezione

- Cos'è un **processo di sviluppo del software**
- Cos'è un **modello** di processo software
- Modelli **lineari**
- Modelli **iterativi**

Nella prossima:

- Modelli **agili**

Successivamente:

- Modelli di processo orientati alla **qualità**
- Modelli **open source**

Costruire software



How the customer explained it



How the Project Leader understood it



How the Analyst designed it



How the Programmer wrote it



How the Business Consultant described it



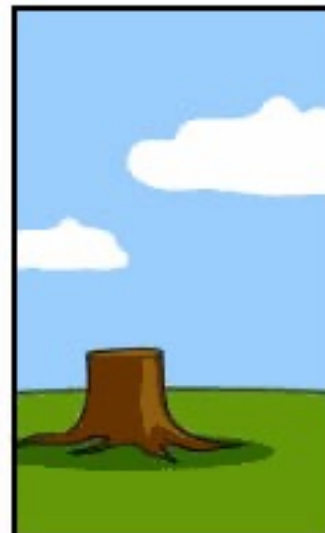
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed

Le fasi “naturali” dello sviluppo software

- Entusiasmo
- Disillusione
- Panico
- Ricerca del colpevole
- Punizione dell'innocente
- Lodi e onori a chi non si è fatto coinvolgere

Non tutti gli sviluppi si svolgono così: in quelli di successo la differenza la fanno sempre **le persone** e le regole (il **processo di sviluppo**) che seguono

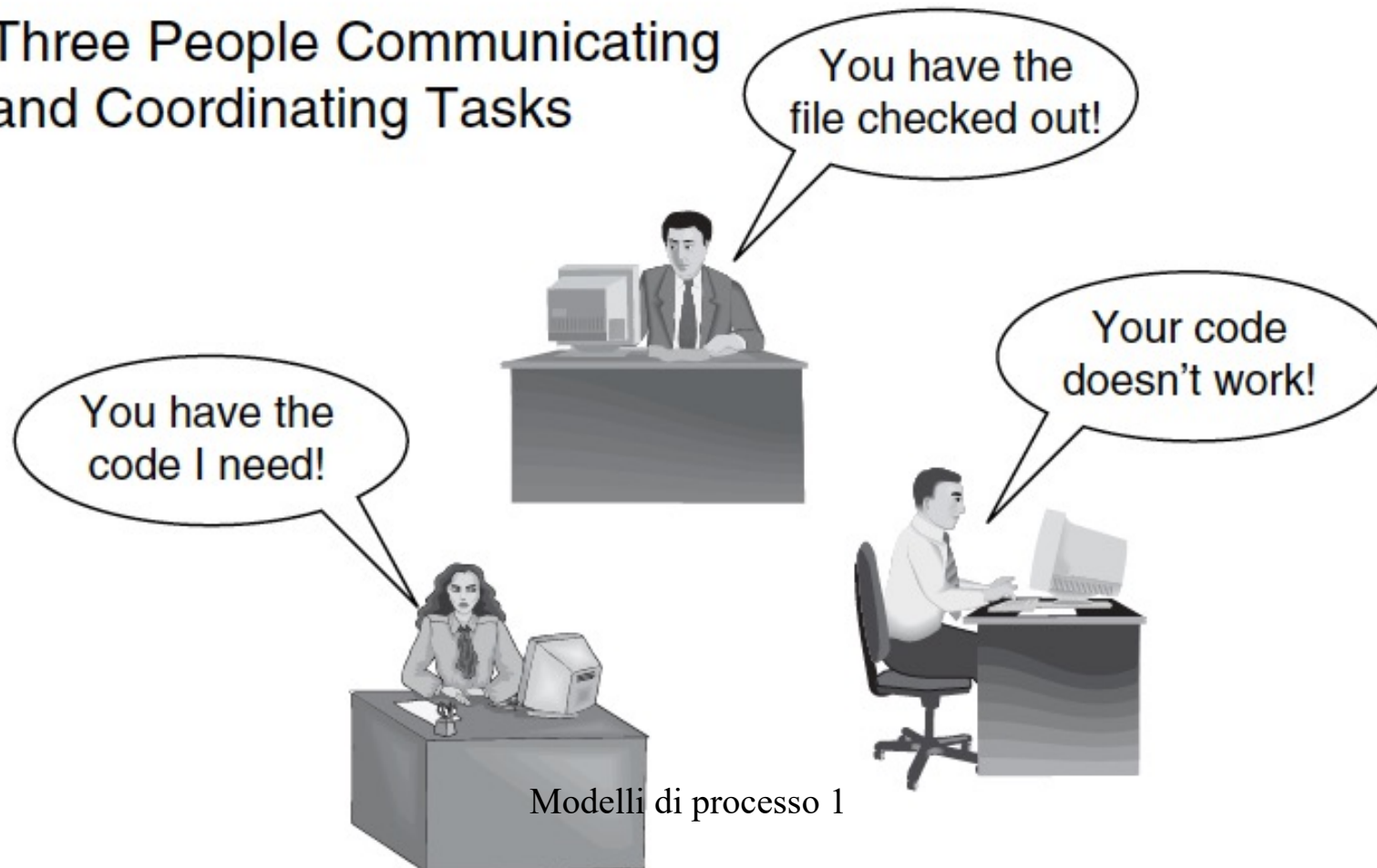
Ciclo di vita e processo di sviluppo

- Il **processo di sviluppo** è una parte del ciclo di vita del software
- Il **ciclo di vita del software** (*software lifecycle*), designa le varie fasi della vita di un software, dalla sua concezione al suo ritiro
- Di solito il processo di sviluppo inizia dalla concezione e finisce col rilascio finale e il successivo deployment

One Person
Very Little Coordination
and Communication
Overhead



Three People Communicating
and Coordinating Tasks



Modelli di processo 1

Il team include più ruoli

- Lo sviluppo in team è molto diverso dallo sviluppo “personale”
- Nel team ci sono persone con esperienze diverse, che ricoprono diversi **ruoli che hanno diverse abilità**:
 - Come progettare il prodotto software (**architetti**)
 - Come costruire il prodotto sw (**programmatori**)
 - A cosa serve il prodotto sw (**esperti di dominio**)
 - Come va fatta l'interfaccia utente (**progettisti di interfaccia**)
 - Come va controllata la qualità del prodotto sw (**testatori**)
 - Come usare le risorse di progetto (**project manager**)
 - Come riusare il software esistente (**gestori delle configurazioni**)

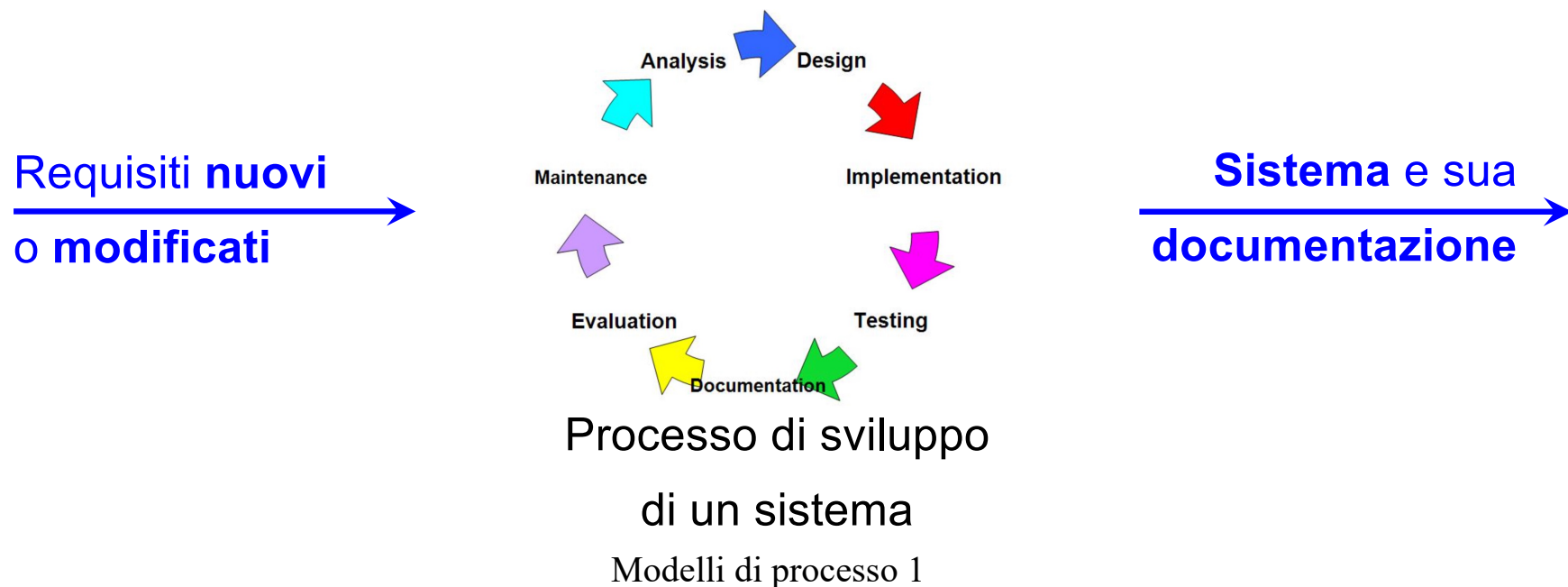
Discussione

- Se dovete creare un sistema di 100 KLOC,
 - Quante *persone* occorrono? Per quanto tempo?
 - Cosa debbono fare?
 - Come le organizzate?
 - Quali documenti debbono produrre? Quando?



Cos' è un processo di sviluppo

Un processo di sviluppo definisce **Chi** fa **Cosa**, **Quando**, e **Come**, allo scopo di conseguire un certo risultato



Perché studiare il processo di sviluppo del sw?

- I sistemi software che costruiamo devono risultare affidabili e sicuri: il processo di sviluppo del software influenza tali **qualità**
- Esistono parecchi modelli di processi software, adatti a prodotti, organizzazioni e mercati **diversi**
- Alcuni **strumenti** sw di sviluppo sono efficaci solo nell'ambito di processi specifici
- Il processo di sviluppo del software impatta l'organizzazione che lo sviluppa
- L'organizzazione che esegue lo sviluppo impatta la struttura del prodotto (**legge di Conway**)

Legge di Conway

Le organizzazioni che progettano sistemi ne progettano la struttura riproducendo le proprie strutture comunicative (es. l'organigramma)

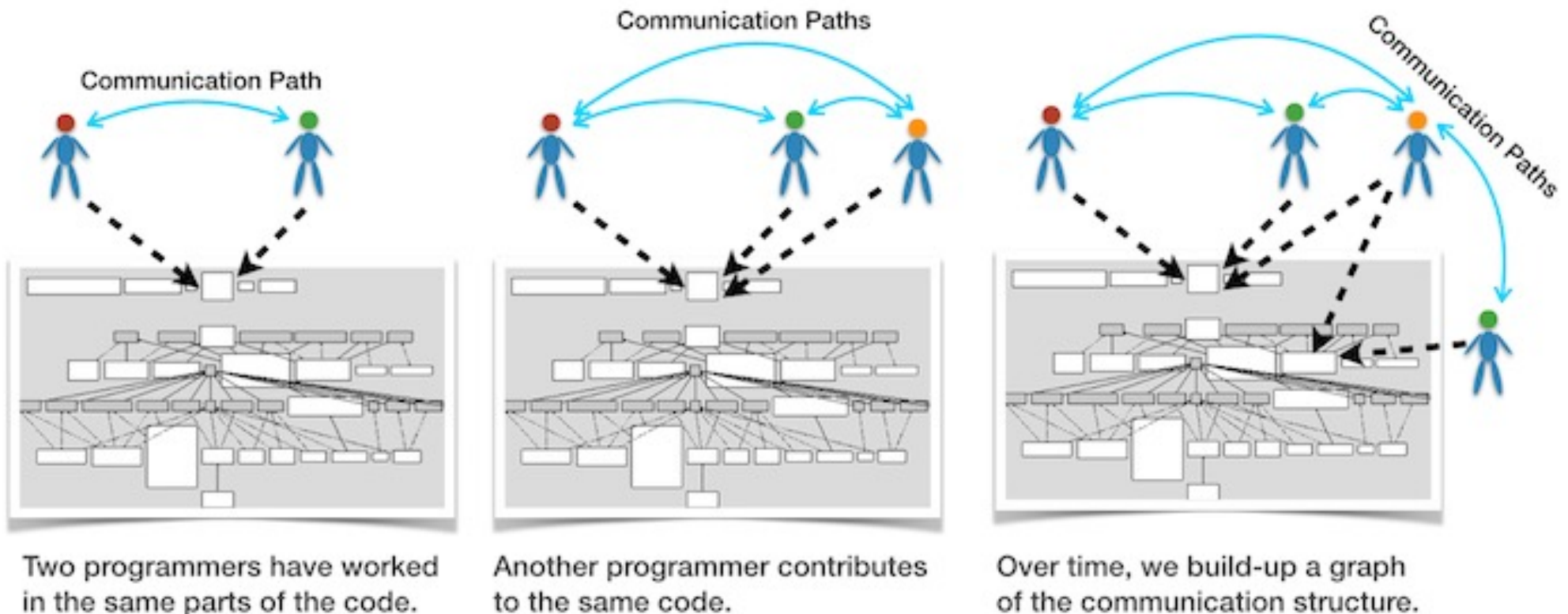
Esempio: se 4 team collaborano a costruire un compilatore, la struttura finale sarà su 4 processi in pipeline

Principio: alcune importanti proprietà di un sistema dipendono dal suo processo di costruzione

Conseguenze della legge di Conway

La legge di Conway ha per conseguenza che gli sviluppatori che lavorano sugli stessi componenti devono poter comunicare senza ostacoli.

Ovvero, devono essere vicini da un punto di vista organizzativo

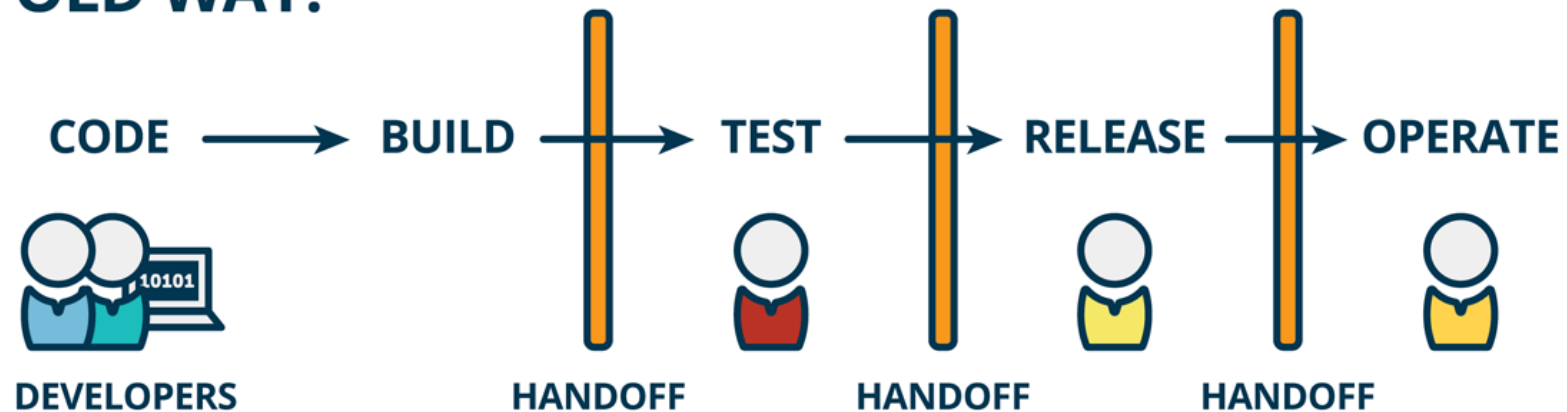


Conseguenze della legge di Conway

- Se le parti di un'organizzazione (team, dipartimenti, divisioni) non riflettono le parti essenziali del prodotto/servizio, o se le loro relazioni non si riflettono nelle relazioni tra le parti del prodotto/servizio, allora il progetto avrà problemi
- Occorre assicurarsi che l'organizzazione di sviluppo e poi di produzione sia compatibile con l'architettura del prodotto
- Esempio: un'organizzazione che produce un portale o sito la cui struttura e contenuto rispecchiano gli interessi interni dell'organizzazione piuttosto che i bisogni degli utenti
- Morale: per innovare mediante i servizi ICT (interni o esterni) cambiate prima la vostra organizzazione, altrimenti è una battaglia persa!

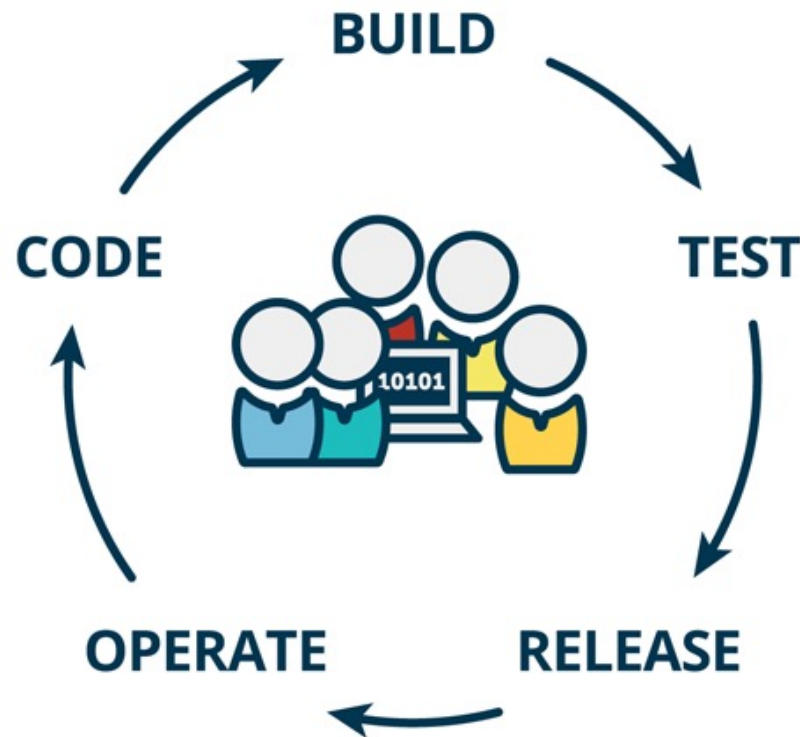
Conseguenze della legge di Conway

OLD WAY:



Conseguenze della legge di Conway

NEW WAY:



Il software è un costrutto sociale

- **Il software è il prodotto di un processo sociale**, che coinvolge molte persone con ruoli e interessi diversi
- Inoltre, spesso, quando un software viene messo in opera modifica i rapporti sociali tra le persone – ad esempio in una banca, o in una pubblica istituzione
- Molte qualità del software si possono giudicare solo in termini «sociali»

Processo software

- Un processo di sviluppo del software (o “*processo software*”) è un **insieme di attività di persone** che costruiscono un **prodotto software**
- Il prodotto può essere costruito **da zero** o mediante **riuso di software esistente** (*asset*) che viene modificato o integrato

Categorie di processi

- **Programming in-the-small:**
uno sviluppatore, un modulo (es. edit-compile-debug)
- **Programming in-the-large:**
uno sviluppatore, più moduli, più versioni, più configurazioni
(es. Personal software process)
- **Programming in-the-many:**
Più sviluppatori, coordinati mediante un **processo di sviluppo**
(es. Scrum)

Processo minimale: “programma e debugga”

Scrivere un programma e rimuovere gli errori

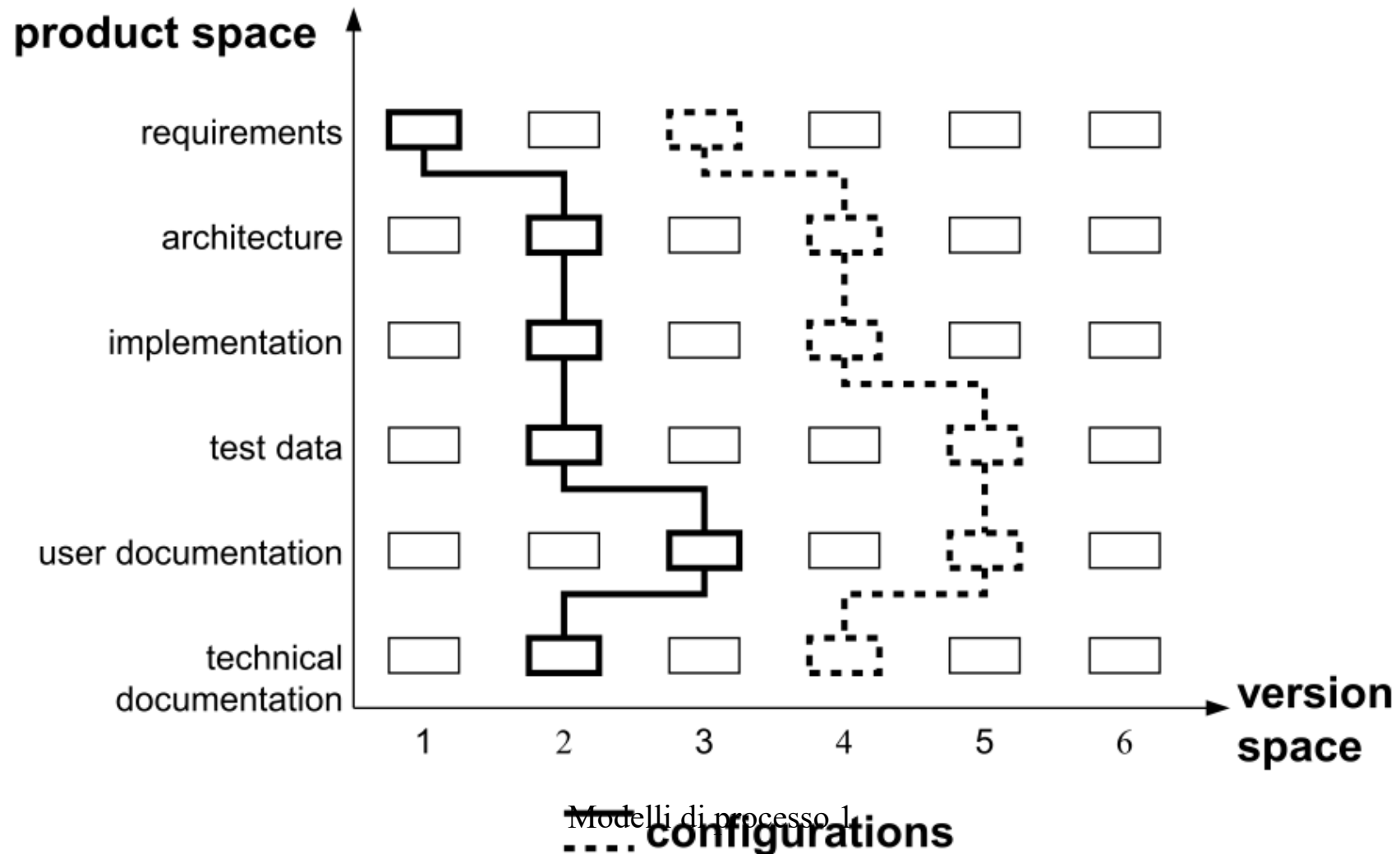
- **Programmare è un attività personale**
di solito non si definisce una sequenza generica di passi “creativi” della programmazione
- **Il debugging è una forma di verifica**
per scoprire e rimuovere errori nel programma

Il processo edit-compile-debug



- **Ciclo molto veloce, feedback rapido**
- **Disponibilità di molti strumenti**
- **Specializzato per la codifica**
- **Non incoraggia la documentazione**
- **Il processo non scala: in-the-large, in-the-many**
- **Ingestibile durante la manutenzione**
- **Il debugging ha bisogno di un processo specifico**

Programming in-the-large: moduli, versioni, configurazioni



Programming in the many

- In una grande azienda lo sviluppo sw è spesso suddiviso tra più team
- Occorre coordinare i team, e di solito si usa una struttura sociale gerarchica
- In ciascun team il Product Owner è la figura di riferimento che definisce gli obiettivi di sviluppo del team



©Project Management Institute. All rights reserved.

<https://www.pmi.org/disciplined-agile/process/program-management/team-structure-of-a-large-program>

Modelli di processo

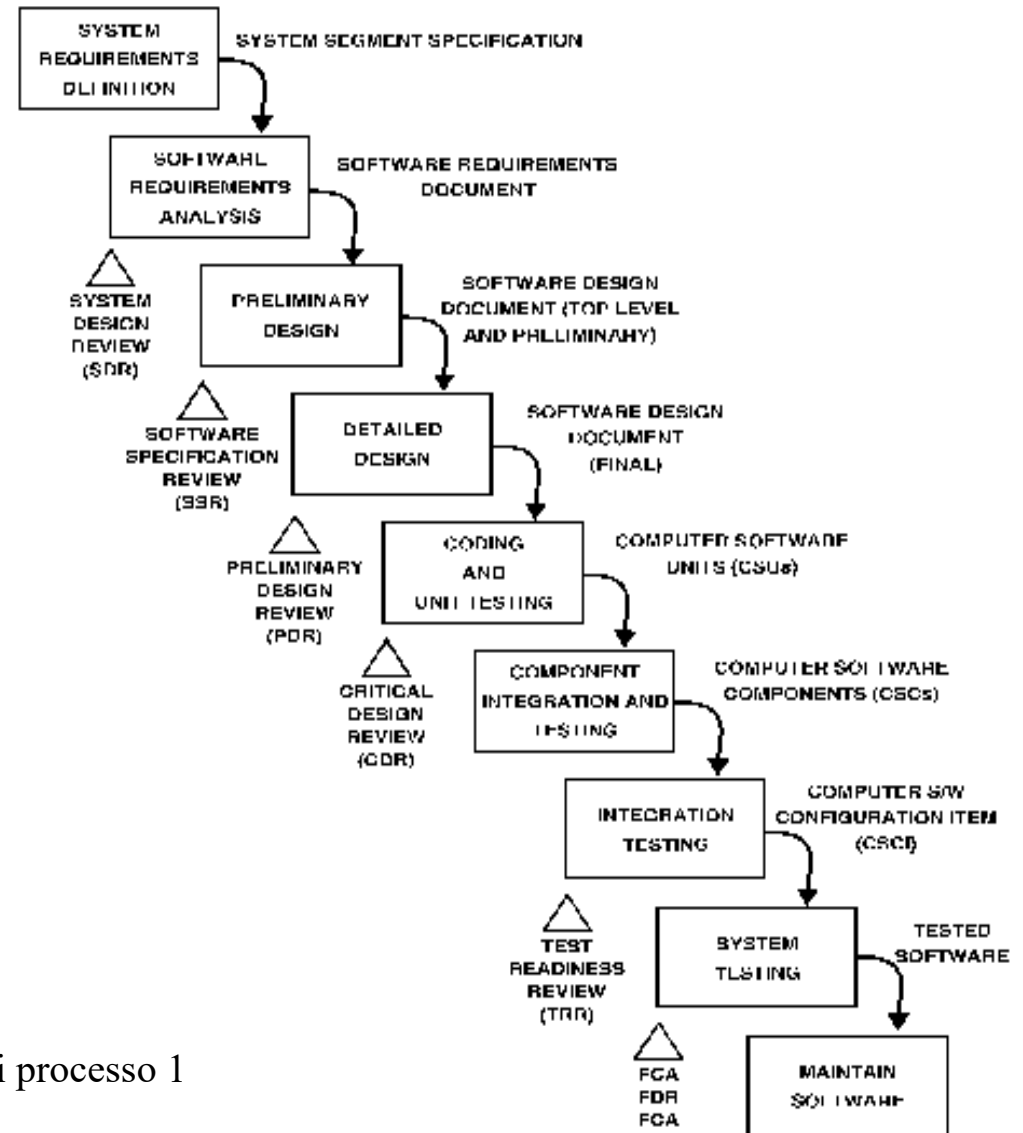
- Un **processo sw** è un insieme strutturato di attività necessarie per sviluppare un sistema sw:
 - i **ruoli**, le **attività** e i **documenti** da produrre
- Un **modello di processo sw** è una **rappresentazione** di una **famiglia di processi**
 - Fornisce una descrizione da prospettive particolari
 - per catturare caratteristiche importanti dei processi sw
 - utili a diversi scopi, ad esempio per valutarli, criticarli o estenderli

Descrivere un processo

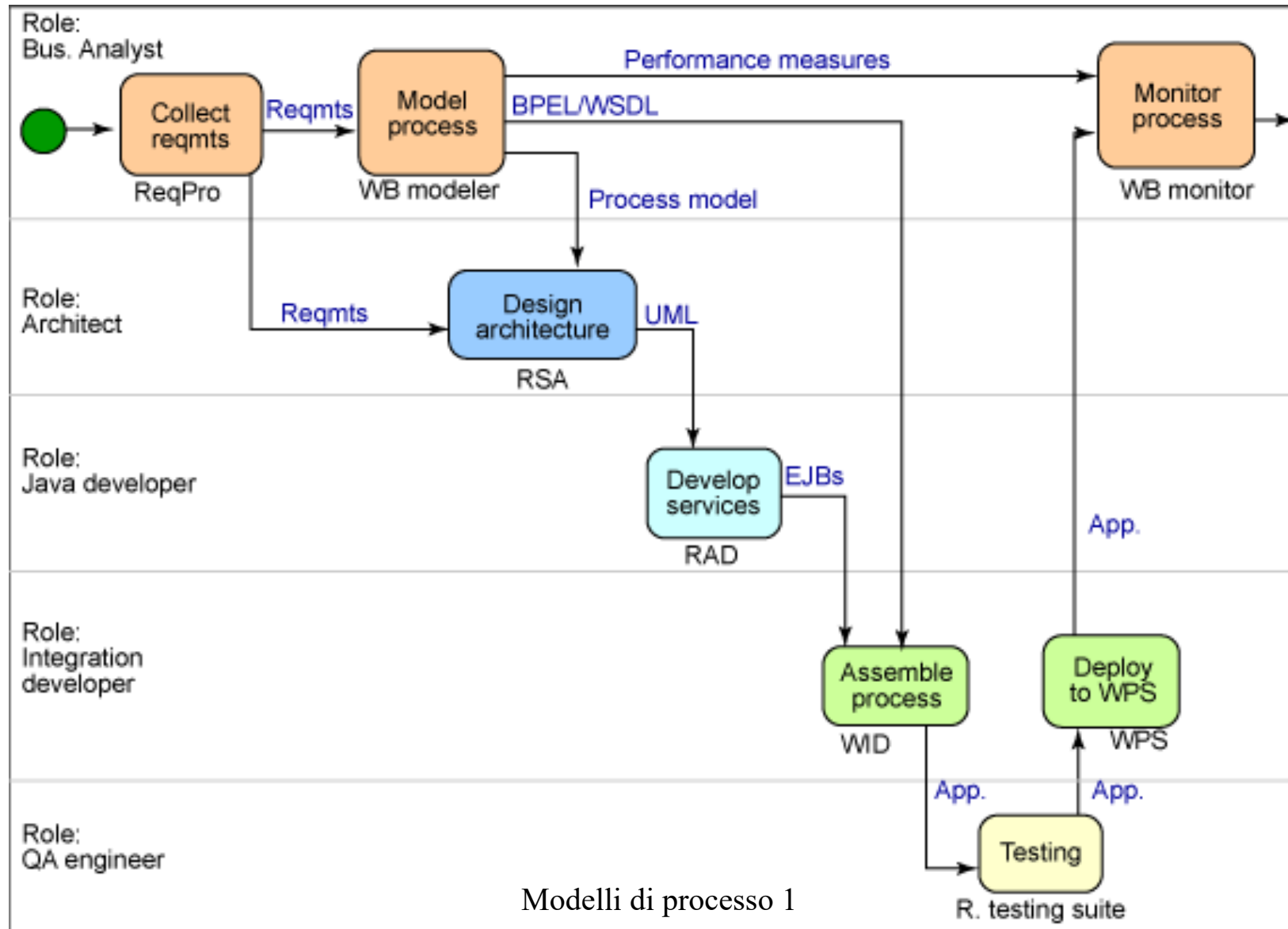
- Occorre descrivere/assegnare i **ruoli** (es. stakeholders)
- Occorre descrivere/monitorare le **attività**
- Occorre descrivere/assemblare gli **strumenti**
- Occorre descrivere/controllare i gli **eventi**
- Occorre descrivere/validare i **documenti** (es. requisiti)
- Occorre descrivere/verificare i **criteri di qualità**

Esempio

- **Modello di processo:** **waterfall**, cioè pianificato lineare
- **Processo:** istanza di un modello waterfall che viene “eseguita” per creare un sistema
 - crea una serie di artefatti come prescritto dal modello



Esempio: ruoli e strumenti in un processo a cascata (IBM WebSphere)



Perché descrivere un processo sw

Processo software:

L'insieme strutturato di attività, eventi, documenti e procedure necessari per la costruzione di un sistema software

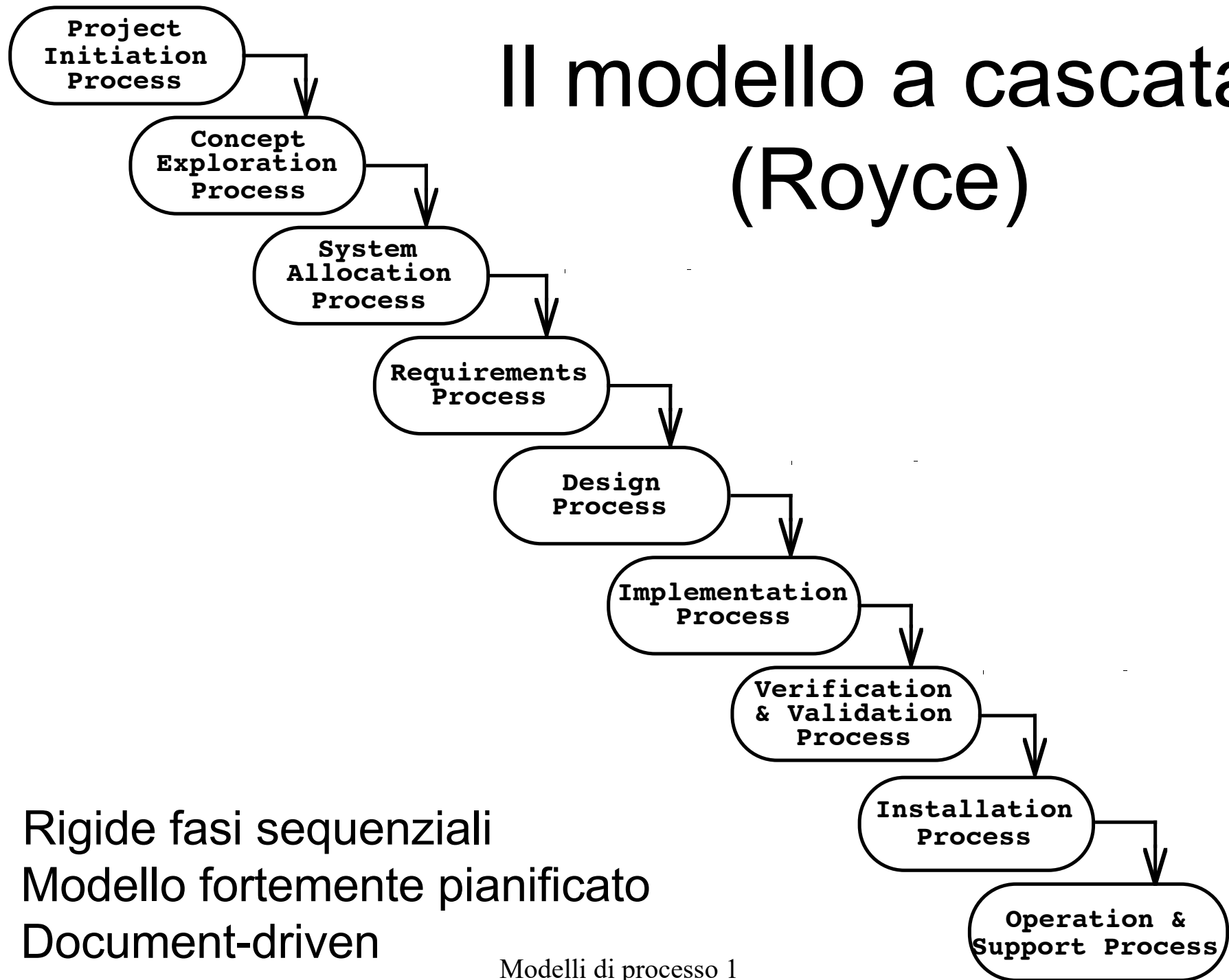
Benefici della modellazione dei processi sw:

- Ciascuno sa cosa deve fare, giorno dopo giorno
- Miglior coordinamento del team
- Accumulazione di esperienza
- Aderenza agli standard internazionali
- “**Migliora il processo per migliorare il prodotto**”

Modelli generici di processo sw

- **Modello a cascata (esempio: Waterfall)**
 - Specifica e sviluppo sono separati e distinti
- **Modello iterativo (esempio: UP)**
 - Specifica e sviluppo sono ciclici e sovrapposti
- **Modello agile**
 - Non pianificato, guidato dall'utente e dai test
- **Sviluppo formale (esempio: B method)**
 - Il modello matematico di un sistema viene trasformato in un'implementazione

Il modello a cascata (Royce)



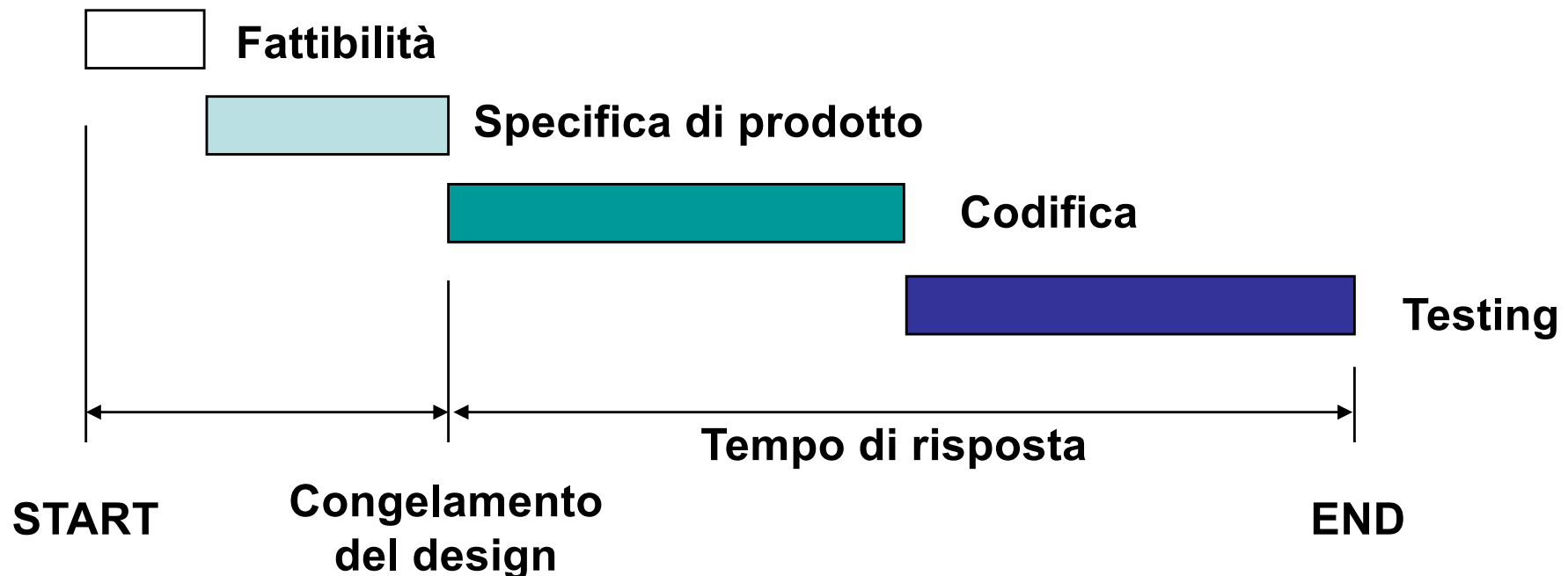
- Rigide fasi sequenziali
- Modello fortemente pianificato
- Document-driven

Modello a cascata: aspetti positivi e negativi?

- Si adatta bene a progetti con requisiti stabili e ben definiti
- Problemi:
 - Il cliente deve sapere definire i requisiti
 - Versione funzionante del software solo alla fine
 - Difficili modifiche “in corsa”
 - Fasi fortemente collegate tra loro e bloccanti

Problema dei processi lineari

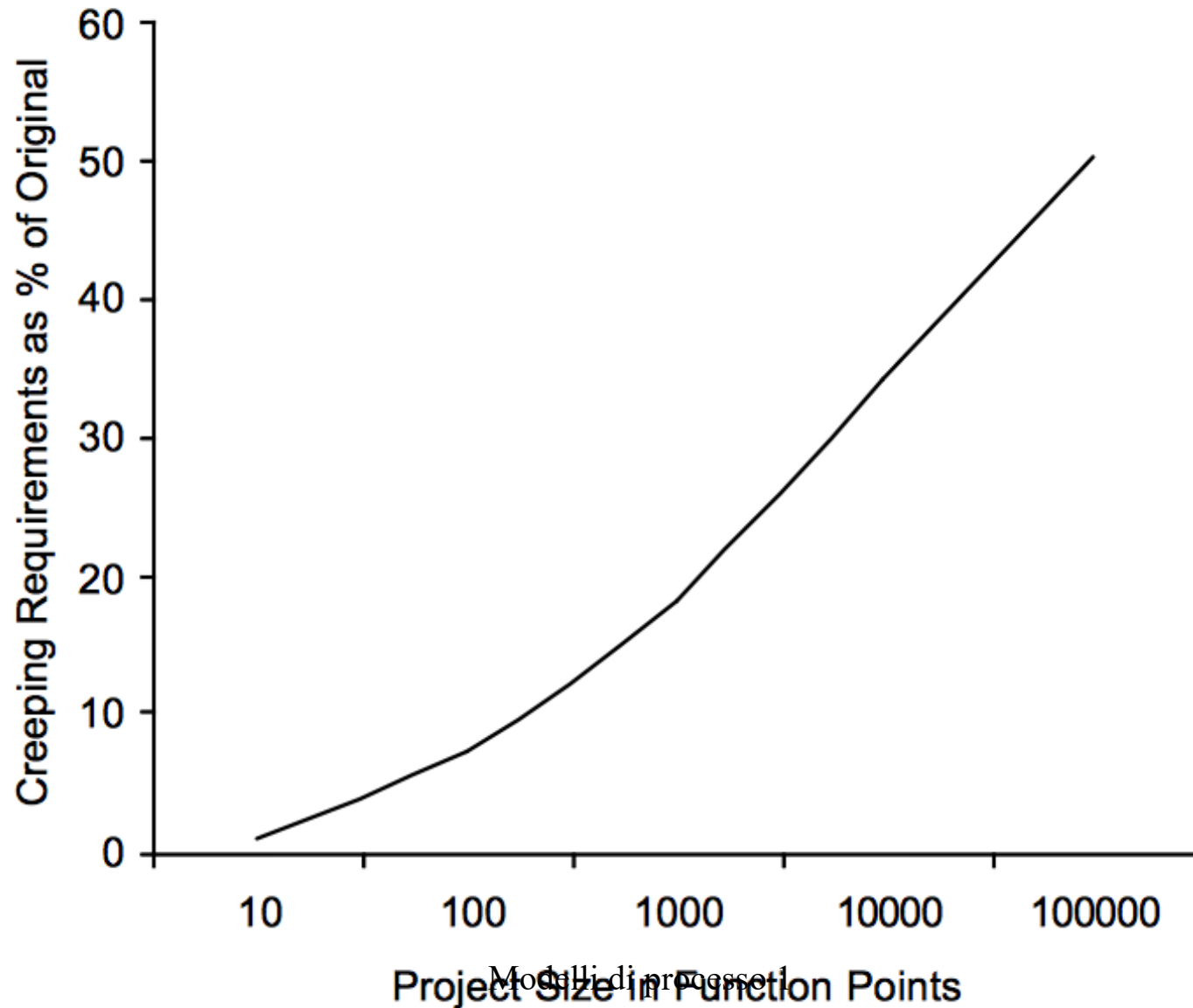
Modello di sviluppo rigidamente sequenziale



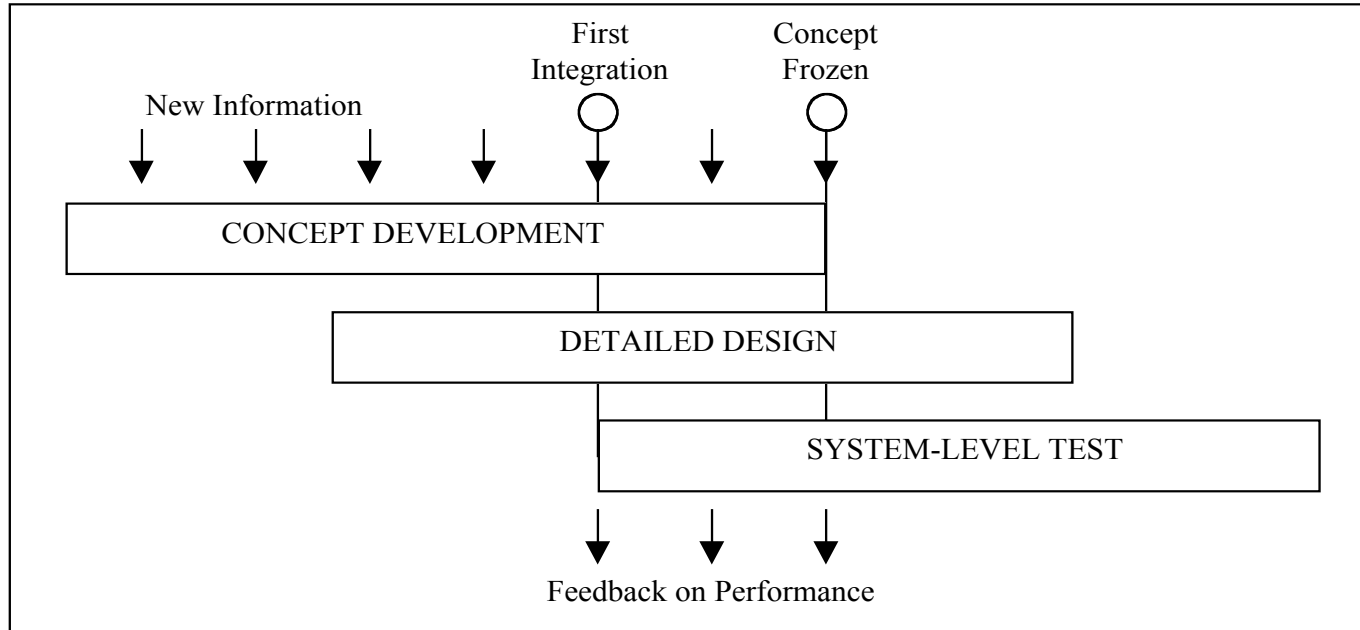
Problema: il processo non risponde ai cambiamenti di mercato che siano più rapidi della sua esecuzione

I requisiti cambiano durante lo sviluppo

(più il sistema è grosso più cambiano!)



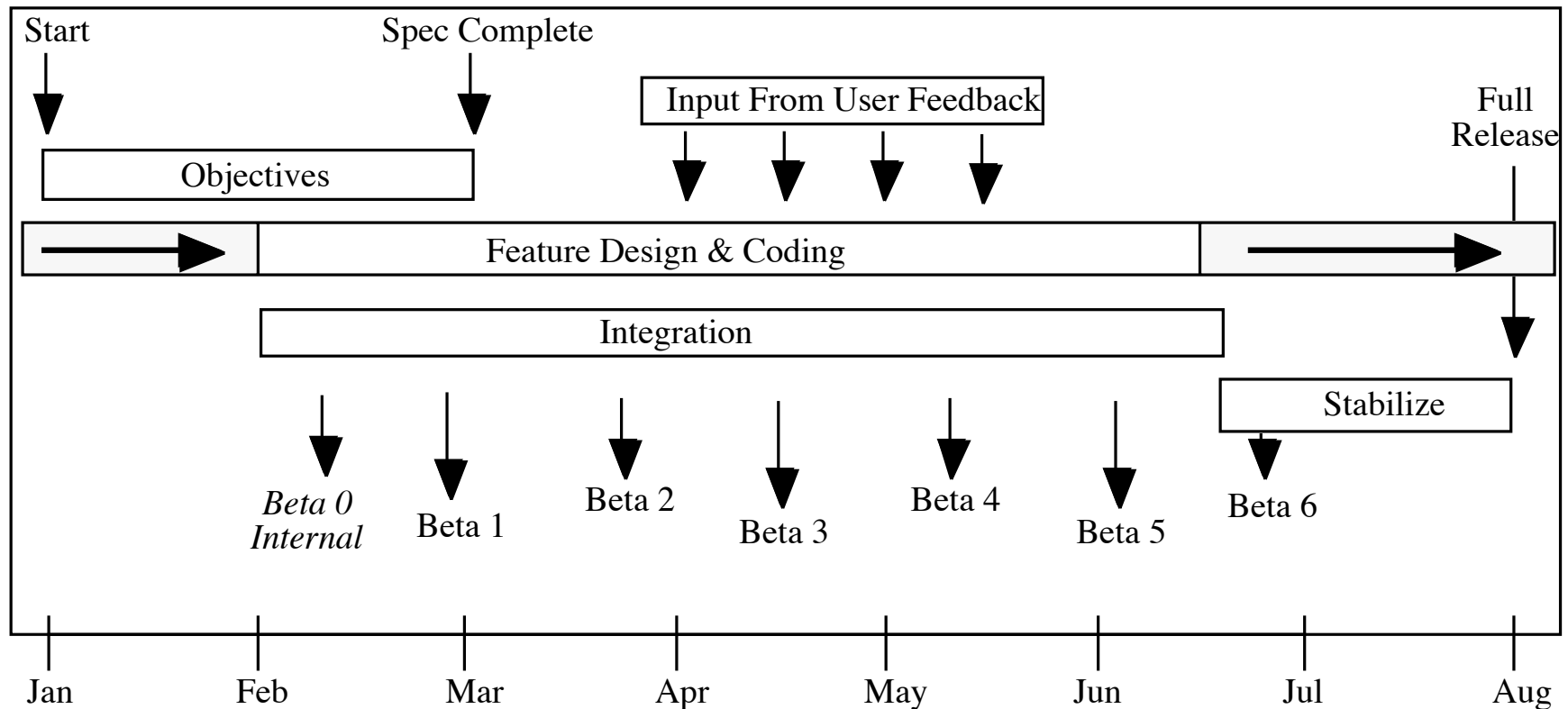
Un modello più flessibile: fasi sovrapposte



- **Caratteristiche**
 - Basato su **apprendimento e adattamento** vs *pianificazione ed esecuzione*
 - Processo iterativo

Esempio

Progetto Netscape's Navigator 3.0

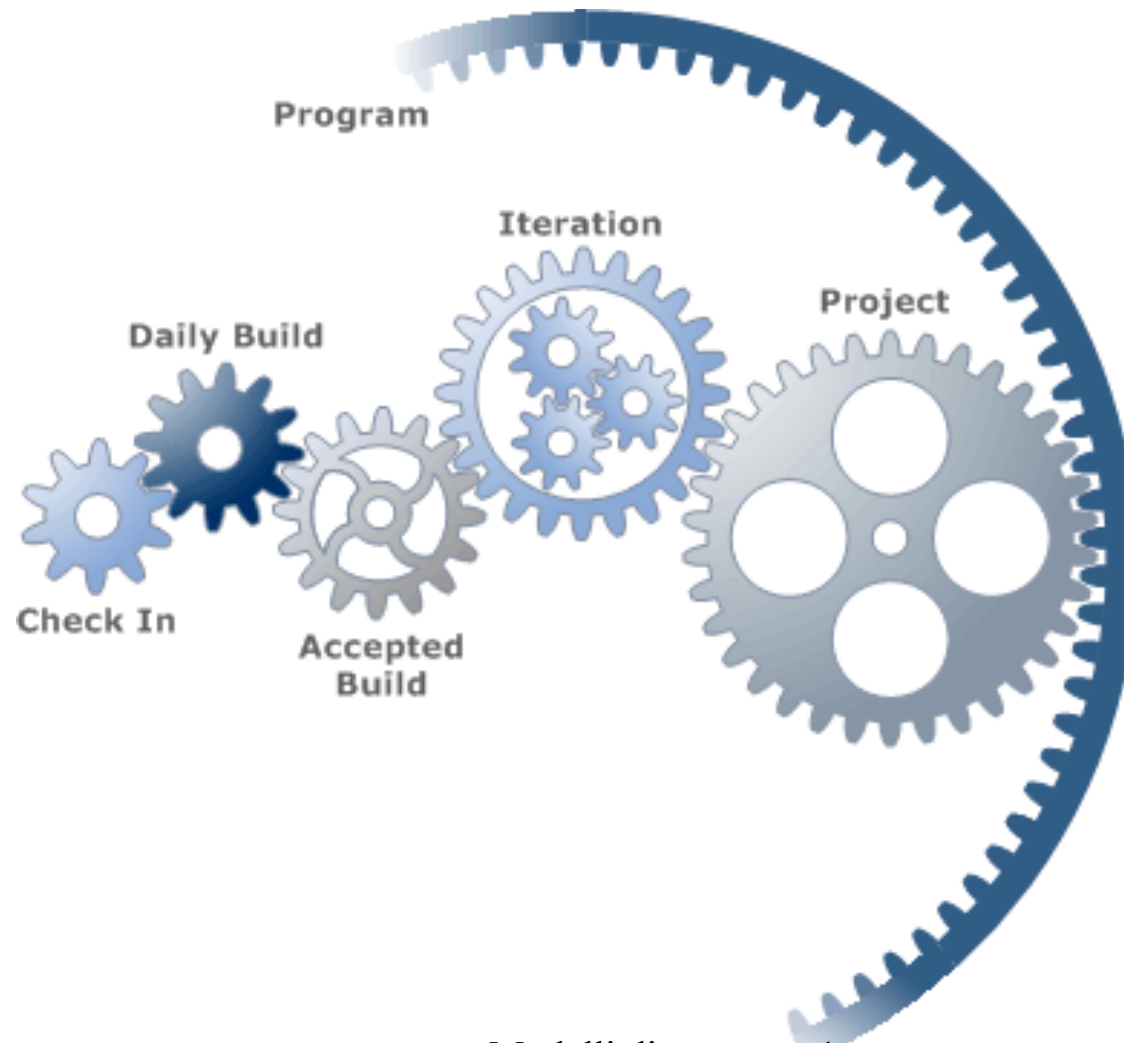


>50% di nuovo codice sviluppato dopo il primo rilascio beta!

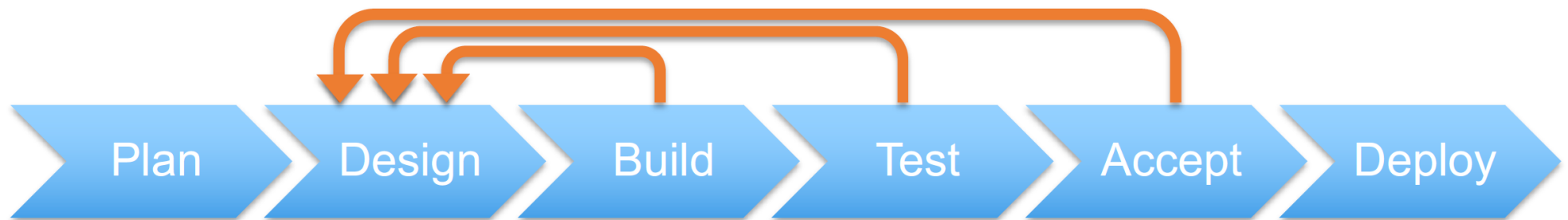
Dimensioni del tempo

- Tempo lineare
 - Collegato ad una nozione intuitiva di progresso
 - “Oggi più di ieri e meno di domani”
- Tempo ciclico
 - Ore, giorni, settimane, mesi, stagioni...
 - Il tempo ciclico in apparenza si ripete senza progredire, ha bisogno di indicazioni di progresso
 - *“Sentinella, a che punto è la notte?” La sentinella risponde: «Viene la mattina, e viene anche la notte. Se volete interrogare, interrogate pure; tornate un'altra volta» Isaia, 21*

Il software si costruisce in cicli (e cicli dentro cicli)

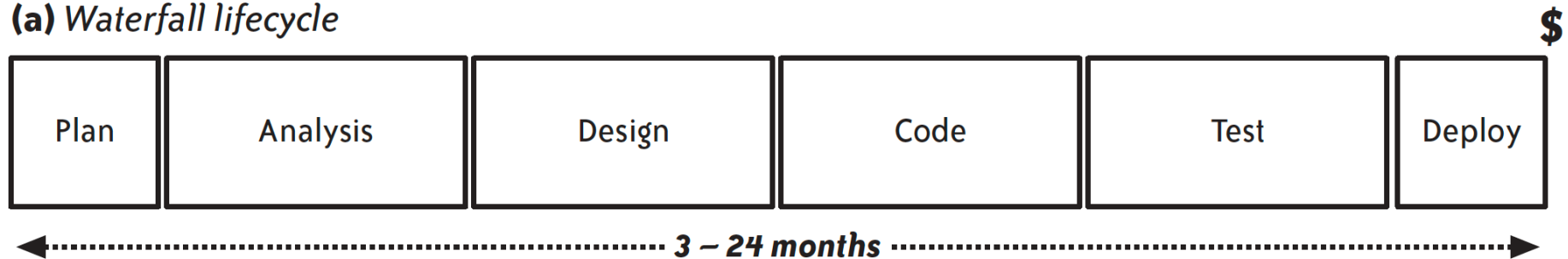


Cicli dentro cicli

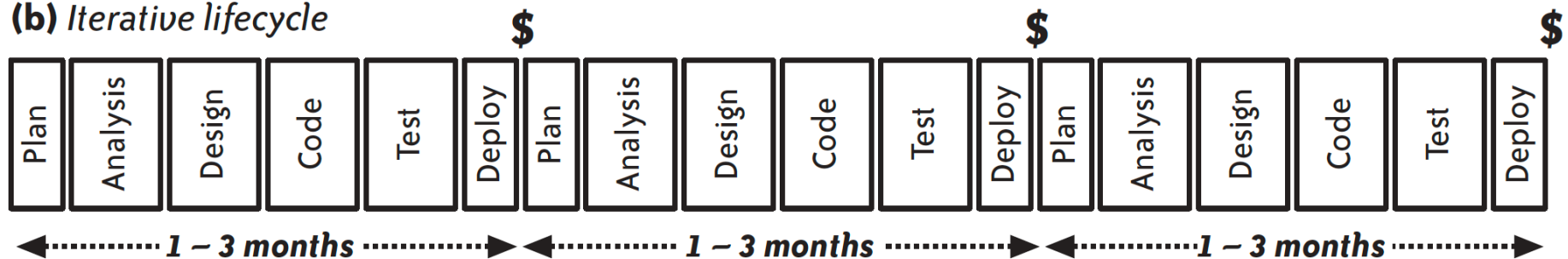


Waterfall vs iterativo

(a) *Waterfall lifecycle*

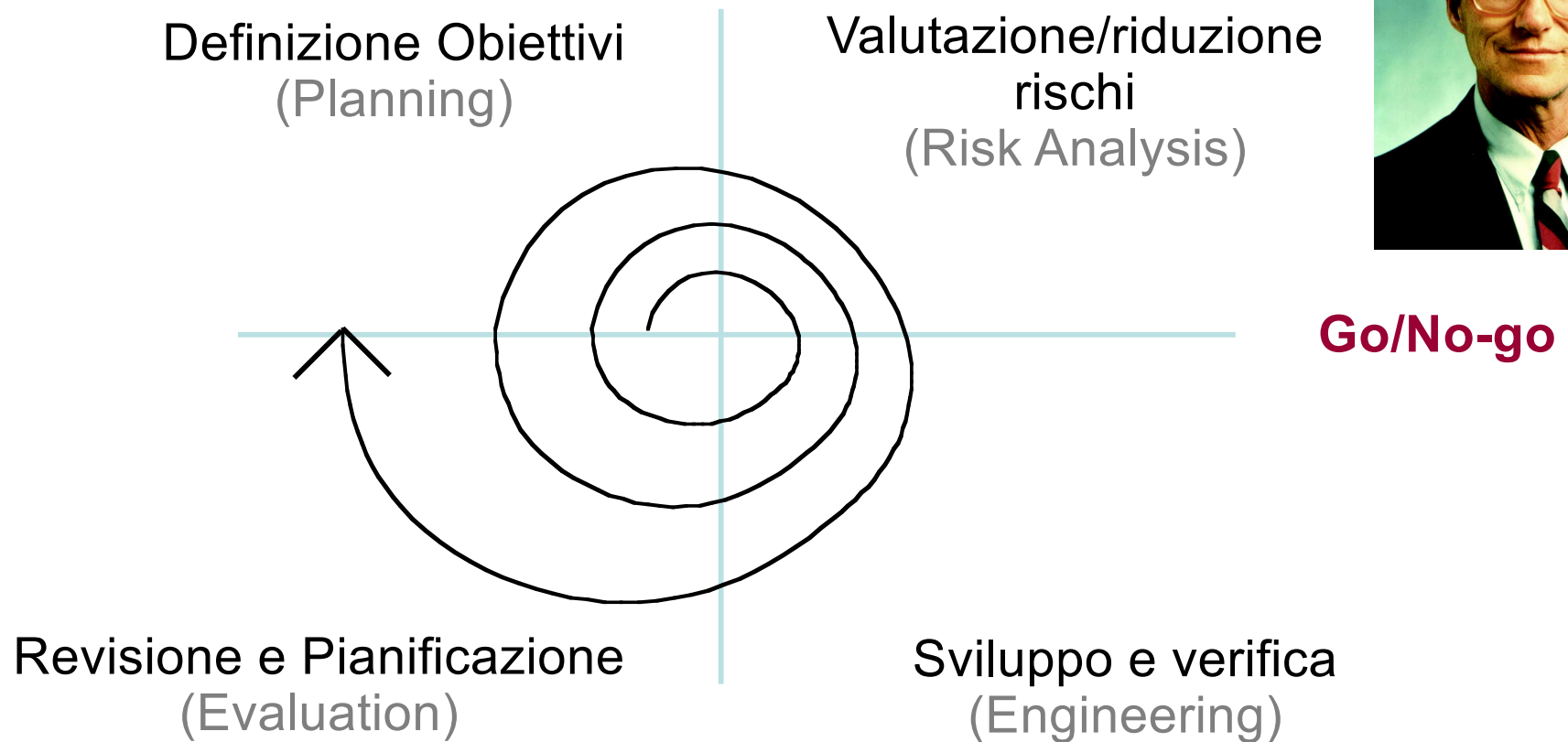
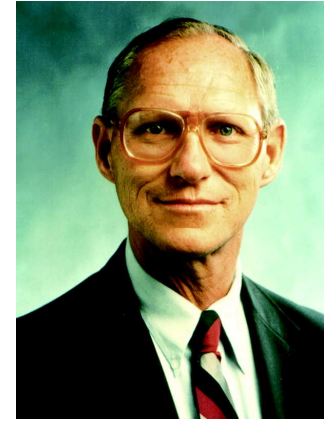


(b) *Iterative lifecycle*



\$ = Potential release

Modello a spirale (Boehm)



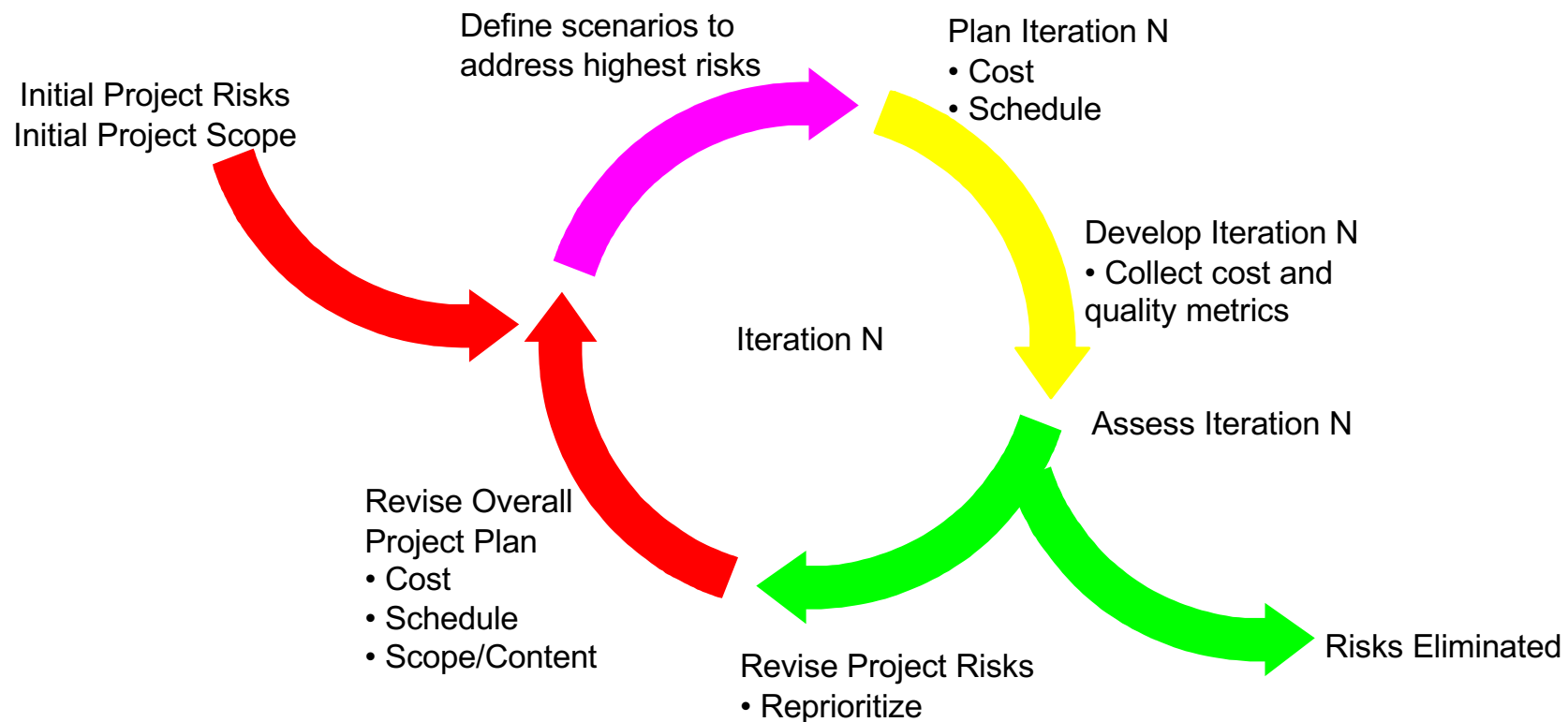
I settori del modello a spirale

- I. Definizione dell'obiettivo
 - Ogni round identifica i propri obiettivi
- II. Valutazione e riduzione dei rischi
 - Messa in priorità dei rischi
 - Ogni rischio deve essere affrontato
- III. Sviluppo e validazione
 - Il modello di sviluppo può essere generico
 - Ogni round include sviluppo e validazione
- IV. Pianificazione
 - Revisione del progetto e pianificazione del suo futuro

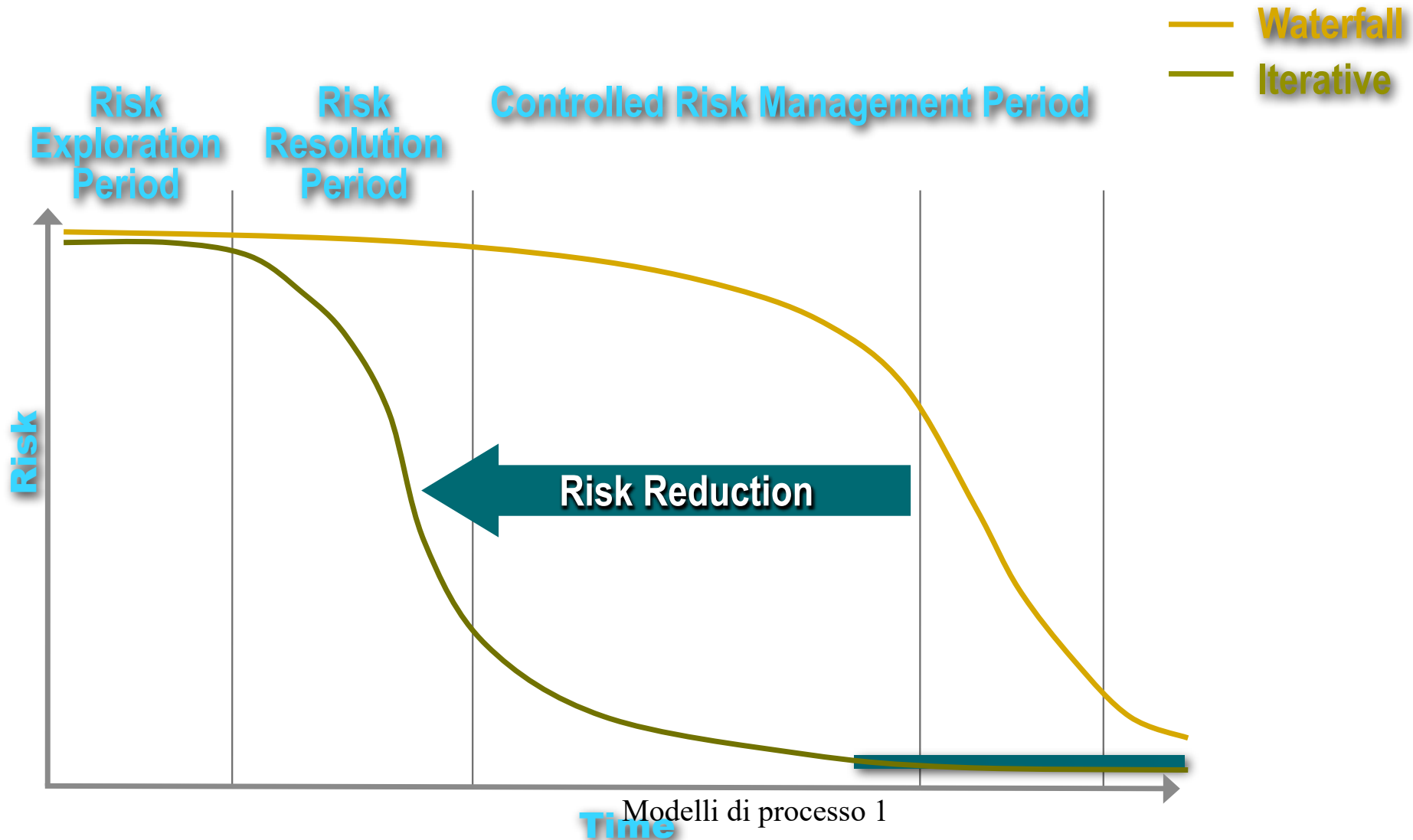
Modello a spirale

- Adatto se requisiti instabili
- Non lineare ma **pianificato**
- Flessibile, si adatta alle esigenze utente
- Valuta il rischio per ogni iterazione
- Può supportare diversi modelli di processo
- Richiede il **coinvolgimento** del cliente
- **Difficile valutare i rischi**
- **Costoso: ROI (Return On Investment)?**

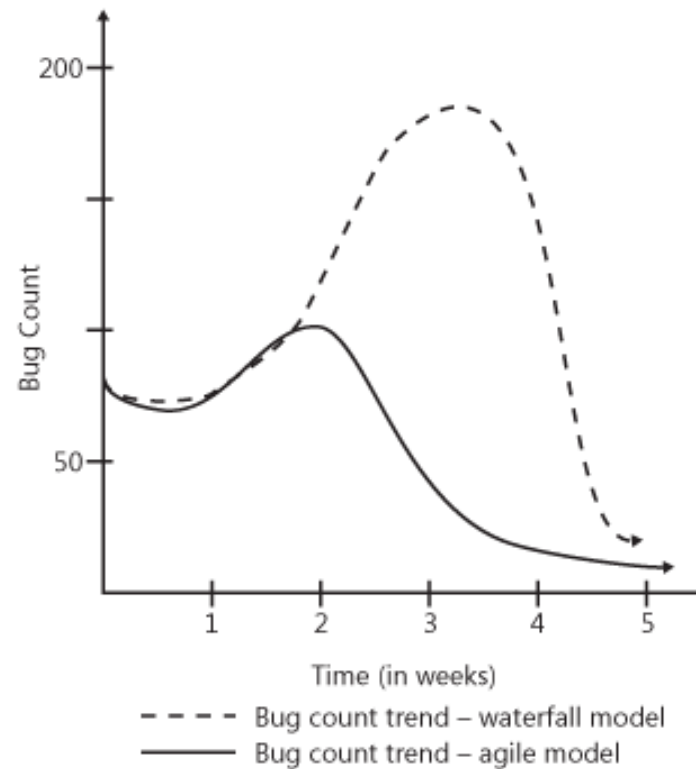
I processi iterativi diminuiscono i rischi



Cascata vs Iterativi

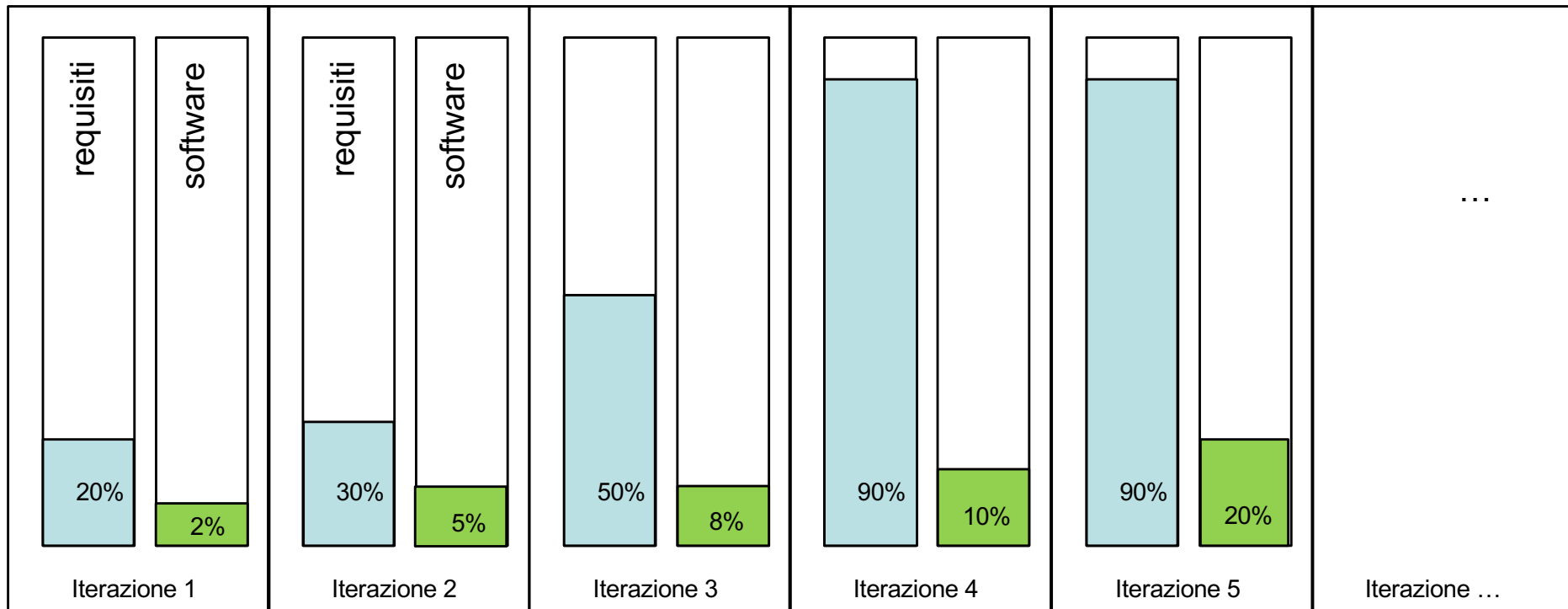


La diminuzione del rischio



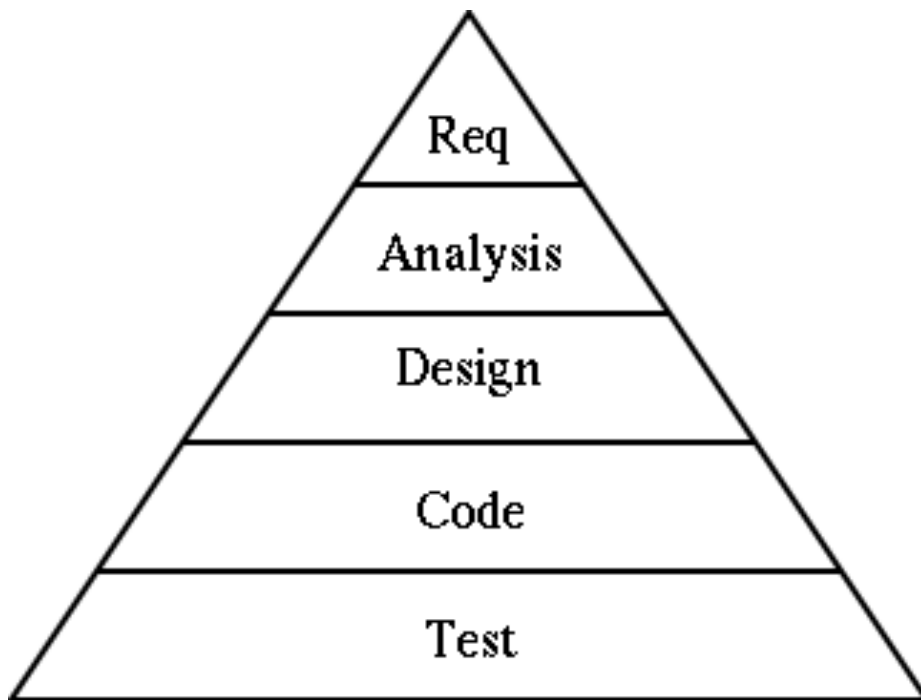
Windows Live Hotmail bug trend comparison, da Marshall, Solid Code, O'Reilly 2009

Iterazioni

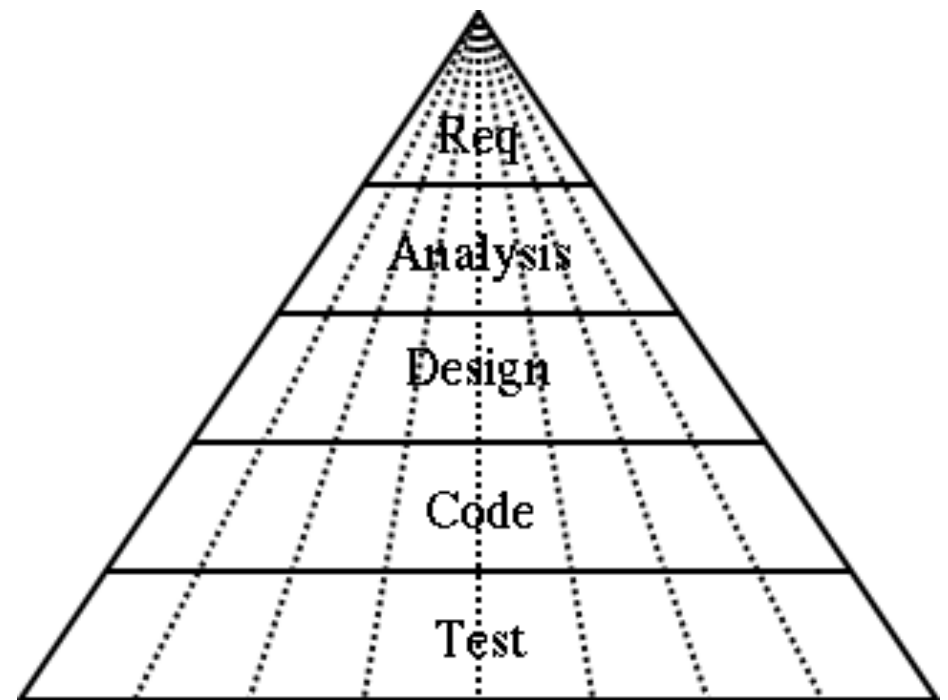


- Questa figura rappresenta un processo iterativo da oltre 5 iterazioni
- Alla fine della quarta iterazione il 90% dei requisiti è stabile ma solo il 10% del software è stato costruito

Cascata vs iterativi: sforzo



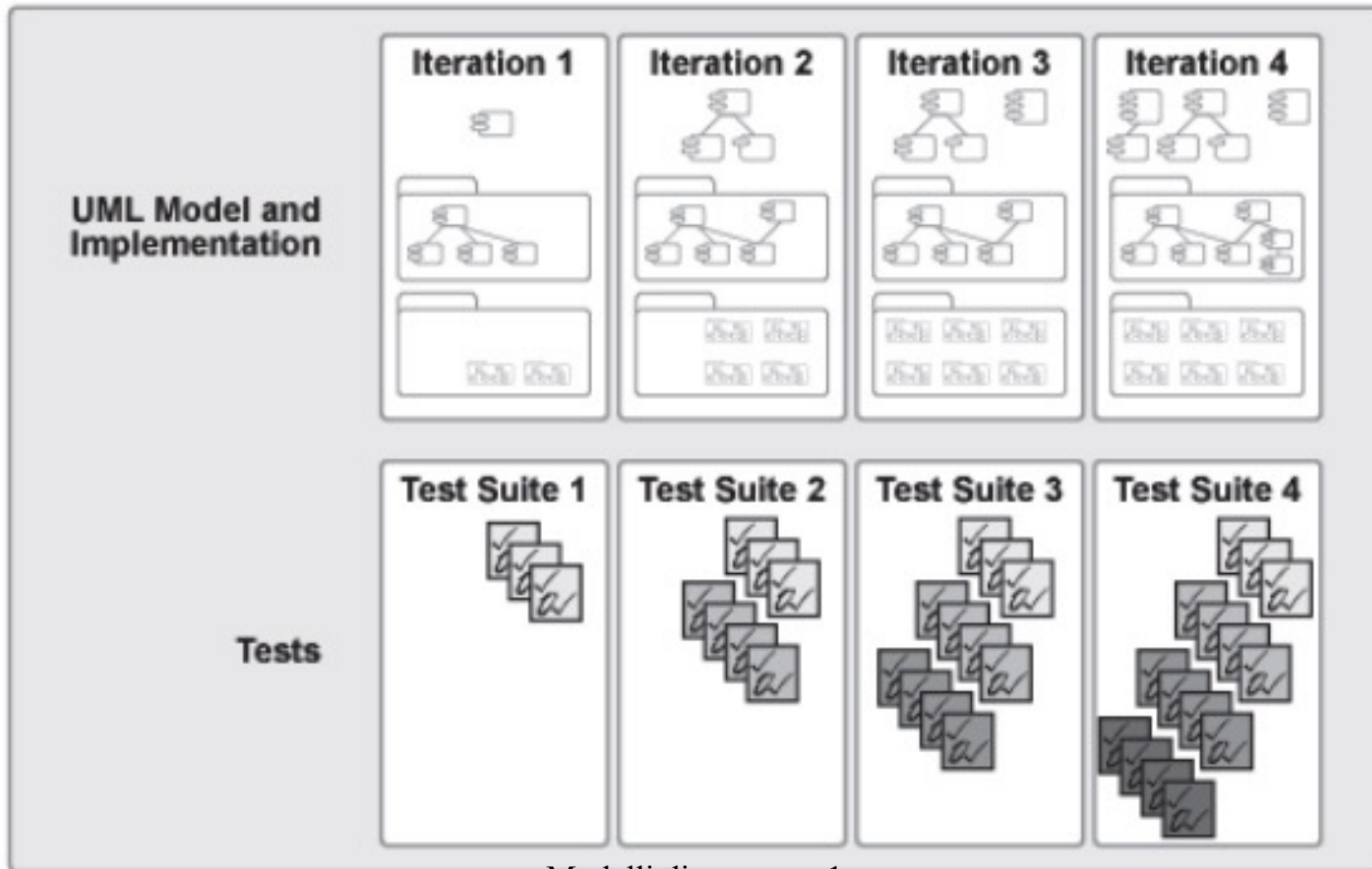
Distribuzione dello sforzo nelle fasi di un processo a cascata: il testing assorbe molto più sforzo delle altre fasi



Segmentazione dello sforzo nelle fasi di un processo iterativo

Modelli di processo 1

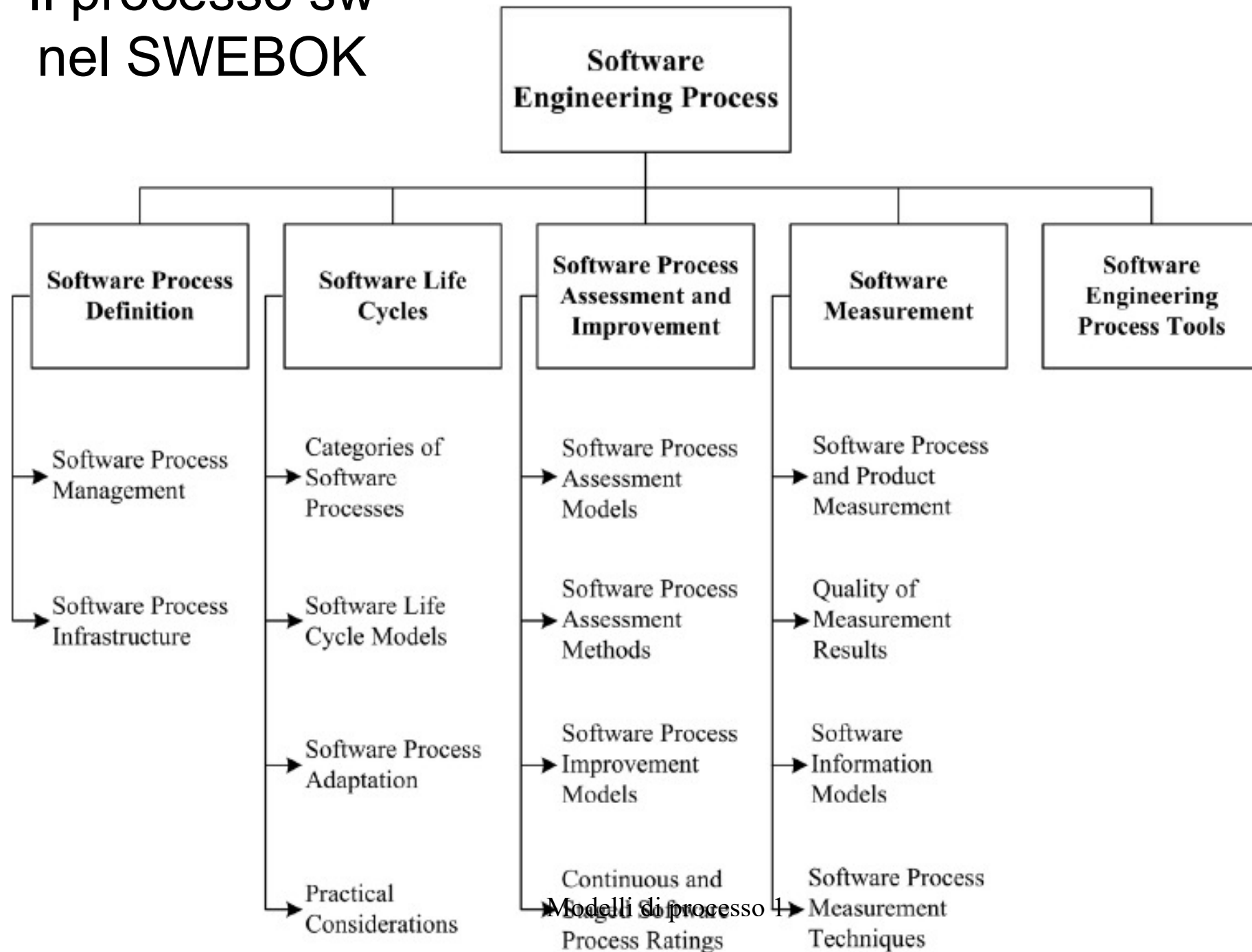
Iterativi: testing incrementale



Processi iterativi

- RUP
- Open UP
- Varianti RUP e MSF
- Synch and stabilize

Il processo sw nel SWEBOK



Qualità dei processi software

1. **Visibilità** – le regole e gli artefatti del processo sono noti a tutti gli stakeholder
2. Misurabilità/accuratezza - il risultato del processo può essere stimato e/o valutato (KPI: Key Performance Indicator). Esempi
 - I. Leadtime: tempo necessario dall'idea al software
 - II. Cycletime: tempo necessario per effettuare un cambiamento al software e renderlo operativo
 - III. Open/close rate: issues – cioè problemi - trovati nel sw e risolti in una unità di tempo
3. **Precisione** – il processo descrive ruoli task e artefatti in modo chiaro e comprensibile a chi deve eseguirlo
4. **Ripetibilità** – il processo può essere ripetuto anche da persone diverse, ottenendo lo stesso risultato

Conclusioni

- Processi waterfall: pianificati, rigidi
- Processi iterativi: pianificati, flessibili
- Processi agili: non pianificati, adattivi
- Esistono molte varianti
- Ogni organizzazione definisce il modello che preferisce, eventualmente adattandolo per classi di prodotti o progetti software

Autotest

- Cos'è un processo software?
- Quali sono le fasi tipiche del processo di sviluppo?
- Quali sono le principali differenze tra processi lineari e processi iterativi?
- Come si riconosce un processo waterfall?
- Quali sono i principali indicatori prestazionali di processo software?

Riferimenti

- Capitolo 8 del SWEBOK: “Software engineering process”
- Boehm: A spiral model of software development and enhancement, *IEEE Computer* 21:5(61-72), May 1988
- Cusumano: How Microsoft builds software, *CACM* 1997
- Kneuper, *Software Processes and Life Cycle Models*, Springer 2018

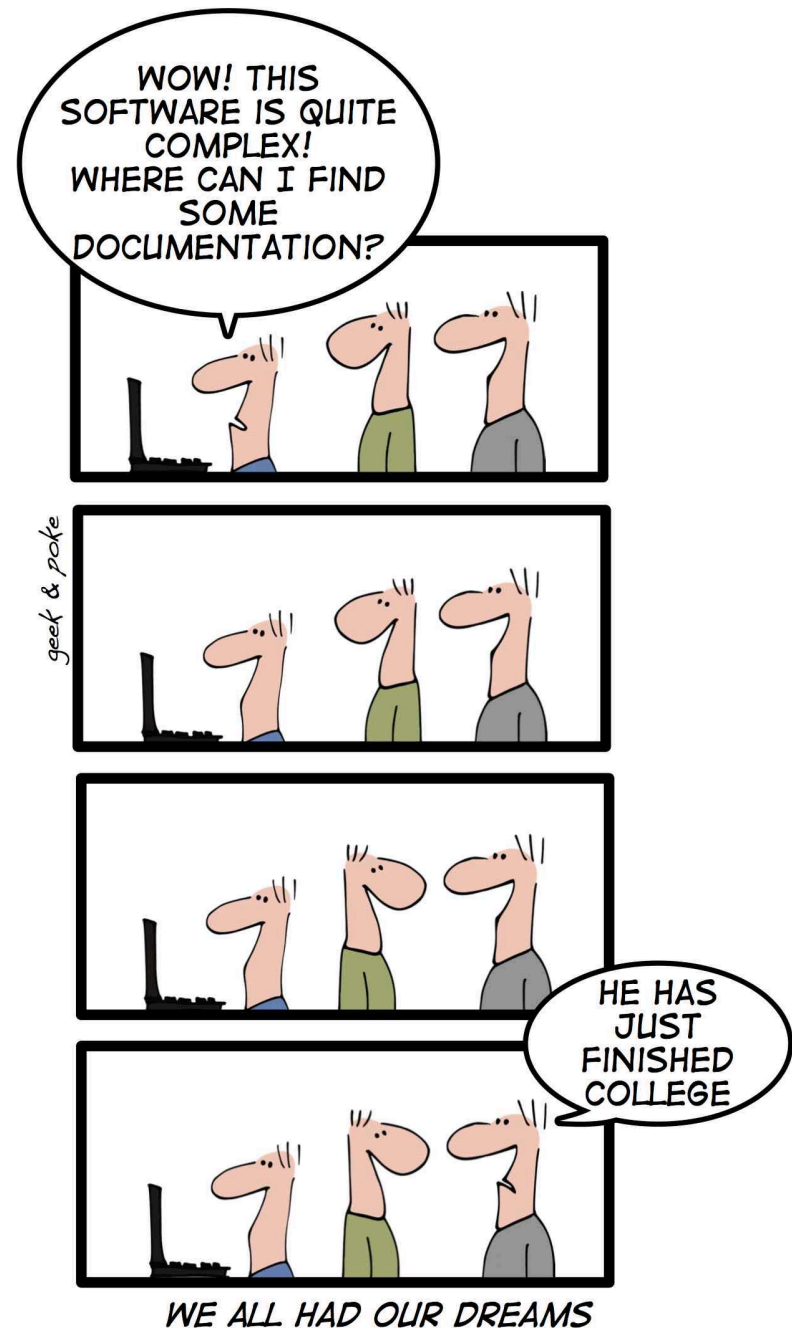
Siti

- **ESSENCE** `semat.org`
- **RUP**
`www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf`
- **Microsoft Solutions Framework**
`www.microsoft.com/technet/itsolutions/msf/default.msp`
- **Personal Software Process**
`resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283`
- **Adaptable Process Model** `www.rspa.com/apm/`

Publicazioni di ricerca

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

Domande?



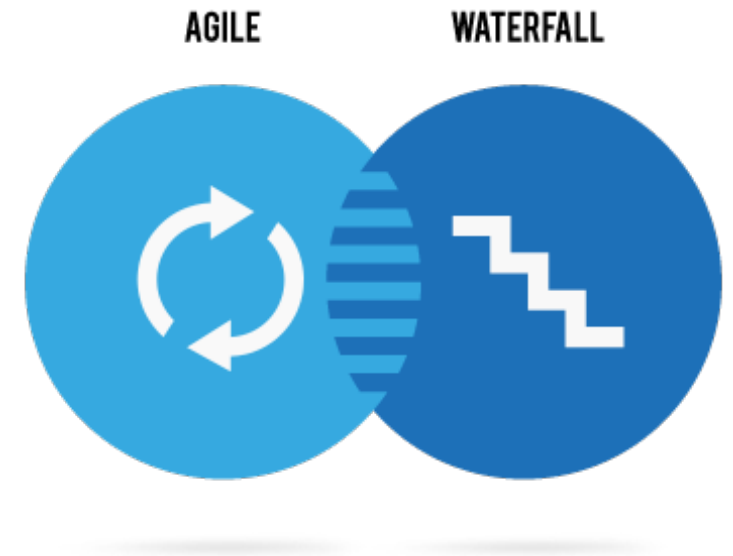
Modelli di processo per lo sviluppo del software: i modelli agili



Corso di Ingegneria del Software
CdL Informatica Università di Bologna

Obiettivi di questa lezione

- I modelli di processo **agili**
- Il manifesto agile
- Extreme Programming (XP)



Sviluppare software è un'attività sociale

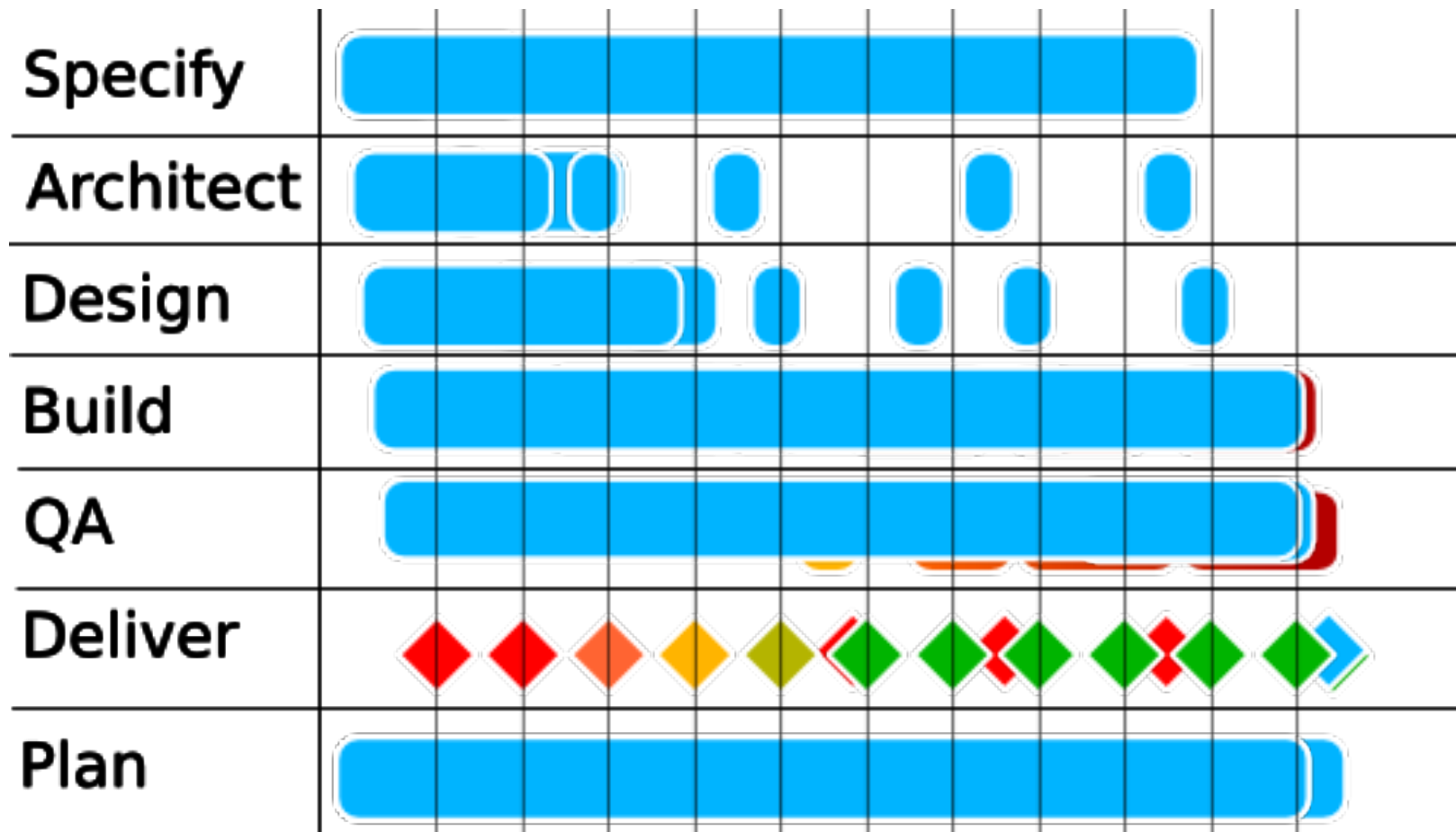
- Quasi tutti i problemi difficili nello sviluppo software riguardano le persone e non le tecnologie.
- Scrivere software non è difficile.
- Tuttavia, scrivere il software *giusto* è molto difficile, perché occorre capire i bisogni degli utenti, negoziare i tempi di consegna, distribuire i compiti, fare compromessi
- Le persone capaci di fare bene il software sono rare: quando hanno successo di solito dimostrano sia abilità tecnica che buone capacità di relazione interpersonale

La pianificazione spesso fallisce

- Alcuni problemi sono complessi, la soluzione richiede il contributo di molte persone
- La soluzione all'inizio non è chiara
- I requisiti della soluzione durante lo sviluppo cambieranno, perché gli utenti non sanno bene cosa vogliono
- Il prodotto può essere consegnato in versioni incrementali successive, ma non sappiamo quanti incrementi ci vorranno
- È richiesta una collaborazione stretta e un feedback rapido e continuo dagli utenti finali

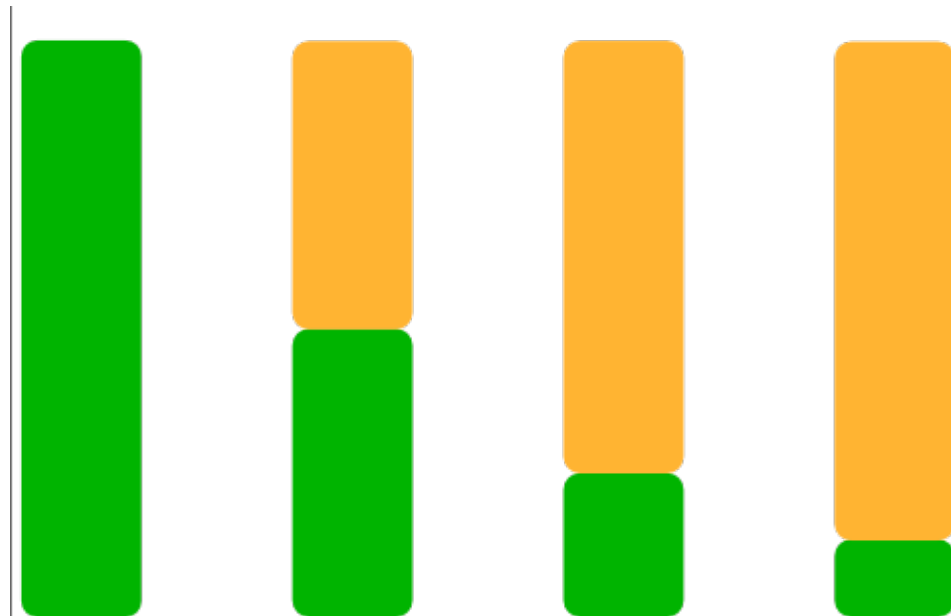
L'interesse dell'utente:

in quale riga lo vedete?



Volatilità dei requisiti

Ogni sei mesi – o anche meno – circa metà dei requisiti di un prodotto software perdono di interesse



Modelli di processo 2

www.infoq.com/articles/the-curse-of-the-change-control-mechanism

....quindi....

- L'utente deve far parte del gioco fin dall'inizio
- Occorre evitare gli sprechi di tempo
- occorre adattarsi al cambiamento delle richieste dell'utente

Migliorare i processi iterativi

- I processi iterativi devono produrre valore per gli stakeholder in modo incrementale e senza sprechi
- La filosofia **agile** ha lo scopo di evitare gli sprechi e le perdite di tempo – le attività inutili

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 valori a destra sono importanti, ma noi preferiamo quelli a sinistra

www.agilemanifesto.org

I principi agili

1. La nostra priorità è soddisfare il cliente mediante consegne anticipate e continue di software di valore
2. Le persone dell'azienda e gli sviluppatori devono quotidianamente lavorare assieme durante tutto il progetto
3. Le modifiche ai requisiti sono benvenute, anche nelle ultime fasi dello sviluppo
4. Consegnare di frequente software funzionante
5. Il software funzionante è la prima misura di progresso
6. I progetti si costruiscono attorno a individui motivati. Dategli l'ambiente ed il supporto di cui hanno bisogno, e confidate che facciano il loro lavoro
7. Le migliori architetture, requisiti, e design emergono da team auto-organizzanti
8. Il metodo più efficace ed efficiente di trasmettere informazioni verso e all'interno di uno sviluppo è mediante conversazioni faccia a faccia
9. I processi agili promuovono lo sviluppo sostenibile
10. L'attenzione costante all'eccellenza tecnica e al buon design esaltano l'agilità
11. La semplicità è essenziale
12. I team di sviluppo valutano la propria efficacia ad intervalli regolari e modificano di conseguenza il proprio comportamento

I principi agili

1. La nostra priorità è **soddisfare il cliente** mediante consegne anticipate e continue di software di valore
2. Le persone dell'azienda e gli sviluppatori devono **quotidianamente lavorare assieme** durante tutto il progetto
3. Le **modifiche ai requisiti sono benvenute**, anche nelle ultime fasi dello sviluppo
4. Consegnare di frequente software **funzionante**
5. Il software funzionante è la **prima misura di progresso**
6. I progetti si costruiscono attorno a **individui motivati**. Dategli l'ambiente ed il supporto di cui hanno bisogno, e confidate che facciano il loro lavoro
7. Le migliori architetture, requisiti, e design emergono da **team auto-organizzanti**
8. Il metodo più efficace ed efficiente di trasmettere informazioni verso e all'interno di uno sviluppo è mediante **conversazioni faccia a faccia**
9. I processi agili promuovono lo **sviluppo sostenibile**
10. L'attenzione costante all'**eccellenza tecnica** e al buon design esaltano l'agilità
11. La **semplicità** è essenziale
12. I team di sviluppo **valutano la propria efficacia** ad intervalli regolari e modificano di conseguenza il proprio comportamento

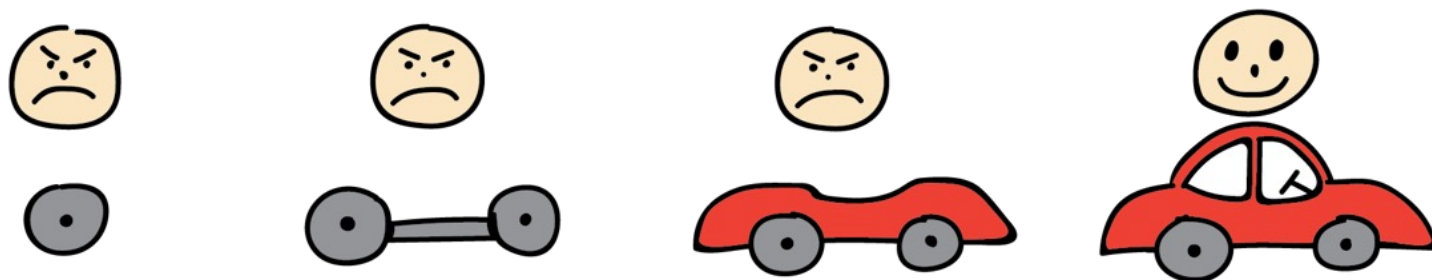
Metodi agili

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

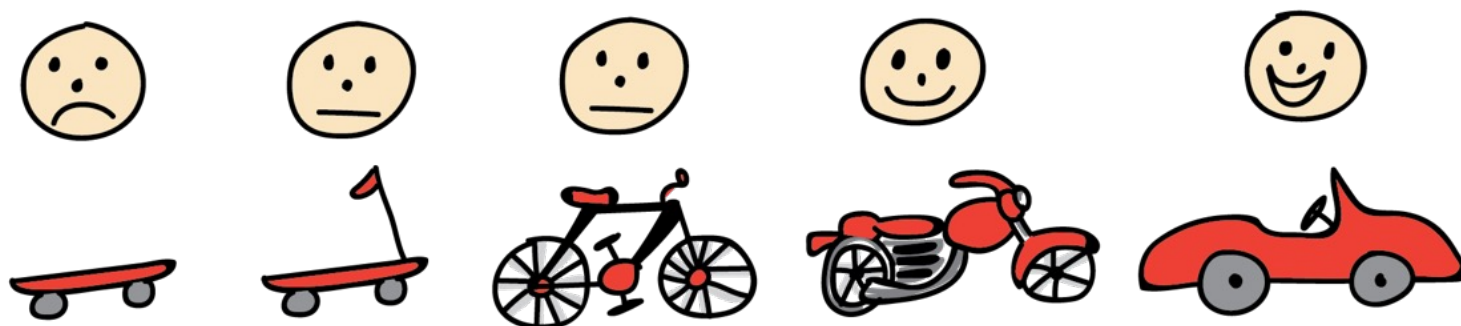
- **Rilasci frequenti** del prodotto sviluppato in modo incrementale
- **Collaborazione** continua del team di sviluppo col cliente
- **Documentazione** di sviluppo **ridotta**
- **Valutazione** sistematica e continua di **valori e rischi** dei **cambiamenti**

Minimal Viable Product (prodotto minimo funzionante)

Not Like
This!



Like This!



MVP: inizi di dropbox

<https://www.youtube.com/watch?v=xy9nSnalvPc>

https://www.youtube.com/watch?v=qxFLfY7_Gqw



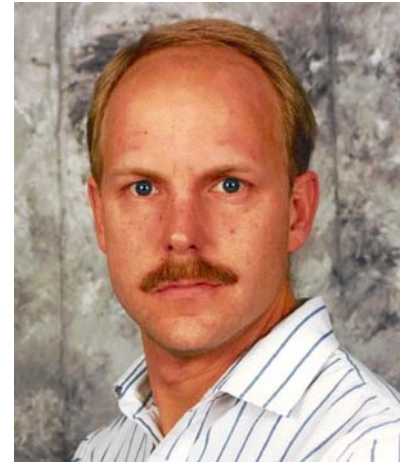
Metodi agili... ne esistono tanti!

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

eXtreme Programming (XP)

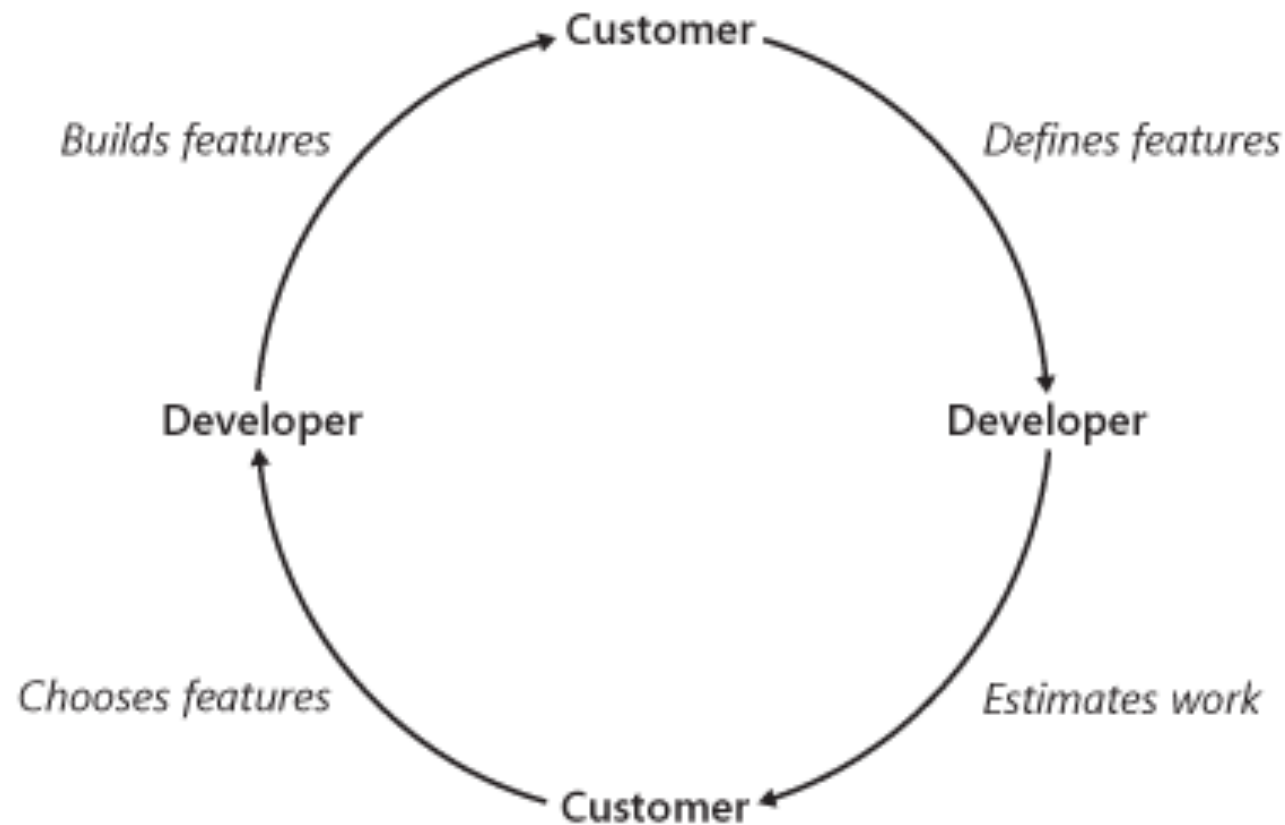
“Extreme Programming è una disciplina di sviluppo software basata sui valori di

- *semplicità*,
- *comunicazione*,
- *feedback*, e
- *coraggio*”.



Kent Beck

La base della collaborazione agile in XP



Il team XP

- Il team di sviluppo, di solito costituito da meno di dieci persone, è riunito nello stesso locale
- E' sempre presente un rappresentante del cliente, capace di rispondere a domande del team riguardo ai requisiti
- 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

Le pratiche 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:


- Brevi frasi, usate al posto di documenti dettagliati di specifica dei requisiti
- Scritte assieme ai clienti: cosa si aspettano dal sistema
- una storia è descritta da una o due frasi in testo naturale, con la terminologia del cliente
- utili al team per stimare i tempi/costi del rilascio della nuova versione (incremento)

Esempi di user stories

- Uno studente può acquistare online un abbonamento di parcheggio
- Gli abbonamenti di parcheggio si possono pagare con carta di credito
- Gli abbonamenti di parcheggio si possono pagare con Paypal
- Un professore può inserire i voti online
- Uno studente può visualizzare l'orario delle lezioni
- Uno studente può ottenere una registrazione della lezione
- Uno studente può iscriversi ad un corso se soddisfa i prerequisiti
- La registrazione di un corso si può visualizzare mediante un browser

Esempio di scheda per user story

173. Students can purchase parking passes.

Priority:  8

Estimate: 4

Story points

Formato **preferito** per le US

Come studente

voglio poter comprare on
line un abbonamento
mensile al parcheggio

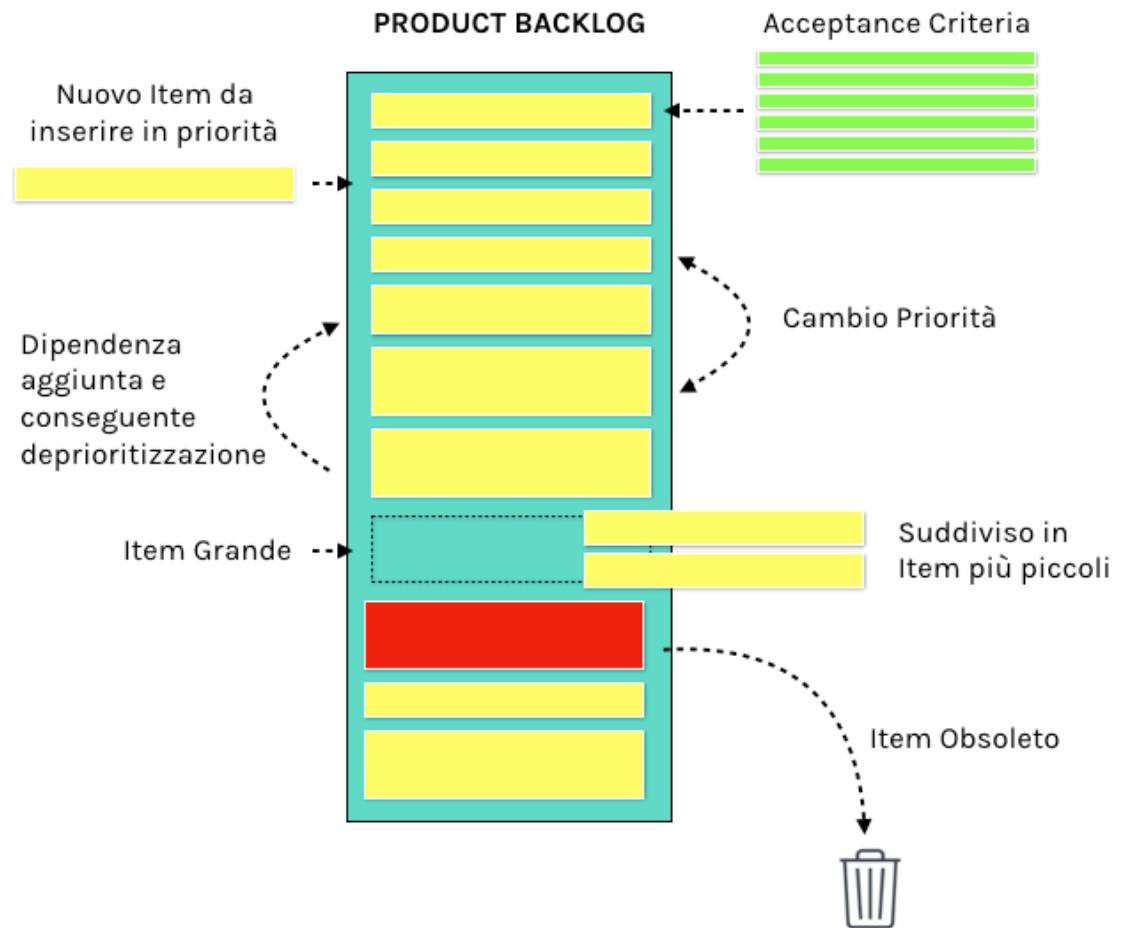
per non tirare fuori soldi
tutti i giorni

→ Tipo di utente

→ Obiettivo

→ Motivazione

Backlog di prodotto: insieme di US



<https://www.productheroes.it/product-backlog-agile-scrum/>

Storie e iterazioni



Il “Planning game”

- Le “**User Stories**” permettono di fare una pianificazione a breve termine
 - 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**
 - **Le risorse** necessarie (**story points**) e i rischi 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 per ciascuna iterazione

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: risponde direttamente il cliente
- Gli sviluppatori non devono attendere le decisioni del cliente
- La comunicazione “faccia a faccia” minimizza la possibilità di ambiguità ed equivoci

Il metodo MOSCOW per mettere in priorità le US

MOSCOW : acronimo per Must/Should/Could/Won't
(DEVE / DOVREBBE / POTREBBE / NO)

- Must: funzioni che DEBBONO esserci nel prodotto
- Should: funzioni che DOVREBBERO esserci
- Could: funzioni che POTREBBERO esserci
- Wont: funzioni che NON INSERIREMO nella versione attuale

Esempio

- L'utente potrà loggarsi con username e passwd
- L'utente potrà registrare un nuovo account
- L'utente potrà vedere tutti i documenti Word inclusi nel file system
- L'utente potrà cancellare tutti i documenti Word
- L'utente potrà accedere ogni documento entro il file system
- L'utente potrà loggarsi con doppia autenticazione

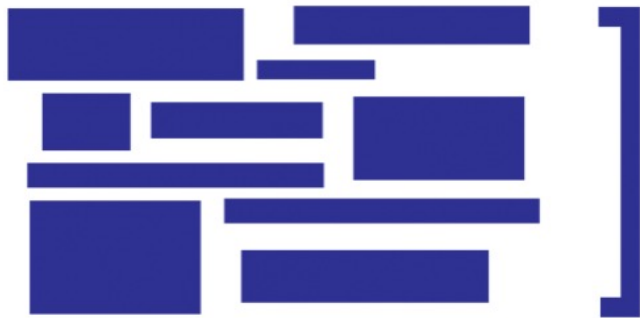
Esempio

- L'utente DEVE loggarsi con username e passwd
- L'utente DOVREBBE registrare un nuovo account
- L'utente DOVREBBE vedere tutti i documenti Word inclusi nel file system
- L'utente POTREBBE cancellare tutti i documenti Word
- L'utente DEVE accedere ogni documento entro il file system
- L'utente NON POTRÀ loggarsi con doppia autenticazione

In scope for this timeframe

(Project / Increment / Timebox)

Must Have



Typically
no more
than
60% effort

Should Have



Could Have



Typically
around
20% effort

Out of scope for this timeframe

Won't Have this time



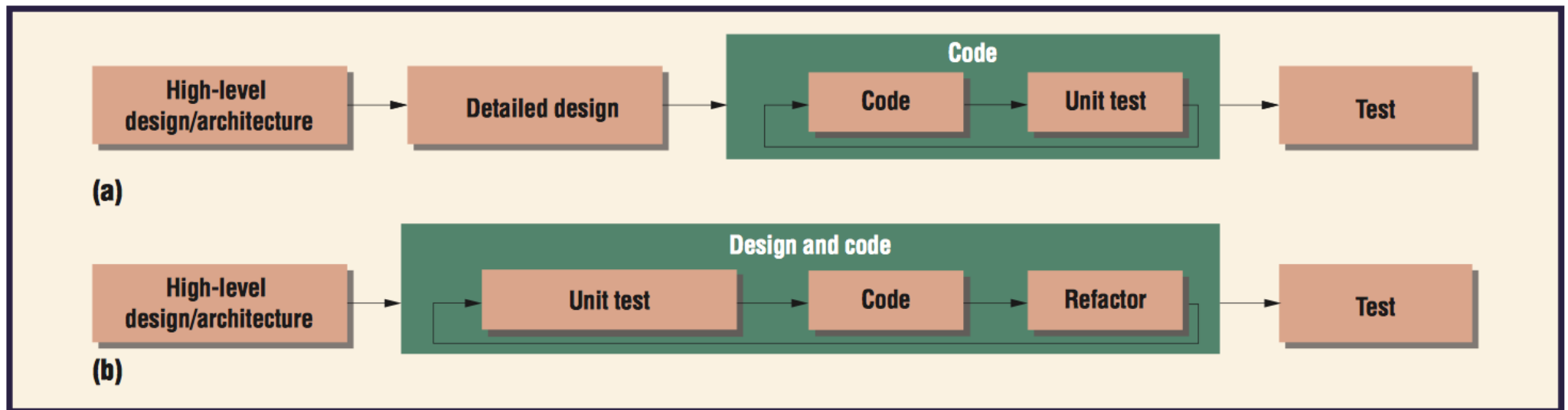
Sviluppo test-driven

Test-Driven Development (TDD)

- Scegliere una user story e definire i test prima del codice
 - TDD è una tecnica di programmazione
 - Automatizzare i test (es. usare xUnit)
 - Deve girare tutto prima di proseguire con altro codice
- *Unit test vs acceptance test*



Test classico vs test driven



a) Testing classico b) test-driven design come tecnica di programmazione

<https://semaphoreci.com/blog/test-driven-development>

Test di accettazione

(customer tests)

Test di accettazione

- Guidati dalle user stories
- Scritti col cliente
- Funzionano come “contratto”
- Misura del progresso



Esempio

Support technician sees customer's history on screen at the start of a call

Esempio di user story su scheda

- Simulate a call with Fred's account number and verify that Fred's info can be read from the screen*
- Verify that the system displays a valid error message for a non-existing account number*
- Omit the account number in the incoming call completely and verify that the system displays the text "no account number provided" on the screen*

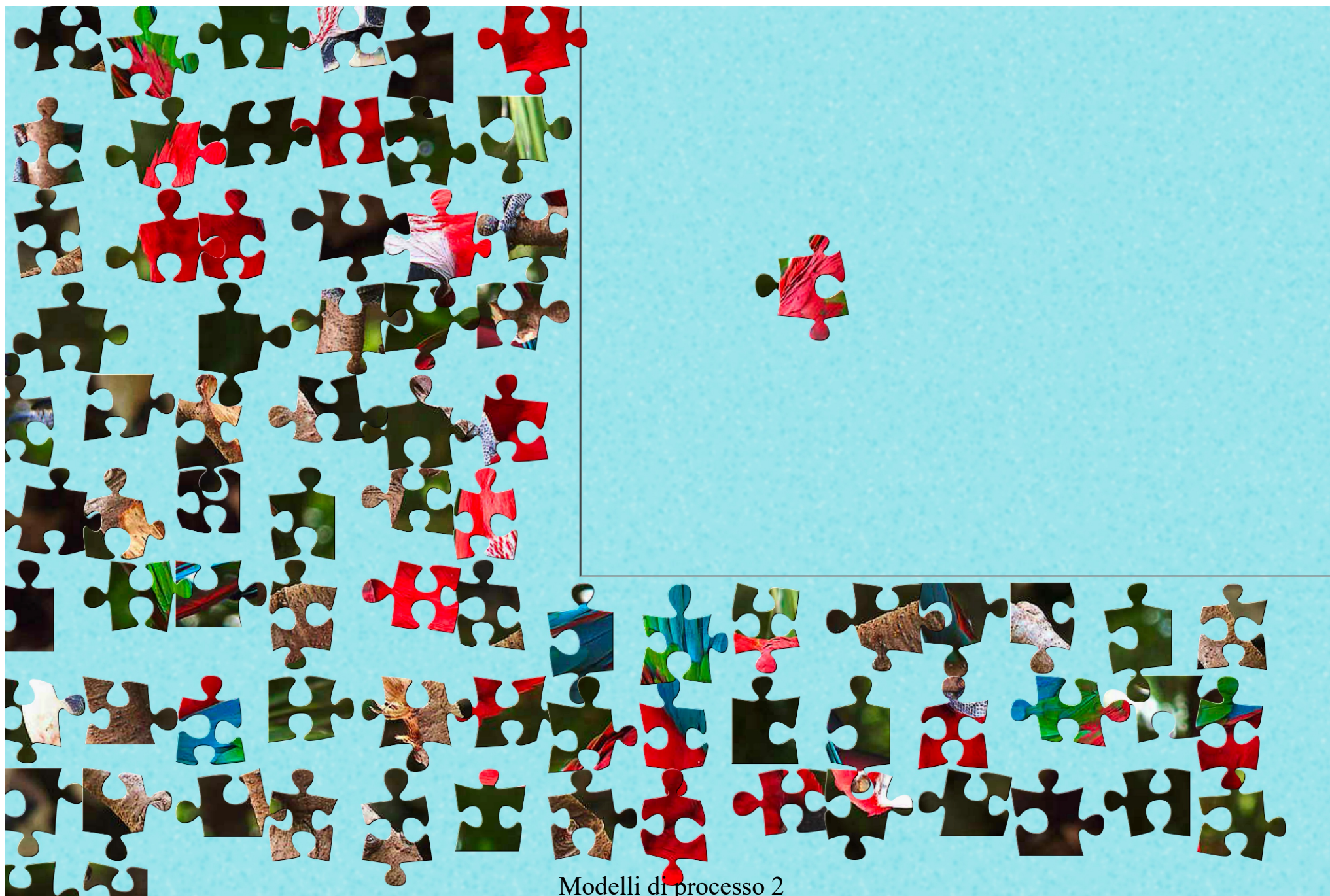
Esempio di test scritto sul retro della user story

Piccoli rilasci

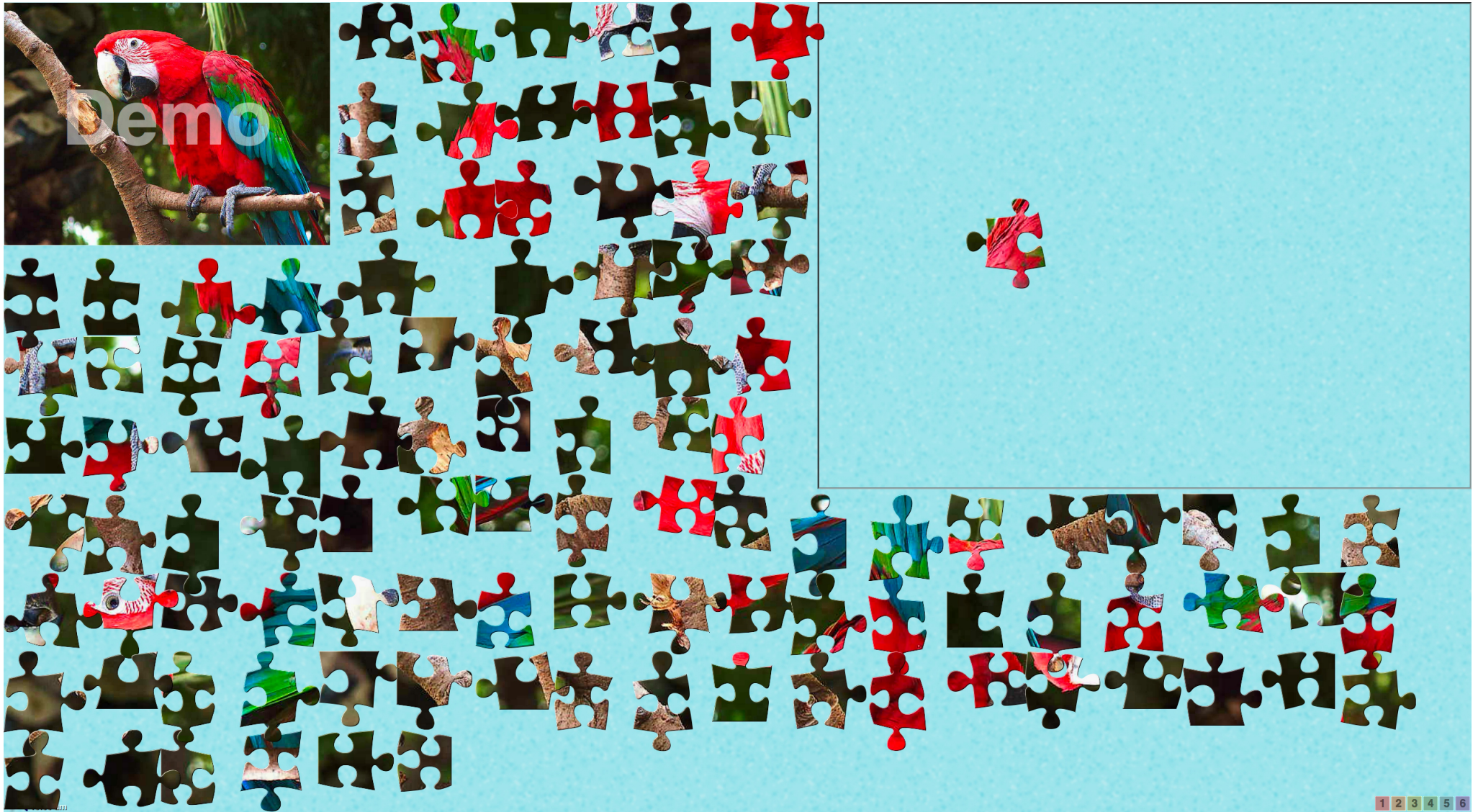
- Timeboxed (ovvero di durata “breve” prefissata)
- Minimali, ma comunque utili (**microincrementi**)
 - Mai cose come ‘implementare il database’
- Ottenere feedback dal cliente presto e spesso
- Eseguire il “planning game” dopo ciascuna iterazione
 - Si voleva qualcosa di diverso?
 - Le priorità sono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



Modelli di processo 2

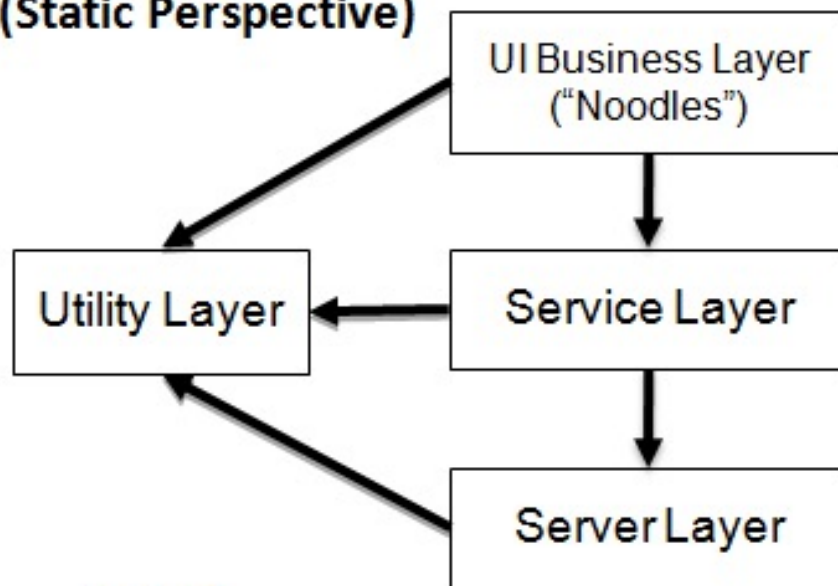


Modelli di processo 2

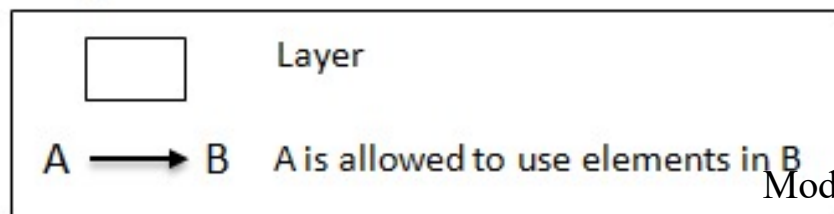
Esempio di metafora

Fonte: neverletdown.net/topics/architecture/

Web Client Organization View (Static Perspective)



Legend



The "Bento Box"

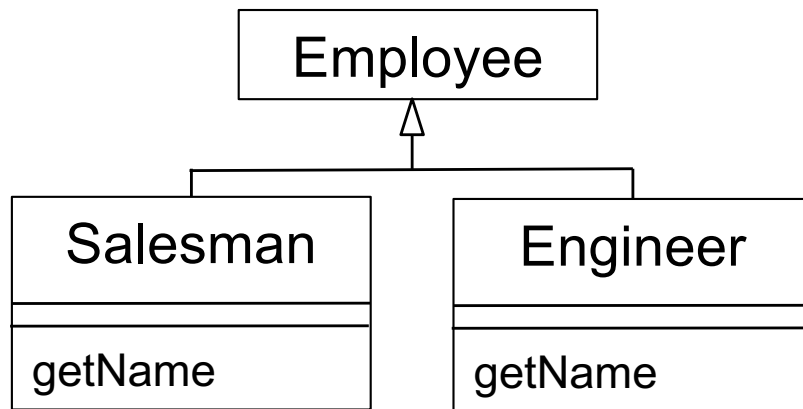
Progettare in modo semplice

- 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)

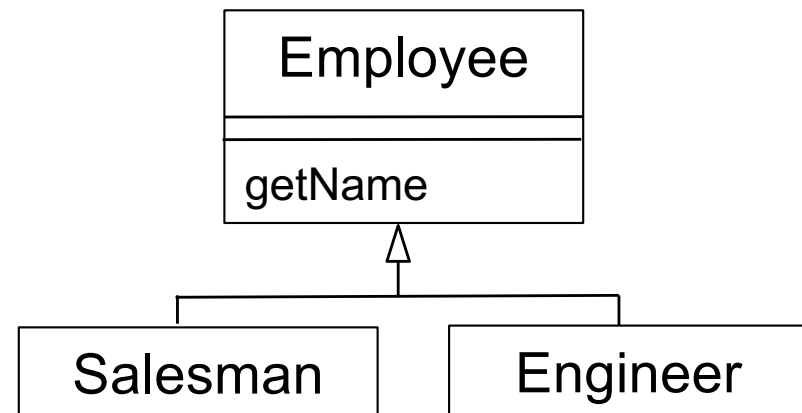
Rifattorizzare (refactoring)

- **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 il codice
- L'altro, **il navigatore**, osserva il codice cercando difetti e partecipa al brainstorming su richiesta
- I ruoli di driver e navigatore vengono scambiati tra i due progettisti periodicamente (es. a metà giornata)
- Le coppie cambiano ogni giorno: lo scopo è che tutti prendano confidenza col codice (vedi: proprietà collettiva del codice)

Pair programming migliora la qualità

	Progetto1: solisti	Progetto2: coppie
Dimensione (KLOC)	20	520
Team	4	12
Sforzo (mesi persona)	4	72
Produttività (KLOC/mp)	5	7.2
Produttività (KLOC/mpair)	n.d.	14.4
Difetti test unità	107 (5.34 difetti/KLOC)	183 (0.4 difetti/KLOC)
Difetti test integrazione	46 (2.3 difetti/KLOC)	82 (0.2 difetti/KLOC)

Fonte: Williams, *Pair Programming Illuminated*, AW, 2002

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 *collective 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

Convenzioni di codifica

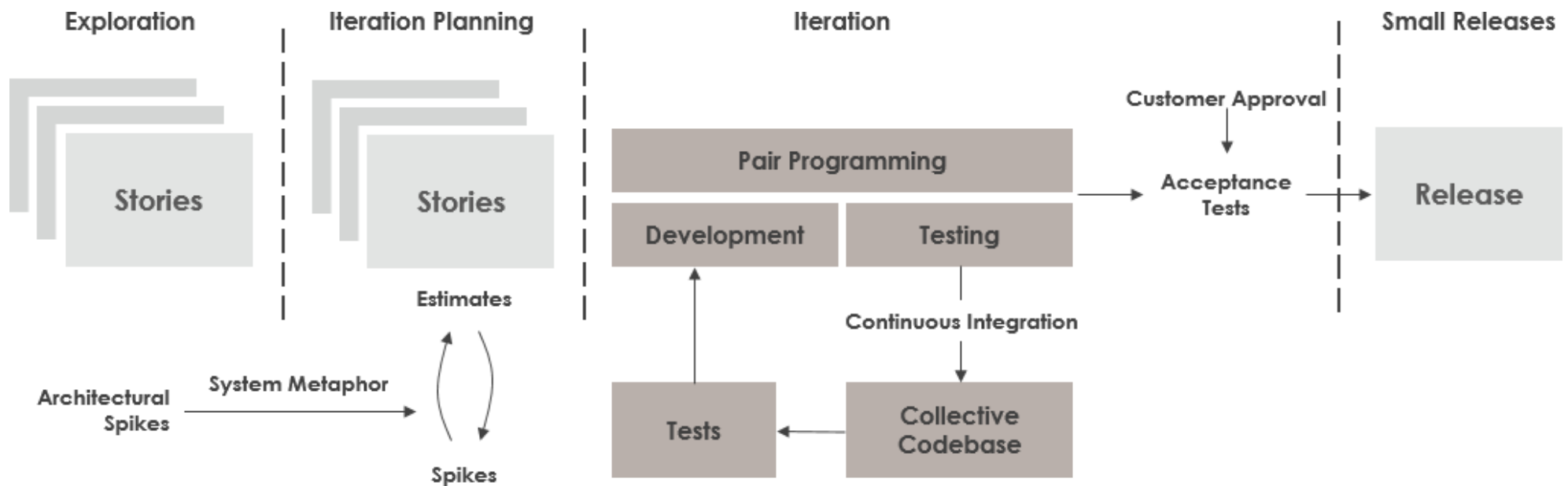
- Usare convenzioni di codifica
 - A causa del Pair Programming, delle rifattorizzazioni e della proprietà collettiva del codice, occorre poter addentrarsi velocemente nel codice altrui
- Commentare il codice
 - Priorità al codice che “svela” il suo scopo
 - Se il codice richiede un commento, riscrivilo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

<https://peps.python.org/pep-0008/>

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

https://en.wikibooks.org/wiki/Computer_Programming/Coding_Style

XP: il processo



Il 13° 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

Conclusioni

I metodi agili, e XP in particolare, prediligono:

- Piccoli team
- Il cliente è nel team - il Product Owner
- Accettano i cambiamenti dei requisiti
- Iterazioni brevi e consegne frequenti
- Praticano l'integrazione continua

Autotest

- Cos'è un MVP (minimum viable product)?
- Quali sono i valori agili?
- Cos'è una user story?
- A cosa serve il metodo MOSCOW?
- Cos'è la proprietà collettiva del codice?
- Cos'è il refactoring?

Lecture raccomandate

- Cohn e Ford, Introducing an agile process to an organization, *IEEE Computer*, 2003
- Janzen & Saiedian, Does Test-Driven Development Really Improve Software Design Quality?, *IEEE Software*, March/April 2008

Riferimenti

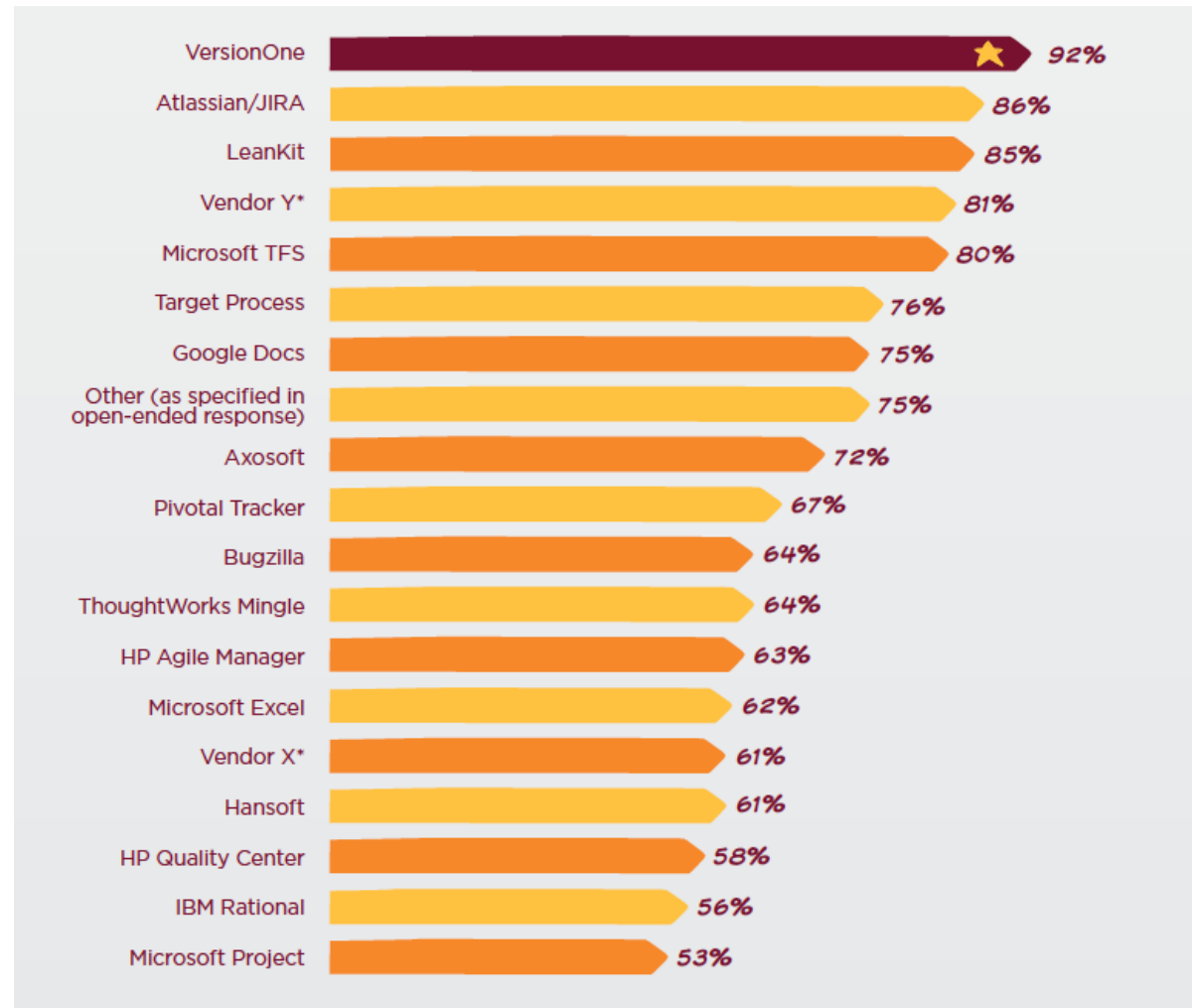
- Beck e altri, Manifesto for Agile development, 2001 agilemanifesto.org
- Beck & Andres, *Extreme Programming Explained: Embrace Change*, AW 2004
- Wilson, *Building software together*, <https://buildtogether.tech>

Siti e blog

- Extreme Programming www.extremeprogramming.org
- Uncle Bob's blog blog.cleancoder.com
- <https://refactoring.com>
- xp123.com/articles/

Strumenti

- Trello/Taiga
- Slack/Mattermost
- Ant, XDoclet, JUnit, Cactus, Maven
- Git/GitLab
- Atlassian Jira
- agiletrack.net
- www.planningpoker.com
- www.collab.net



Modelli di processo 2

Domande?

