Il processo software

- **Processo software**
  - serie di attività necessarie alla realizzazione del prodotto software nei tempi, con i costi e con le desiderate caratteristiche di qualità.
- Nel suo contesto:
  - si applicano metodi, tecniche e strumenti
  - si creano prodotti (sia intermedi che finali)
  - si stabilisce il controllo gestionale del progetto
  - si garantisce la qualità
  - si governano le modifiche

Fasi del processo

- Come visto, il processo software segue un **ciclo di vita** che si articola in 3 stadi (sviluppo, manutenzione, dismissione). Nel primo stadio si possono riconoscere due tipi di fasi:
  - fasi di tipo **definizione**
  - fasi di tipo **produzione**
- Le fasi di **definizione** si occupano di "*cosa*" il software deve fornire. Si definiscono i requisiti, si producono le specifiche
- Le fasi di **produzione** definiscono "*come*" realizzare quanto ottenuto con le fasi di definizione. Si progetta il software, si codifica, si integra e si rilascia al cliente
- Lo stadio di *manutenzione* è a supporto del software realizzato e prevede fasi di definizione e/o produzione al suo interno
- Durante ogni fase si procede ad effettuare il **testing** di quanto prodotto, mediante opportune tecniche di *verifica e validazione* (*V&V*) applicate sia ai prodotti intermedi che al prodotto finale

Tipi di manutenzione

- Manutenzione **correttiva**, che ha lo scopo di eliminare i difetti (*fault*) che producono guasti (*failure*) del software
- Manutenzione **adattativa**, che ha lo scopo di adattare il software ad eventuali cambiamenti a cui è sottoposto l'ambiente operativo per cui è stato sviluppato
- Manutenzione **perfettiva**, che ha lo scopo di estendere il software per accomodare funzionalità aggiuntive
- Manutenzione **preventiva** (o *software reengineering*), che consiste nell'effettuare modifiche che rendano più semplici le correzioni, gli adattamenti e le migliorie

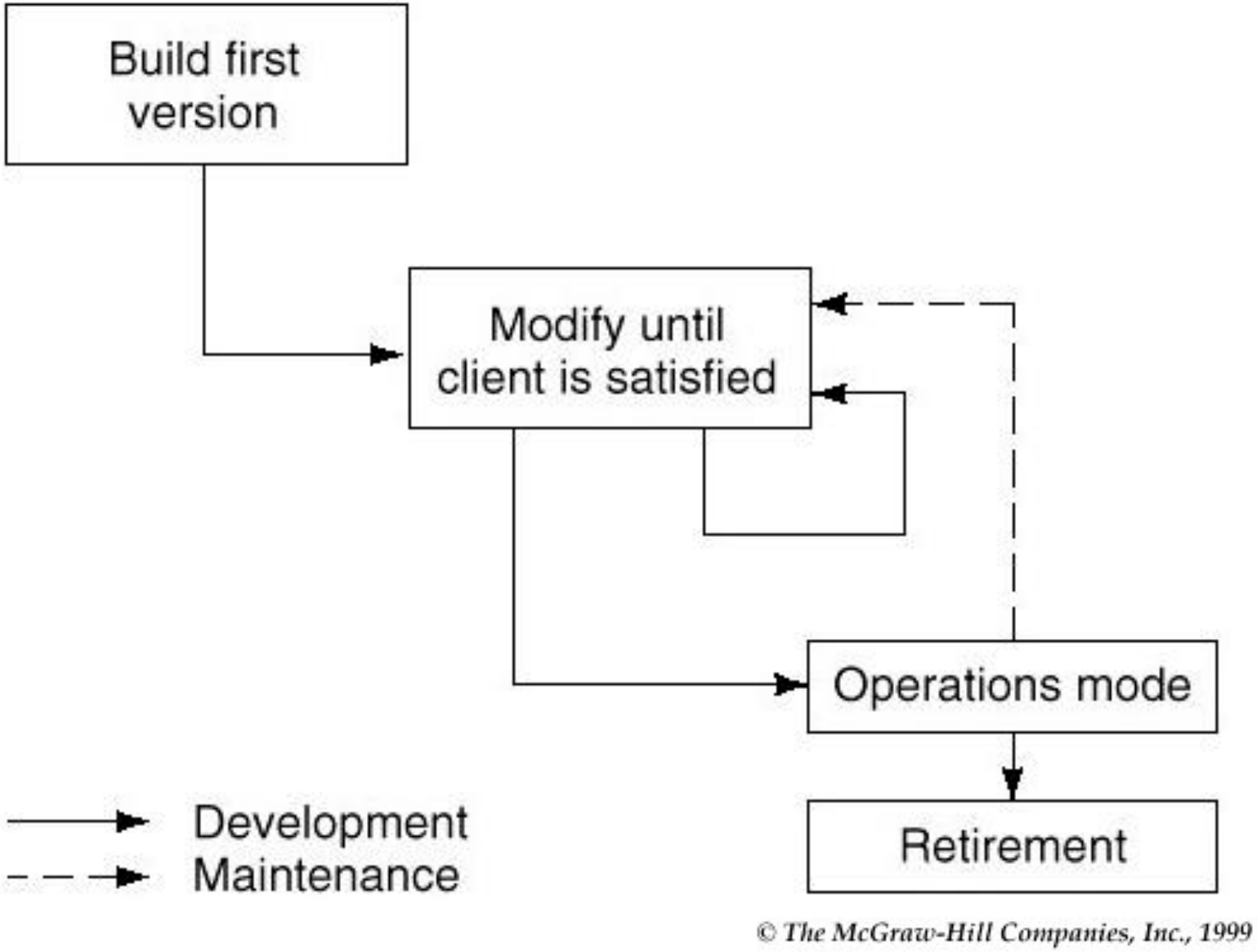
Definizione di ciclo di vita

- Def. IEEE Std 610-12 (*Software Eng. Terminology*)
  - intervallo di tempo che intercorre tra l'istante in cui nasce l'esigenza di costruire un prodotto software e l'istante in cui il prodotto viene dismesso
  - include le fasi di definizione dei requisiti, specifica, pianificazione, progetto preliminare, progetto dettagliato, codifica, integrazione, testing, uso, manutenzione e dismissione
  - *Nota*: tali fasi possono sovrapporsi o essere eseguite in modo iterativo

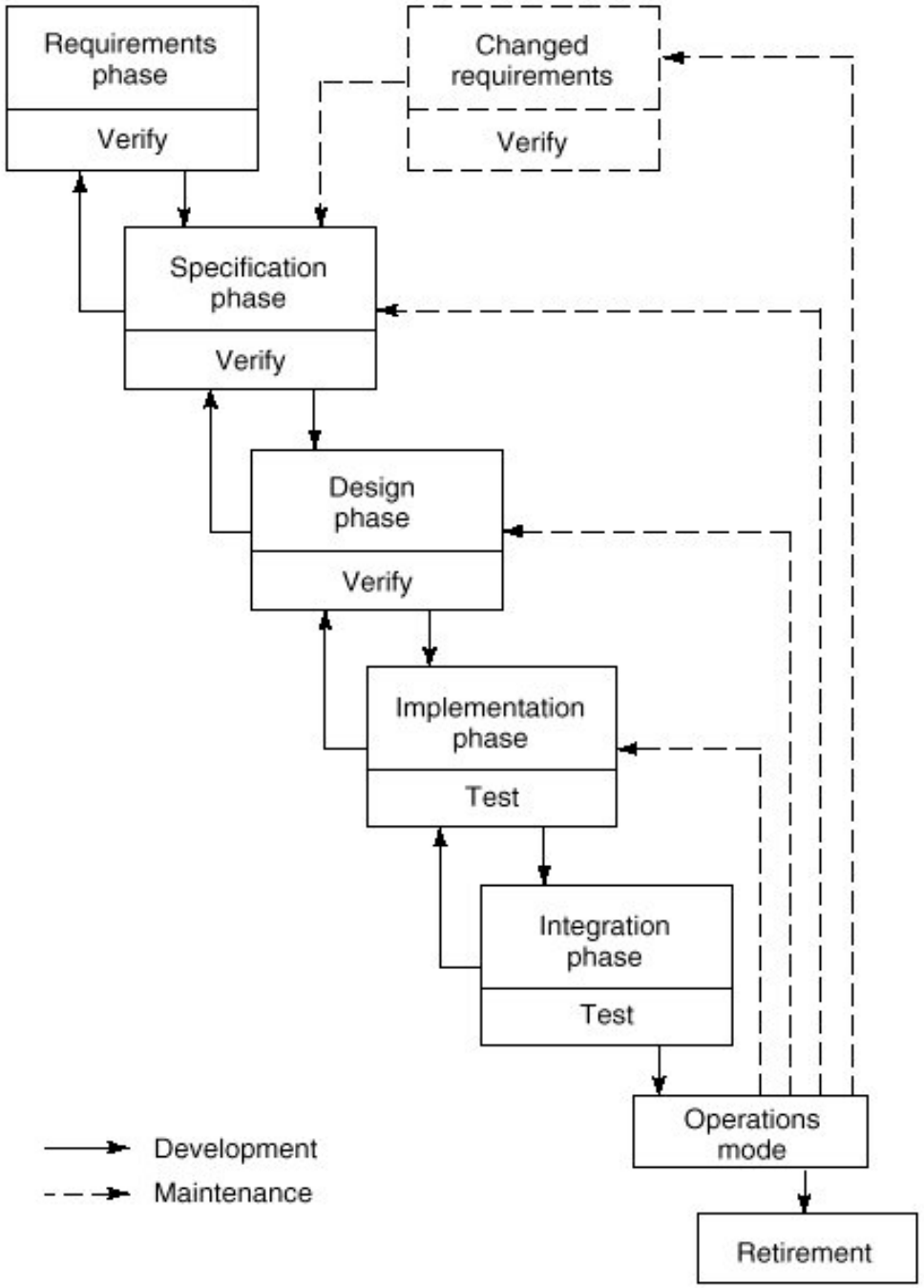
Modelli di ciclo di vita

- Il **modello del ciclo di vita** del software specifica la serie di fasi attraverso cui il prodotto software progredisce e l'ordine con cui vanno eseguite, dalla definizione dei requisiti alla dismissione
- La scelta del modello dipende dalla natura dell'applicazione, dalla maturità dell'organizzazione, da metodi e tecnologie usate e da eventuali vincoli dettati dal cliente
- L'assenza di un modello del ciclo di vita corrisponde ad una modalità di sviluppo detta "*build & fix*" (o "*fix-it-later*"), in cui il prodotto software viene sviluppato e successivamente rilavorato fino a soddisfare le necessità del cliente

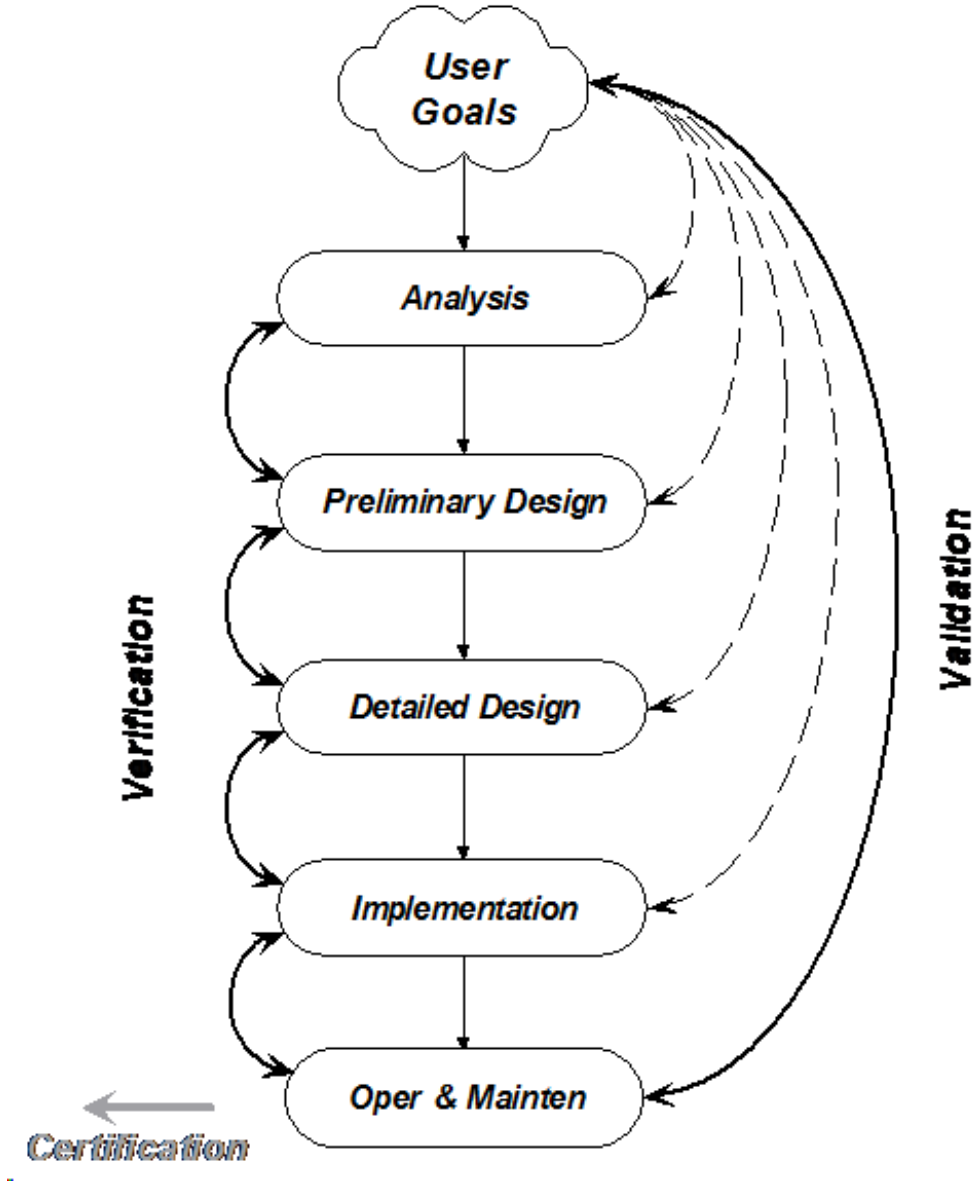
Build&Fix



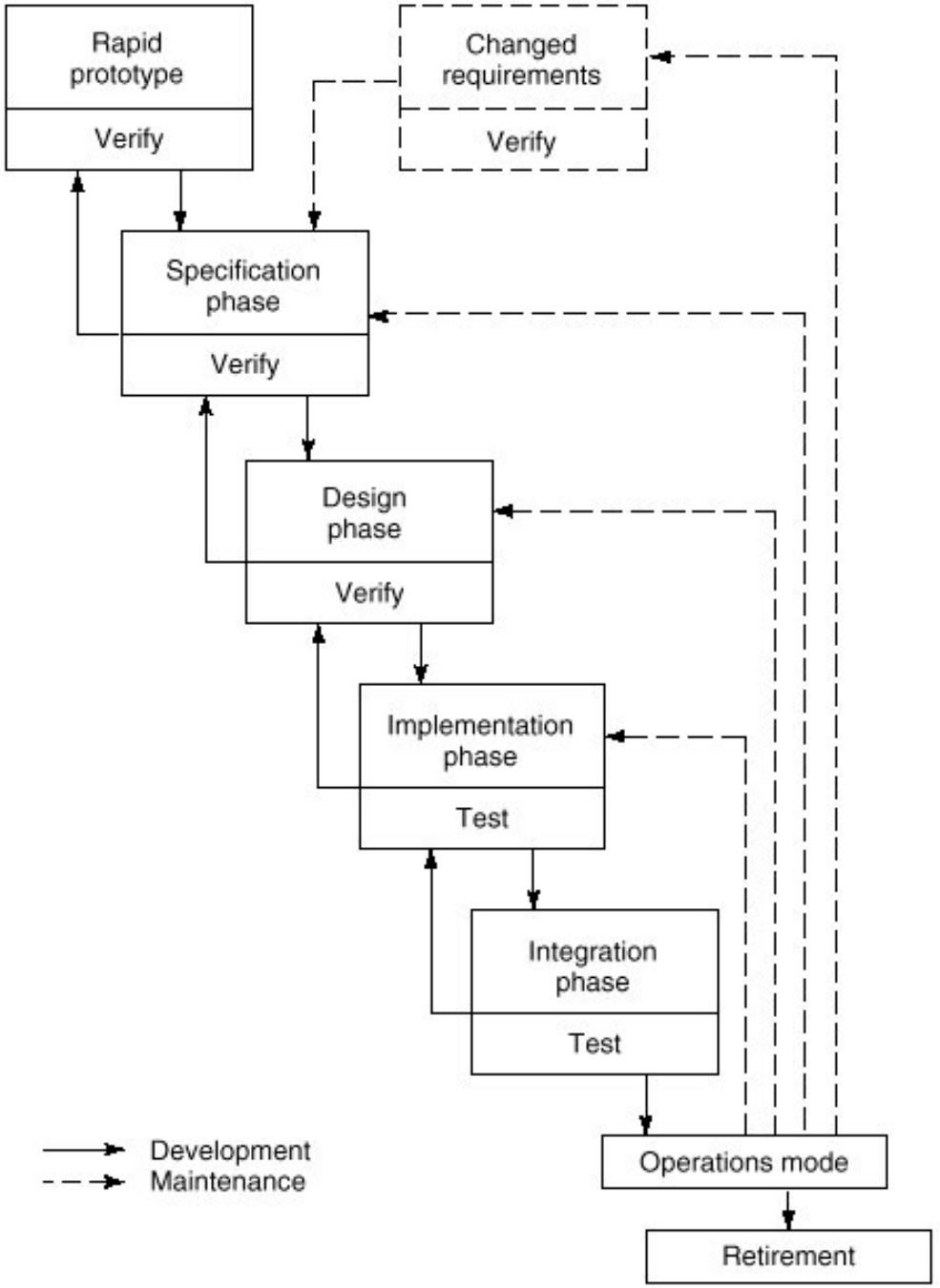
Modello Waterfall



Verification & Validation (V&V) nel Waterfall



Rapid Prototyping Model



Software Prototyping

Rapid software development to

elicit or *validate* requirements

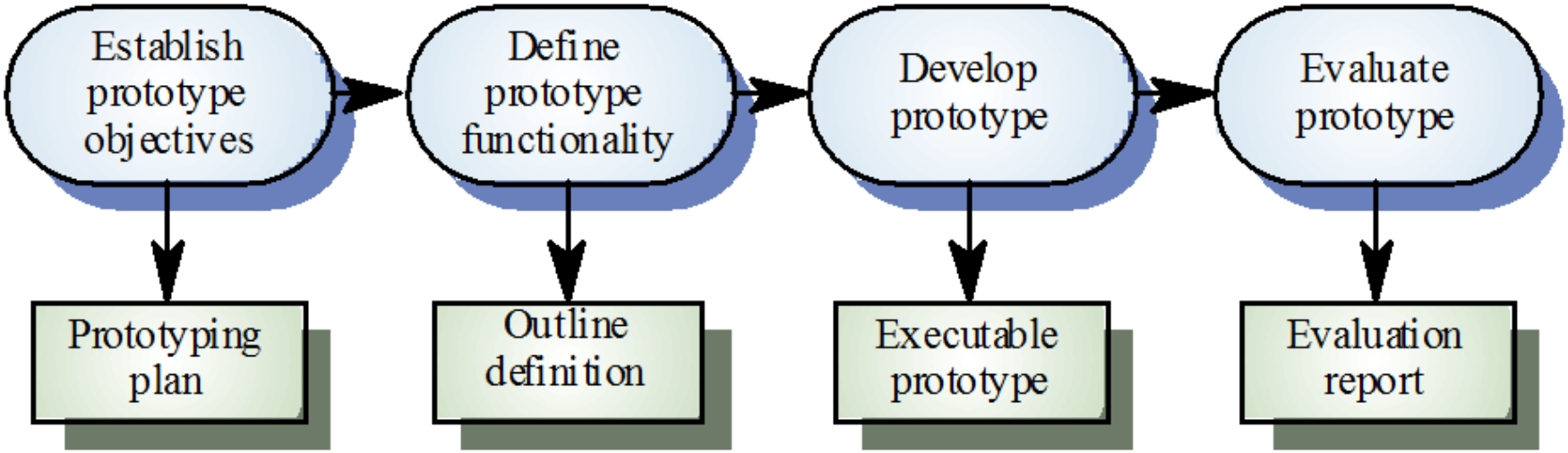
Uses of system prototypes

- The principal use is to help customers and developers understand the software requirements
  - Requirements elicitation: users can experiment with a prototype to see how the system supports their work
  - Requirements validation: the prototype can reveal errors and omissions in the requirements
- Prototyping can be considered as a risk reduction activity which reduces requirements risks

Prototyping benefits

- Misunderstandings between software users and developers are exposed
- Missing services may be detected and confusing services may be identified
- A working system is available early in the process
- The prototype may serve as a basis for deriving a software specification
- The prototype can support user training and product testing

Prototyping process



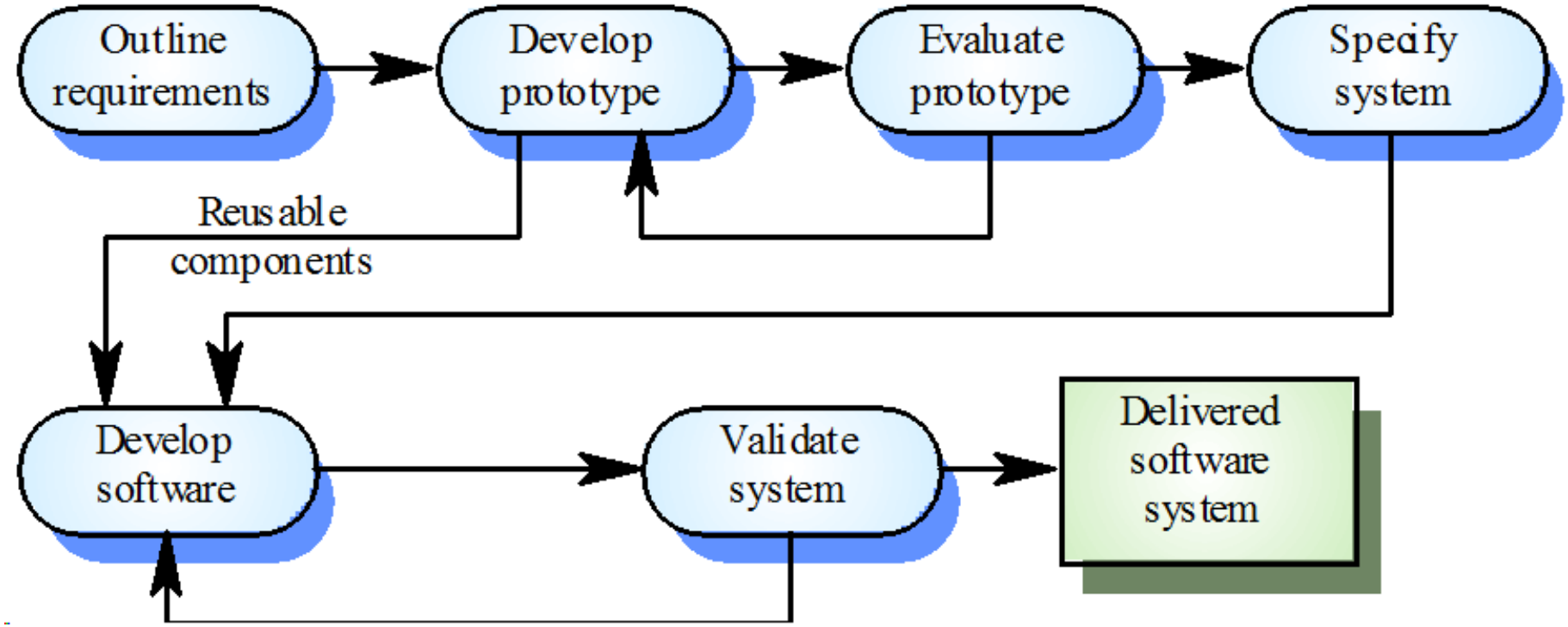
Prototypes as specifications

- Some parts of the requirements (e.g. safety-critical functions) may be impossible to prototype and so do not appear in the specification
- An implementation has no legal standing as a contract
- Non-functional requirements cannot be adequately tested in a software prototype

Throw-away prototyping

- A prototype which is usually a practical implementation of the product is produced to help discover requirements problems and then discarded. The product is then developed using some other development process
- Used to reduce requirements risk
- The prototype is developed from an initial requirement, delivered for experiment then discarded
- The throw-away prototype should NOT be considered as a final product
  - Some characteristics may have been left out
  - There is no specification for long-term maintenance
  - The product will be poorly structured and difficult to maintain

Throw-away prototyping process



Throw-away prototype delivery

- Developers may be pressurised to deliver a throw-away prototype as a final product
- This is not recommended
  - It may be impossible to tune the prototype to meet non-functional requirements
  - The prototype is inevitably undocumented
  - The structure will be degraded through changes made during development
  - Normal organisational quality standards may not have been applied

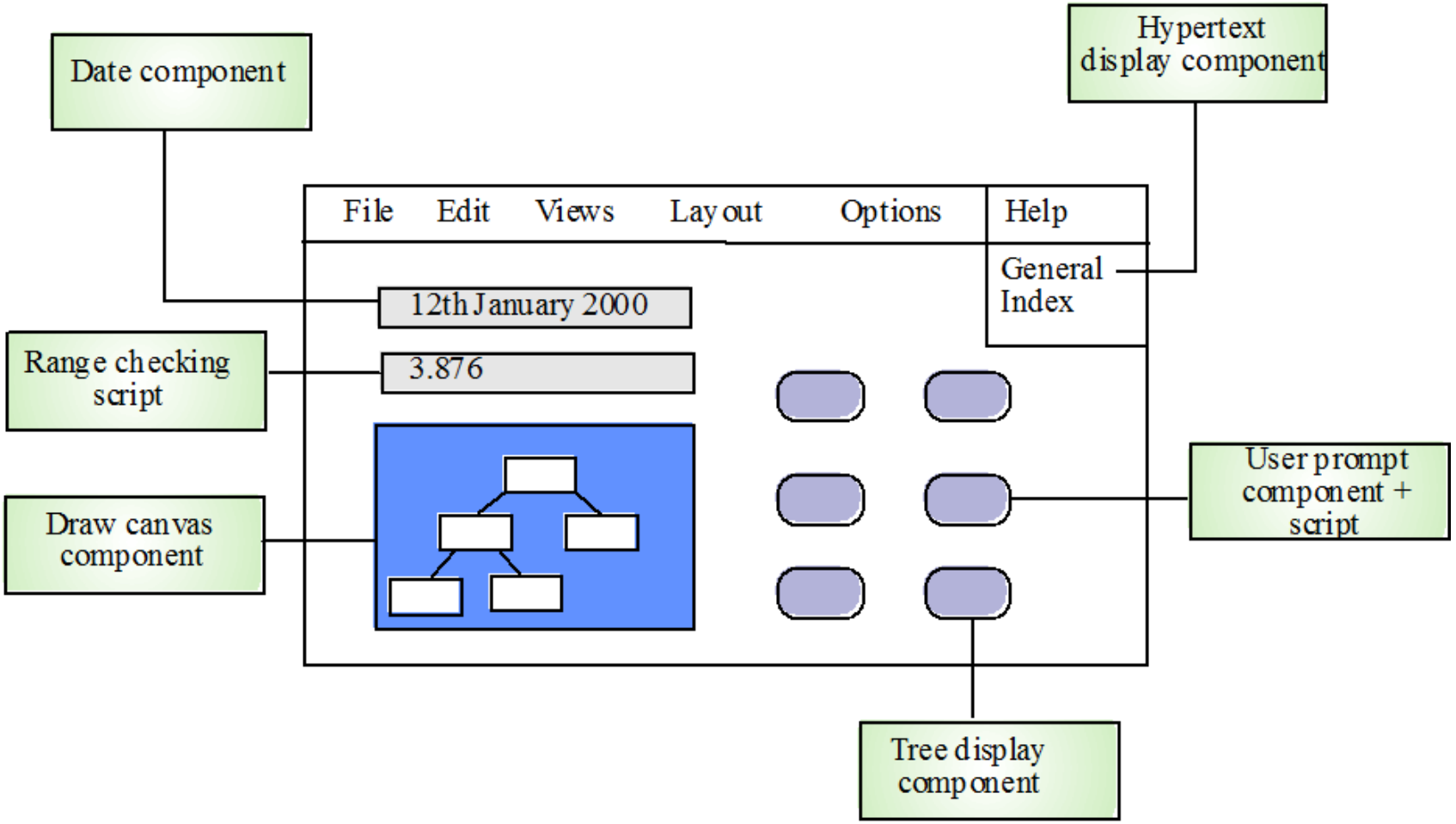
Prototyping key points

- A prototype can be used to give end-users a concrete impression of the product's capabilities
- Prototyping is becoming increasingly used for product development where rapid development is essential
- Throw-away prototyping is used to understand the product requirements
- Rapid development of prototypes is essential. This may require leaving out functionality or relaxing non-functional constraints
- Visual programming is an inherent part of most prototype development methods

Visual programming

- Scripting languages such as Visual Basic support visual programming where the prototype is developed by creating a user interface from standard items and associating components with these items
- A large library of components exists to support this type of development
- These may be tailored to suit the specific application requirements

Visual programming (2)



Problems with visual development

- Difficult to coordinate team-based development
- No explicit software architecture
- Complex dependencies between parts of the program can cause maintainability problems

Process iteration

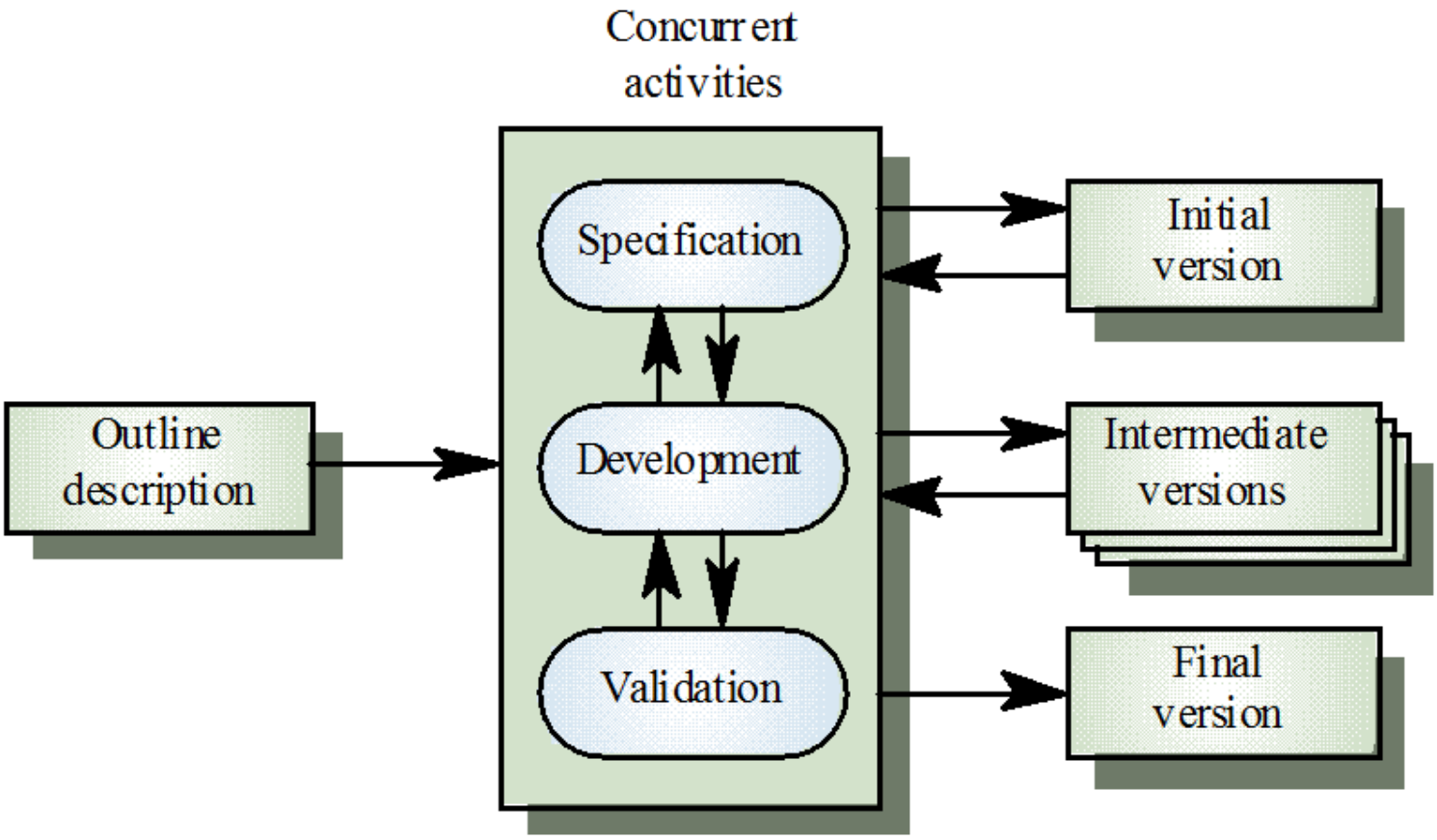
- Requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large products
- Iteration can be applied to any of the generic process models
- Two (related) approaches
  - Incremental development
  - Spiral development

Incremental development

- The product is developed and delivered in increments after establishing an overall architecture
- Requirements and specifications for each increment may be developed
- Users may experiment with delivered increments while others are being developed. Therefore, these serve as a form of prototype
- Intended to combine some of the advantages of prototyping but with a more manageable process and better structure

Modello incrementale

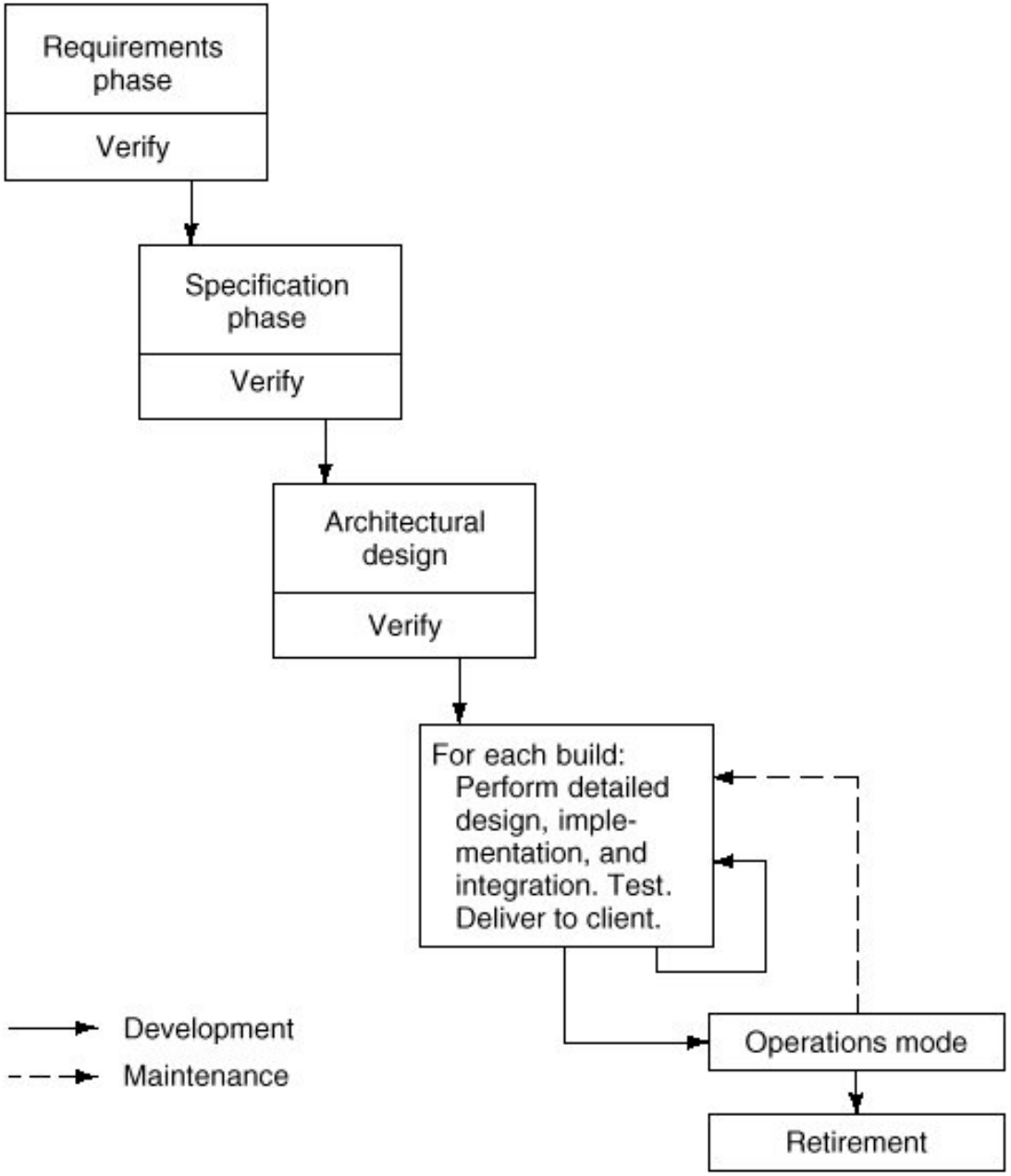
- Il prodotto software viene sviluppato e rilasciato per incrementi (*build*) successivi



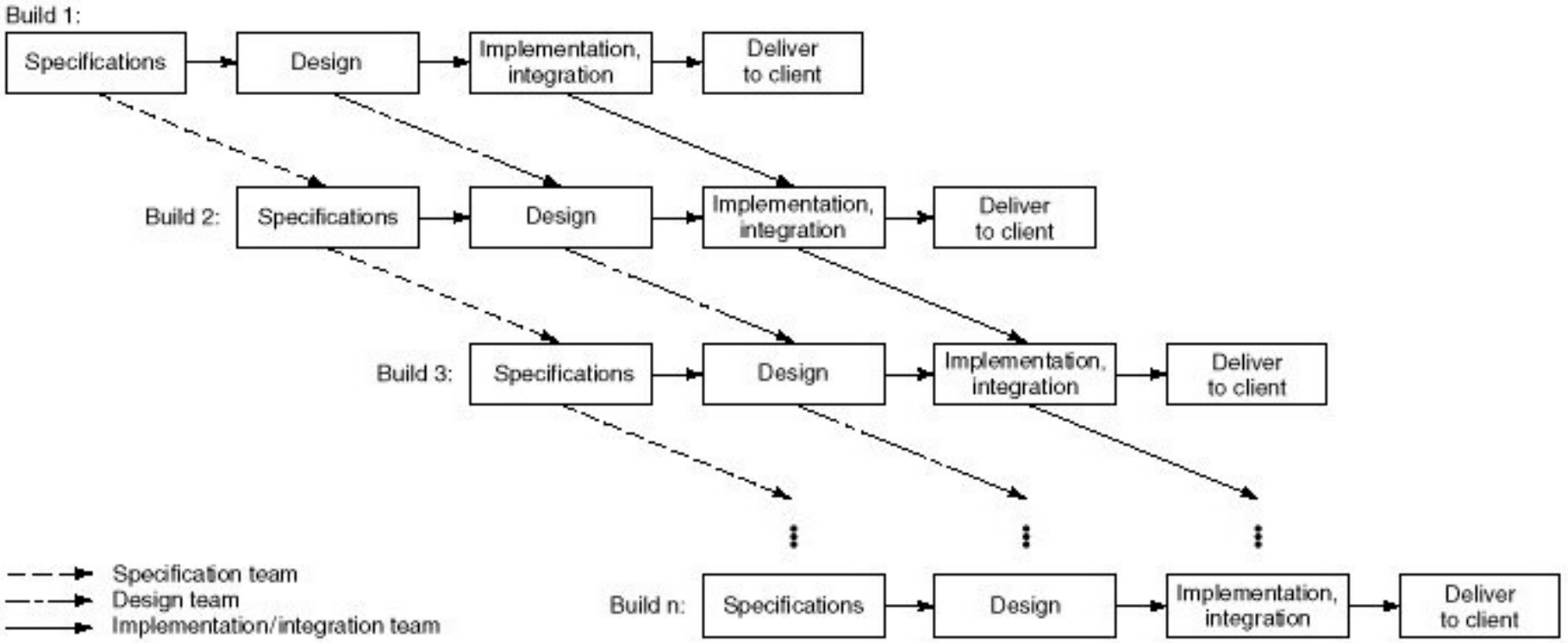
Modello incrementale (cont.)

- Include aspetti tipici del modello basato su *rapid prototyping* (l'utente può sperimentare l'utilizzo del prodotto contenente gli incrementi consegnati, mentre i restanti sono ancora in fase di sviluppo)
- Si rivela efficace quando il cliente vuole continuamente verificare i progressi nello sviluppo del prodotto e quando i requisiti subiscono modifiche
- Può essere realizzato in due versioni alternative:
  - versione con *overall architecture*
  - versione senza *overall architecture* (più rischiosa)

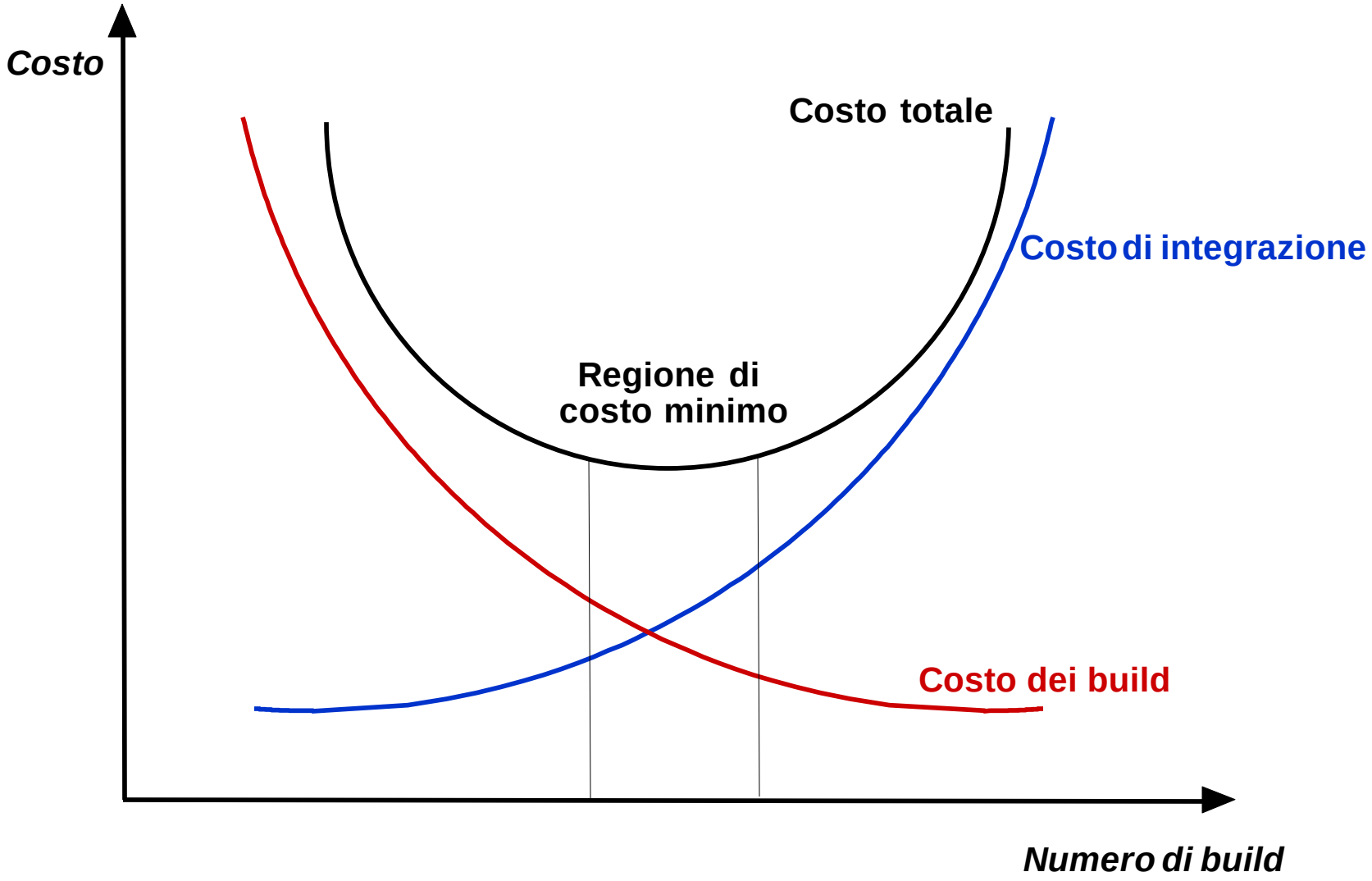
Versione con *overall architecture*



Versione senza *overall architecture*



Impatto sui costi del software



Confronto con modello a cascata

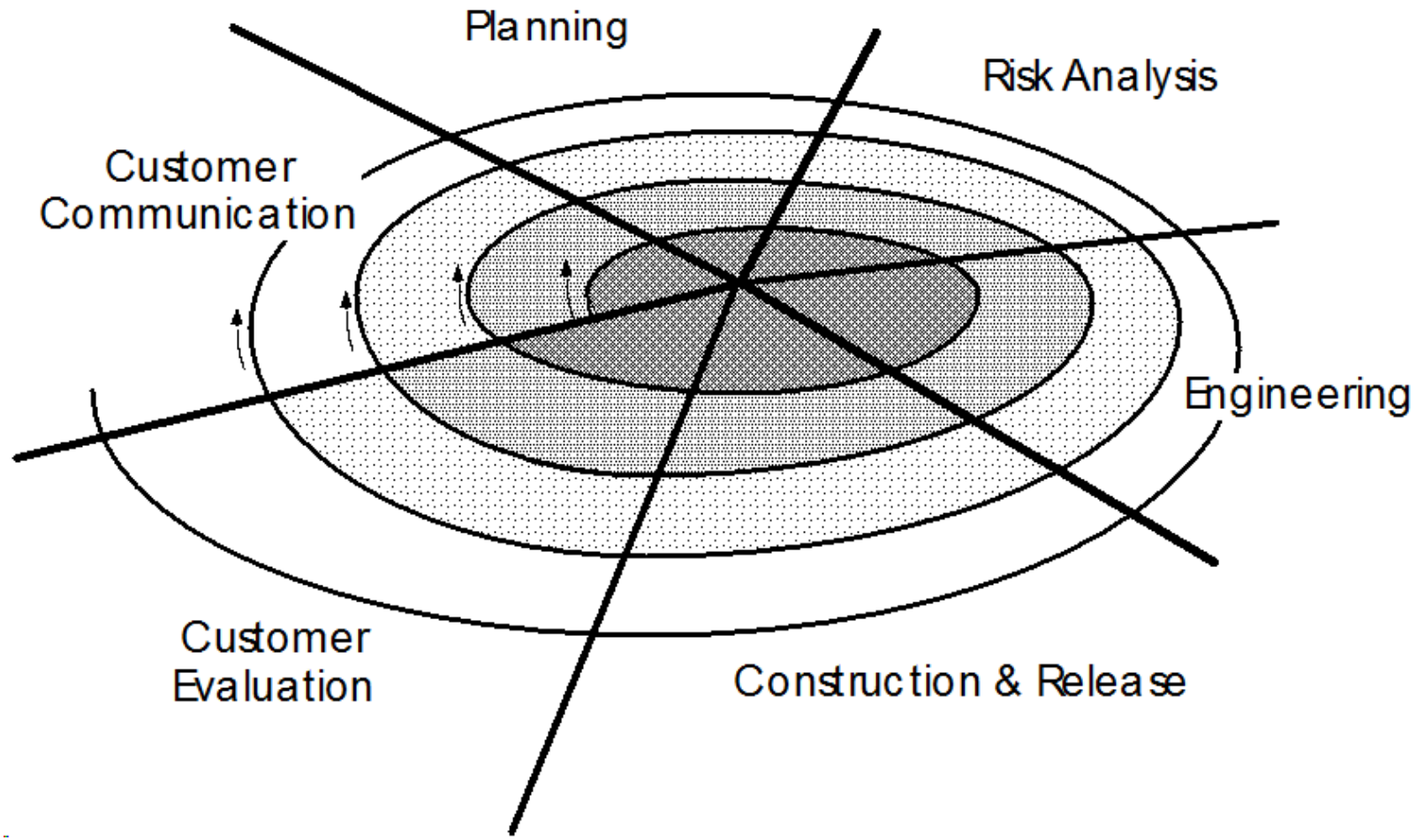
Modello a cascata

- Requisiti “congelati” al termine della fase di specifica
- Feedback del cliente solo una volta terminato lo sviluppo
- Fasi condotte in rigida sequenza (l’output di una costituisce input per la successiva)
- Prevede fasi di progetto dettagliato e codifica dell’intero prodotto
- Team di sviluppo costituito da un numero elevato di persone

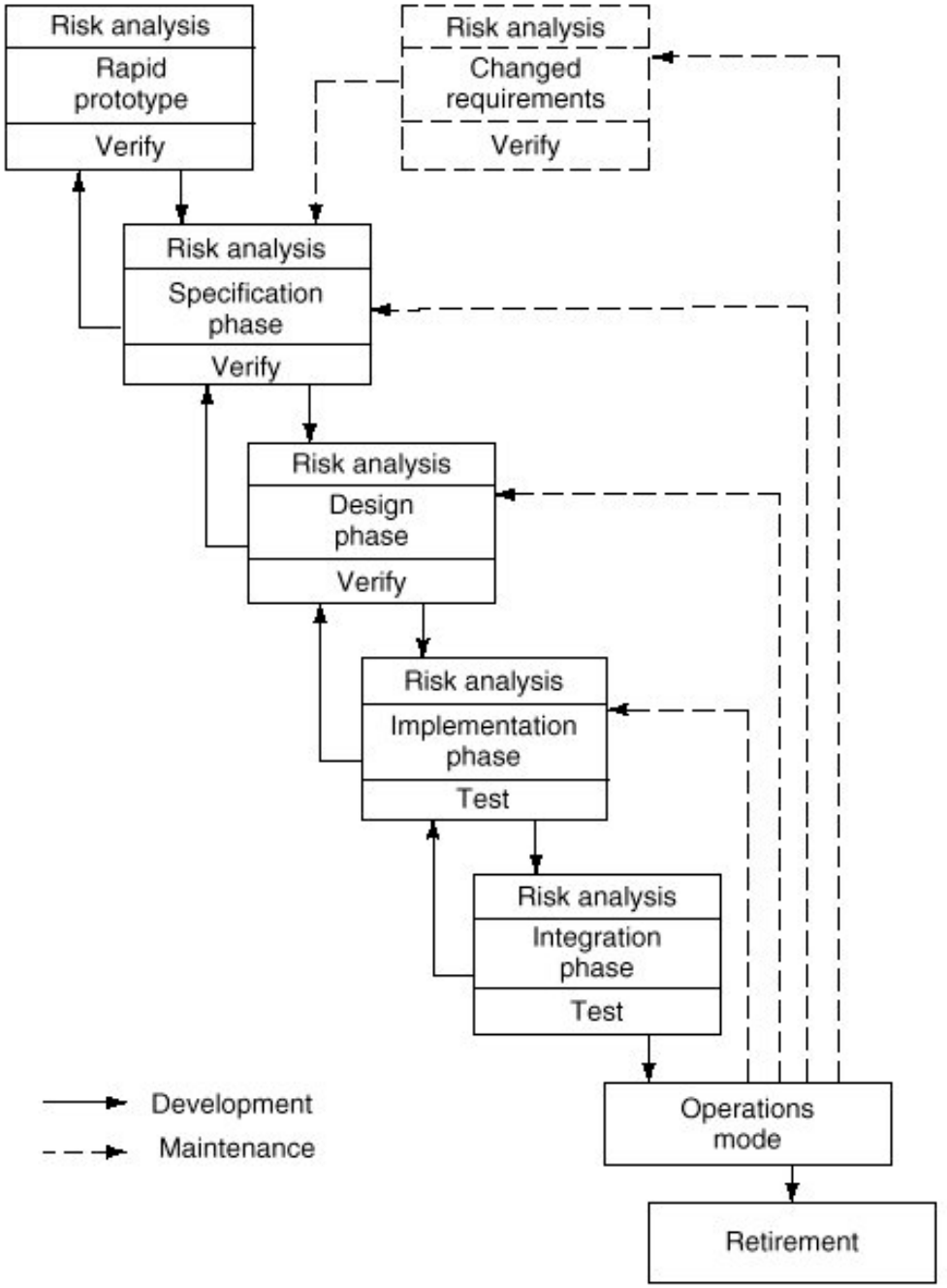
Modello incrementale

- Requisiti suddivisi in classi di priorità e facilmente modificabili
- Continuo feedback da parte del cliente durante lo sviluppo
- Fasi che possono essere condotte in parallelo
- Progetto dettagliato e codifica vengono effettuate sul singolo *build*
- Differenti team di sviluppo, ciascuno di piccole dimensioni

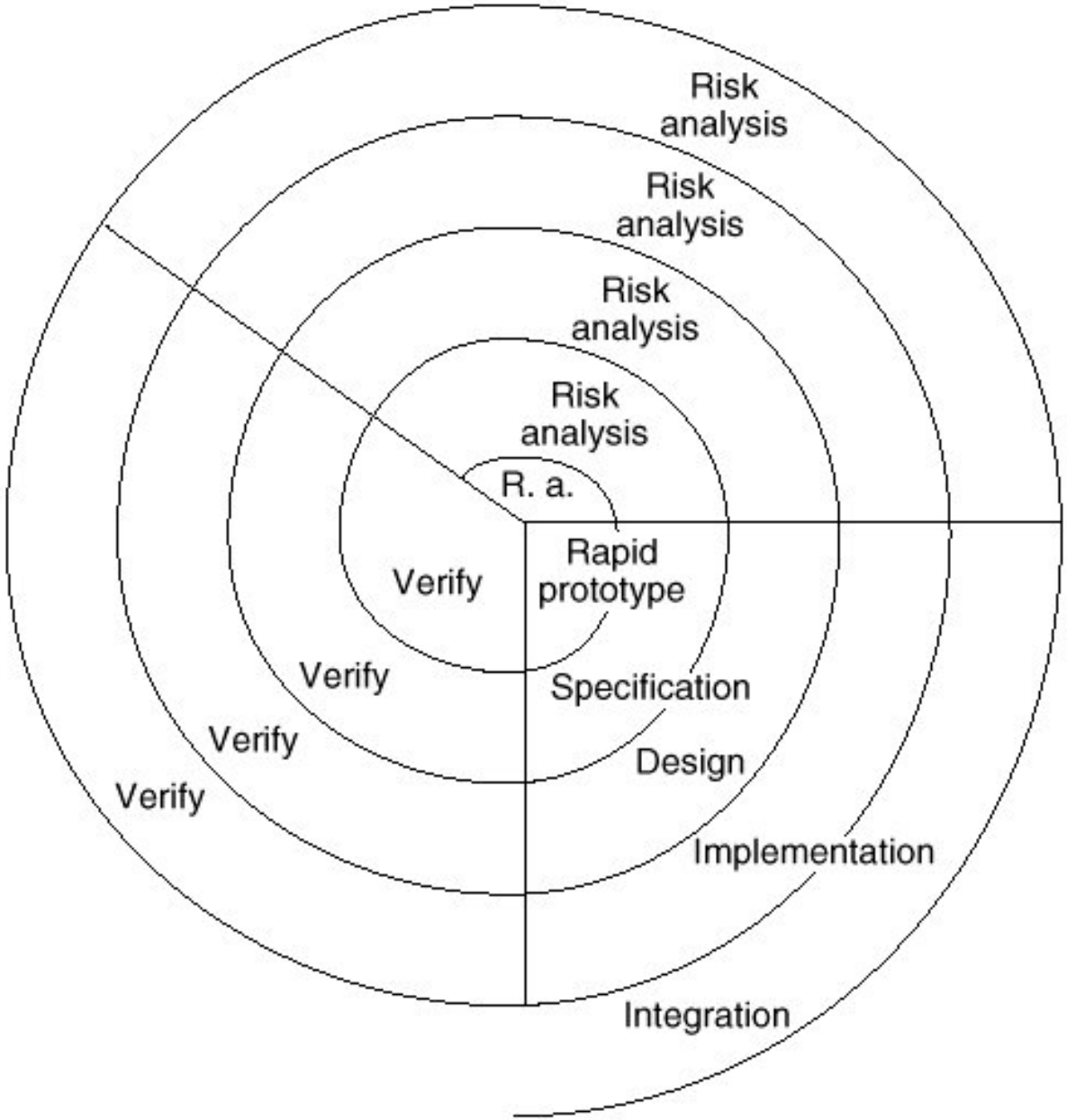
Modello a spirale



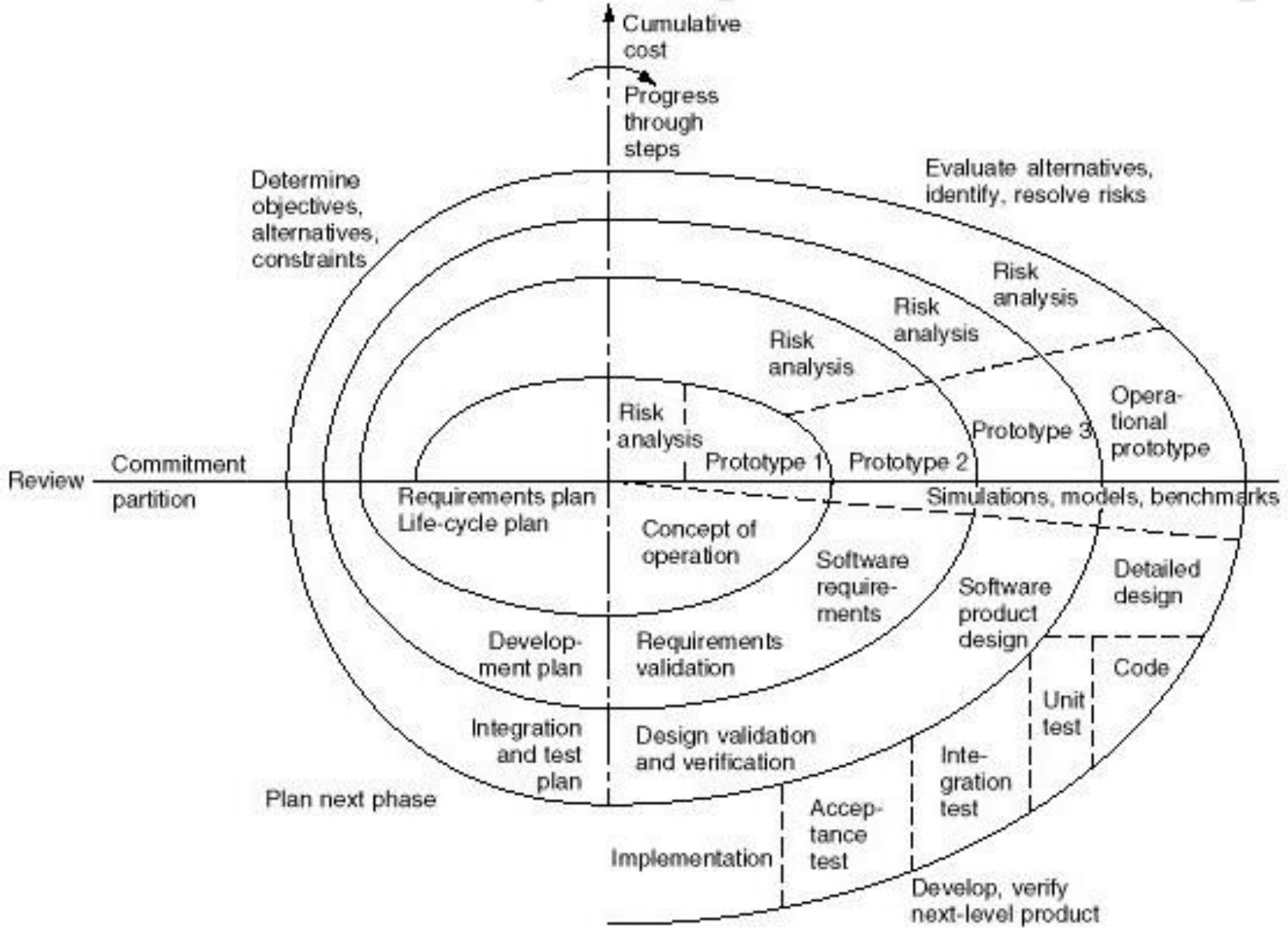
Modello a spirale semplificato (versione lineare)



Modello a spirale semplificato



Modello full-spiral [Boehm, 1988]



Risk management

- Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project
- A *risk* is a *probability* that some adverse circumstance will occur
- Categories of risk
  - *Project* risks affect schedule or resources
  - *Product* risks affect the quality or performance of the software being developed
  - *Business* risks affect the organisation developing or procuring the software

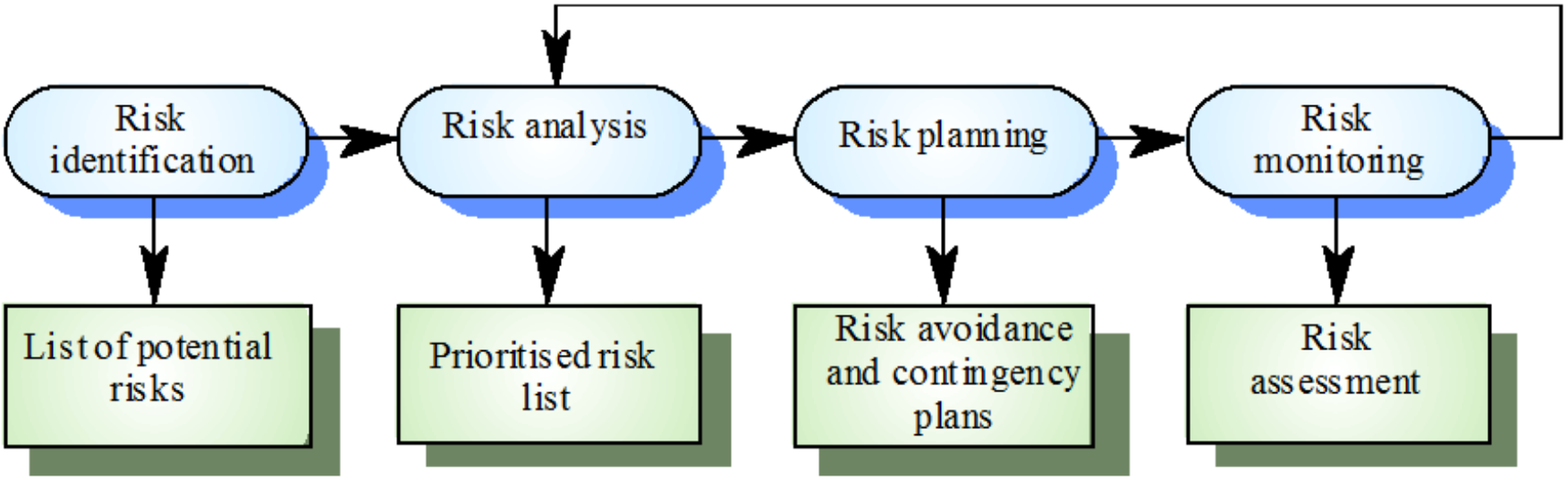
Risks by category

Risk	Risk type	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organisational management with different priorities.
Hardware unavailability	Project	Hardware which is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool under-performance	Product	CASE tools which support the project do not perform as anticipated
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

The risk management process

- Risk identification
  - Identify project, product and business risks
- Risk analysis
  - Assess the likelihood and consequences of these risks
- Risk planning
  - Draw up plans to avoid or minimise the effects of the risk
- Risk monitoring
  - Monitor the risks throughout the project

The risk management process (2)



Risk identification (1)
Risk types

- Technology risks
- People risks
- Organisational risks
- Tools risks
- Requirements risks
- Estimation risks

Risk identification (2)

Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. Software components which should be reused contain defects which limit their functionality.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unavailable at critical times. Required training for staff is not available.
Organisational	The organisation is restructured so that different management are responsible for the project. Organisational financial problems force reductions in the project budget.
Tools	The code generated by CASE tools is inefficient. CASE tools cannot be integrated.
Requirements	Changes to requirements which require major design rework are proposed. Customers fail to understand the impact of requirements changes.
Estimation	The time required to develop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

Risk analysis (1)

- Assess probability and seriousness of each risk
- Risk *probability* may be:
  - very low (<10%)
  - low (10-25%)
  - moderate (25-50%)
  - high (50-75%)
  - very high (>75%)
- Risk *effects* might be catastrophic, serious, tolerable or insignificant

Risk analysis (2)

Risk	Probability	Effects
Organisational financial problems force reductions in the project budget.	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project.	High	Catastrophic
Key staff are ill at critical times in the project.	Moderate	Serious
Software components which should be reused contain defects which limit their functionality.	Moderate	Serious
Changes to requirements which require major design rework are proposed.	Moderate	Serious
The organisation is restructured so that different management are responsible for the project.	High	Serious
The database used in the system cannot process as many transactions per second as expected.	Moderate	Serious
The time required to develop the software is underestimated.	High	Serious
CASE tools cannot be integrated.	High	Tolerable
Customers fail to understand the impact of requirements changes.	Moderate	Tolerable
Required training for staff is not available.	Moderate	Tolerable
The rate of defect repair is underestimated.	Moderate	Tolerable
The size of the software is underestimated.	High	Tolerable
The code generated by CASE tools is inefficient.	Moderate	Insignificant

Risk analysis (3)

- Identify e.g., the *top-ten risks* by considering:
  - all *catastrophic* risks
  - all *serious* risks that have more than a *moderate* probability of occurrence
- Rank such risks by order of importance

Risk planning

- Consider each risk and develop a strategy to manage that risk
- Avoidance strategies
  - The probability that the risk will arise is reduced
- Minimisation strategies
  - The impact of the risk on the project or product will be reduced
- Contingency plans
  - If the risk arises, contingency plans are strategies to deal with that risk

Risk management strategies

Risk	Strategy
Organisational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Recruitment problems	Alert customer of potential difficulties and the possibility of delays, investigate buying-in components.
Staff illness	Reorganise team so that there is more overlap of work and people therefore understand each other's jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.
Requirements changes	Derive traceability information to assess requirements change impact, maximise information hiding in the design.
Organisational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying in components, investigate use of a program generator.

Risk monitoring (1)

- Assess each identified risks regularly to decide whether or not it is becoming less or more probable
- To perform assessment look at ***risk factors*** (see next slide)
- Also assess whether the effects of the risk have changed (in such case go back to risk analysis)
- Each key risk should be discussed at management progress meetings

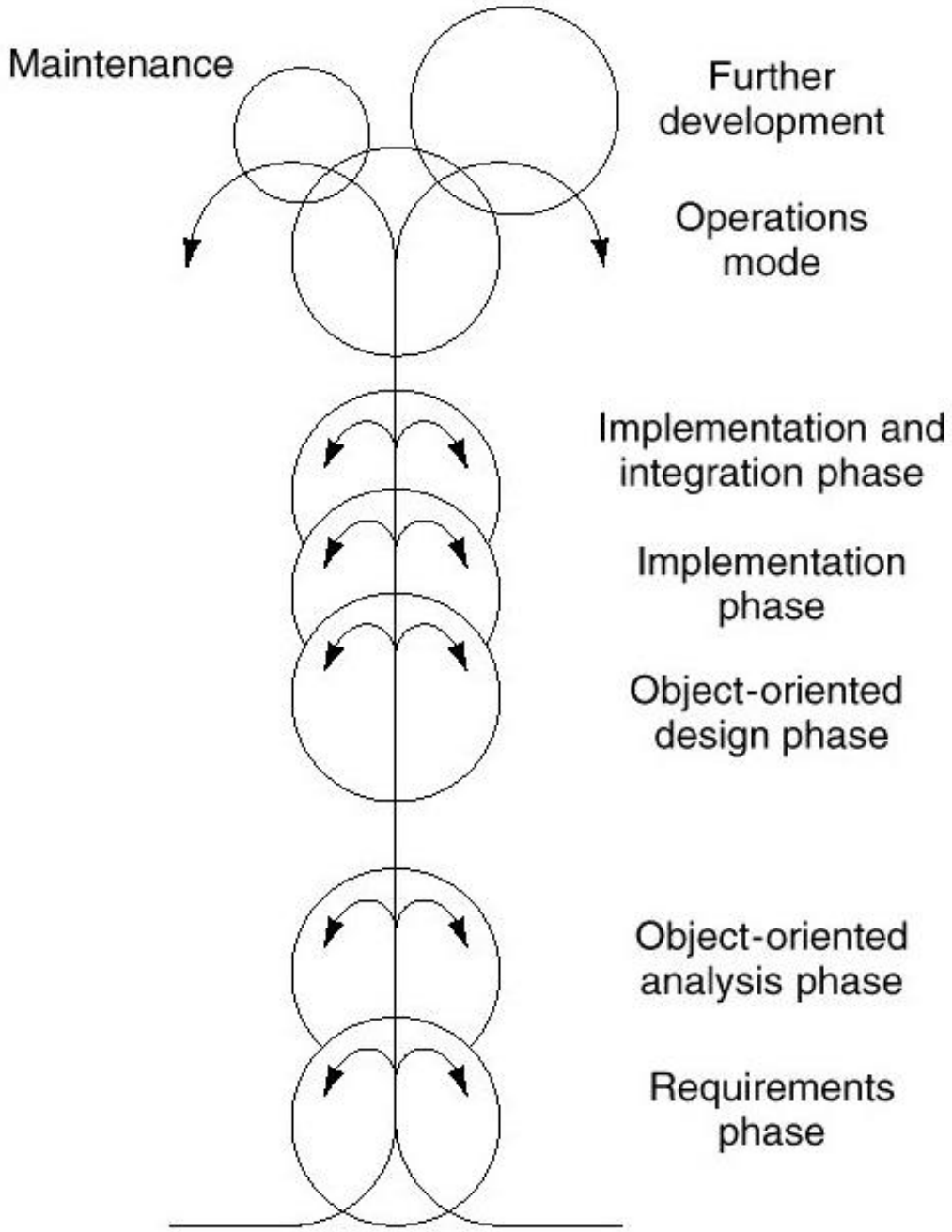
Risk monitoring (2)

Risk factors

Risk type	**Potential indicators**
Technology	Late delivery of hardware or support software, many reported technology problems
People	Poor staff morale, poor relationships amongst team member, job availability
Organisational	organisational gossip, lack of action by senior management
Tools	reluctance by team members to use tools, complaints about CASE tools, demands for higher-powered workstations
Requirements	many requirements change requests, customer complaints
Estimation	failure to meet agreed schedule, failure to clear reported defects

Altri modelli (1)

Modello object-oriented



Altri modelli (2)

- **Modello di *ingegneria simultanea (o concorrente)***
  - ha come obiettivo la riduzione di tempi e costi di sviluppo, mediante un approccio sistematico al progetto integrato e concorrente di un prodotto software e del processo ad esso associato.
  - Le fasi di sviluppo coesistono invece di essere eseguite in sequenza.
- **Modello basato su *metodi formali***
  - comprende una serie di attività che conducono alla specifica formale matematica del software, al fine di eliminare ambiguità, incompletezze ed inconsistenze e facilitare la verifica dei programmi mediante l'applicazione di tecniche matematiche.
  - La ***Cleanroom Software Engineering*** (1987) ne rappresenta un esempio di realizzazione, in cui viene enfatizzata la possibilità di rilevare i difetti del software in modo più tempestivo rispetto ai modelli tradizionali

Il Modello Microsoft[1]

- La Microsoft, come altre organizzazioni che sviluppano software commerciale, ha dovuto affrontare, fin dalla metà degli anni 80, problemi di:
  - incremento della qualità dei prodotti software
  - riduzione di tempi e costi di sviluppo
- Per cercare di risolvere tali problemi si è adottato un processo che è al tempo stesso *iterativo*, *incrementale* e *concorrente* e che permette di esaltare le doti di creatività delle persone coinvolte nello sviluppo di prodotti software

[1] M.A. Cusumano and R.W. Selby, How Microsoft Builds Software, *Communications of the ACM*, vol. 40, n. 6, June 1997.

Approccio *synch-and-stabilize*

- L'approccio usato attualmente da Microsoft è noto come "***synchronize-and-stabilize***"
- Tale approccio è basato su:
  - **sincronizzazione** quotidiana delle attività svolte da persone che lavorano sia individualmente che all'interno di piccoli team (da 3 a 8 persone), mediante assemblaggio dei componenti software sviluppati (anche parzialmente) in un prodotto (*daily build*) che viene testato e corretto
  - **stabilizzazione** periodica del prodotto in incrementi (*milestone*) successivi durante l'avanzamento del progetto, piuttosto che un'unica volta alla fine

Ciclo di sviluppo a 3 fasi

- *Planning* phase
  - Define product vision, specification and schedule
- *Development* phase
  - Feature development in 3/4 sequential subprojects, each resulting in a milestone release
- *Stabilization* phase
  - Comprehensive internal and external testing, final product, stabilization and ship

Planning phase

Planning Phase Define product vision, specification, and schedule

- **Vision Statement** Product and program management use extensive customer input to identify and priority-order product features.
- **Specification Document** Based on vision statement, program management and development group define feature functionality, architectural issues, and component interdependencies.
- **Schedule and Feature Team Formation** Based on specification document, program management coordinates schedule and arranges feature teams that each contain approximately 1 program manager, 3–8 developers, and 3–8 testers (who work in parallel 1:1 with developers).

Development phase

Development Phase Feature development in 3 or 4 sequential subprojects that each results in a milestone release

Program managers coordinate evolution of specification. Developers design, code, and debug. Testers pair with developers for continuous testing.

- **Subproject I** First 1/3 of features (Most critical features and shared components)
- **Subproject II** Second 1/3 of features
- **Subproject III** Final 1/3 of features (Least critical features)

Stabilization phase

Stabilization Phase Comprehensive internal and external testing, final product stabilization, and ship

Program managers coordinate OEMs and ISVs and monitor customer feedback. Developers perform final debugging and code stabilization. Testers recreate and isolate errors.

- **Internal Testing** Thorough testing of complete product within the company

- **External Testing** Thotough testing of complete product outside the company by "beta" sites, such as OEMs, ISVs, and end users

- **Release preparation** Prepare final release of "golden master" disks and documentation for manufacturing

Strategie e Principi

1. Strategia per definire prodotto e processo: "*considerare la creatività come elemento essenziale*"

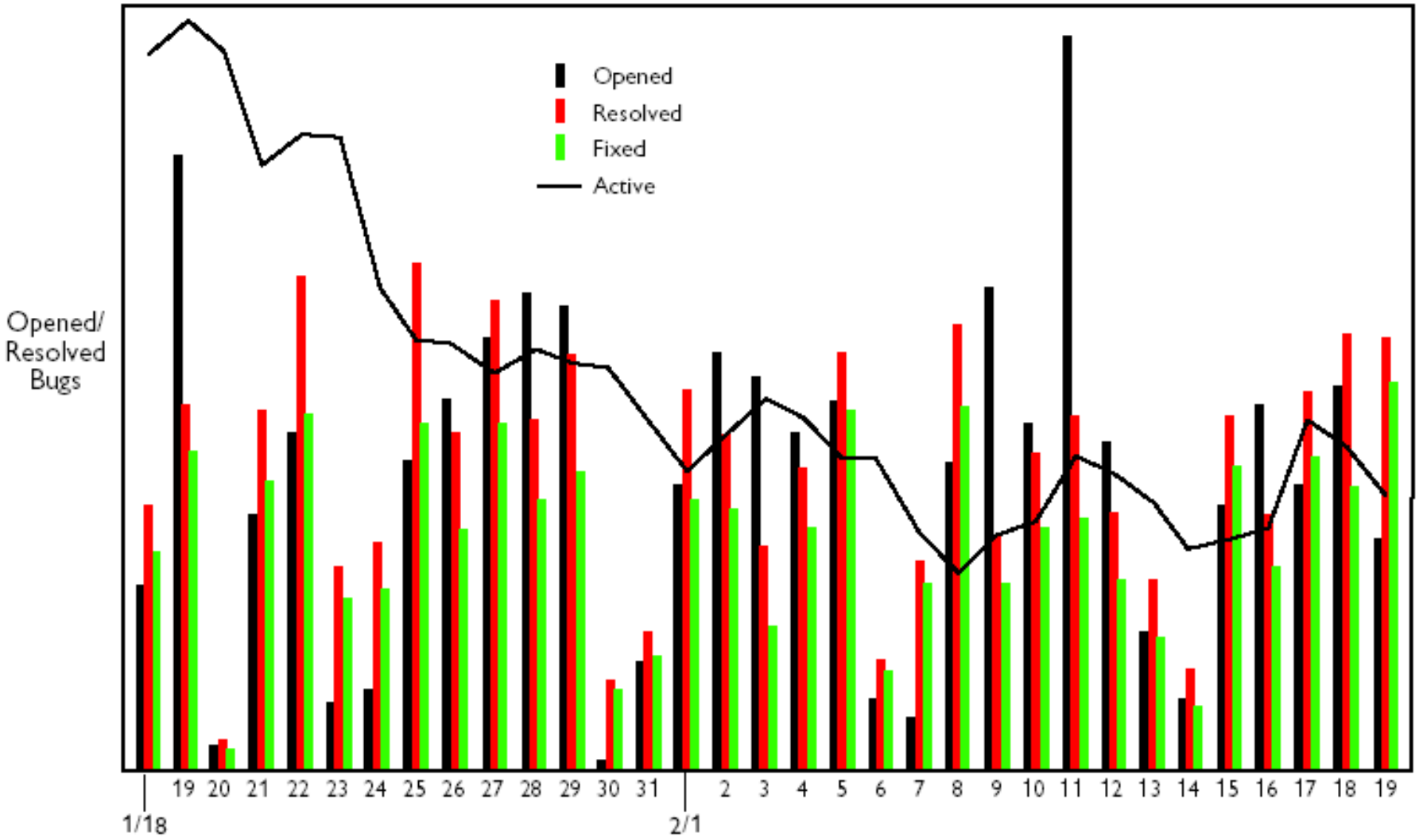
   Principi di realizzazione:

   a. Dividere il progetto in milestone (da 3 a 4)
   b. Definire una "product vision" e produrre una specifica funzionale che evolverà durante il progetto
   c. Selezionare le funzionalità e le relative priorità in base alle necessità utente
   d. Definire un'architettura modulare per replicare nel progetto la struttura del prodotto
   e. Assegnare task elementari e limitare le risorse

Strategie e Principi (2)

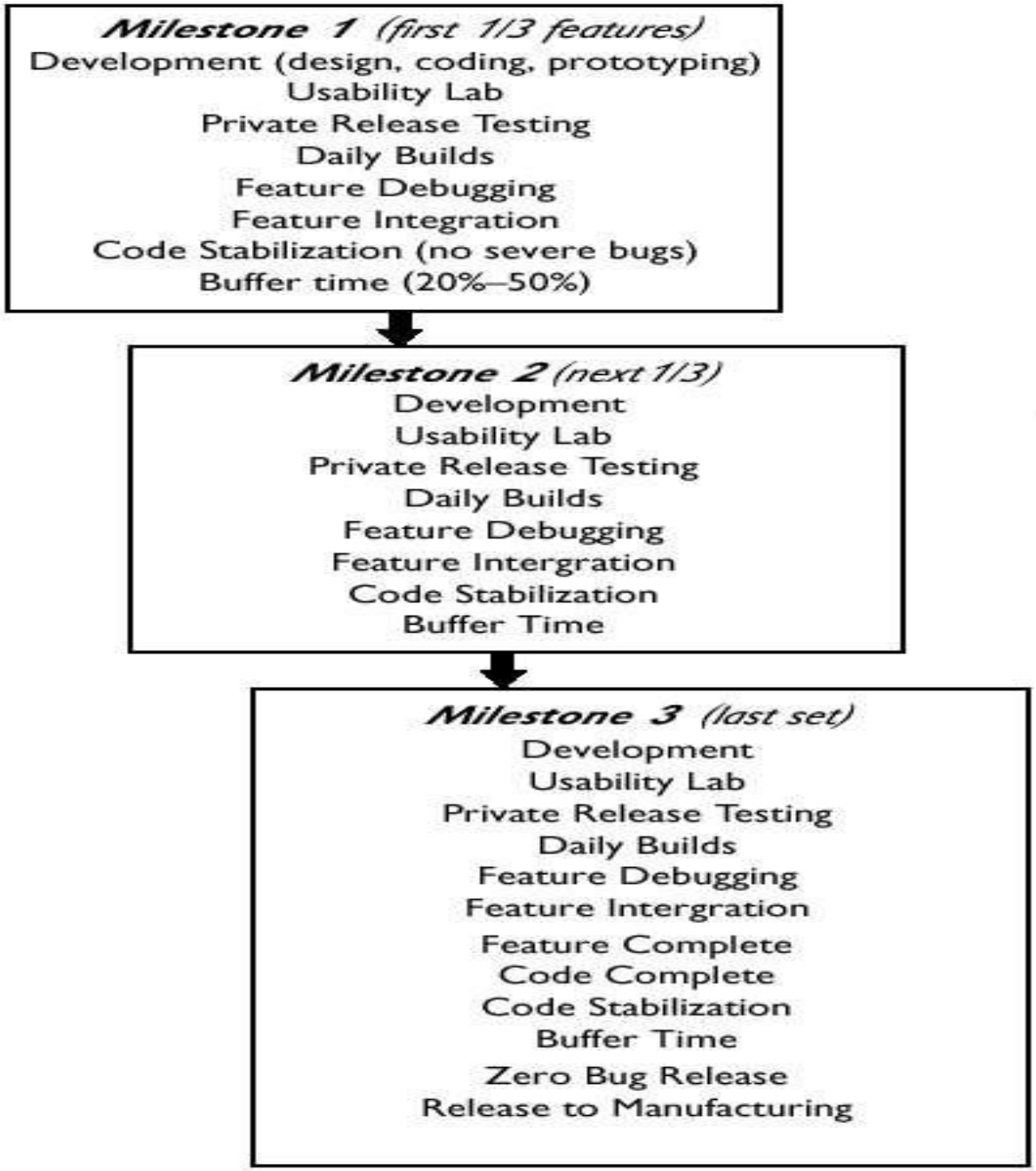
2. Strategia per lo sviluppo e la consegna dei prodotti: "*lavorare in parallelo con frequenti sincronizzazioni*"
   Principi di realizzazione:
   a. Definire team paralleli ed utilizzare *daily build* per la sincronizzazione
   b. Avere sempre un prodotto da consegnare, con versioni per ogni piattaforma e mercato
   c. Usare lo stesso linguaggio di programmazione all'interno dello stesso sito di sviluppo
   d. Testare continuamente il prodotto durante il suo sviluppo
   e. Usare metriche per il supporto alle decisioni

Esempio di metriche collezionate

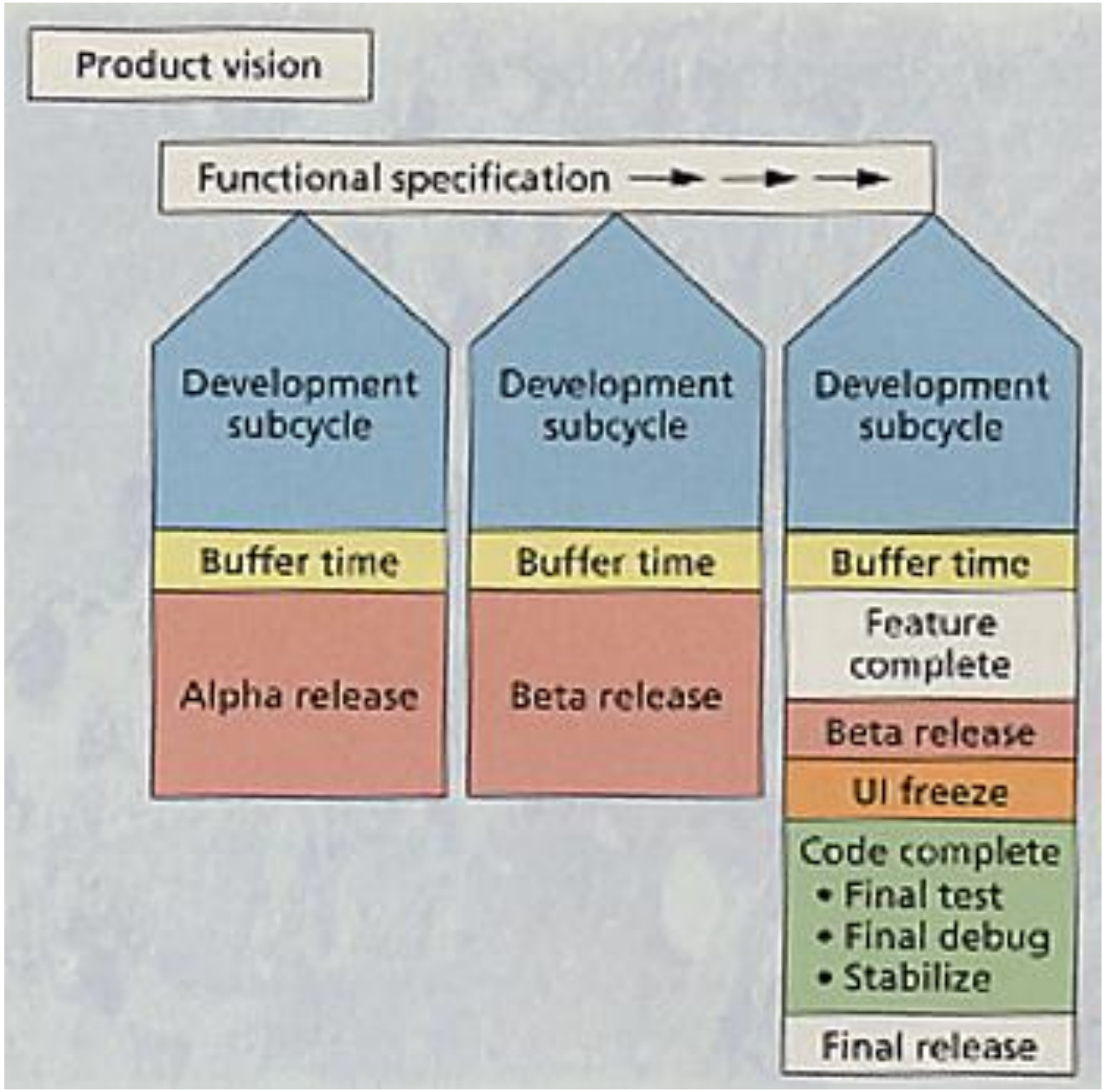


Source: Microsoft internal document

Milestones


Milestone 1 (first 1/3 features)
Development (design, coding, prototyping)
Usability Lab
Private Release Testing
Daily Builds
Feature Debugging
Feature Integration
Code Stabilization (no severe bugs)
Buffer time (20%–50%)
Milestone 2 (next 1/3)
Development
Usability Lab
Private Release Testing
Daily Builds
Feature Debugging
Feature Intergration
Code Stabilization
Buffer Time
Milestone 3 (last set)
Development
Usability Lab
Private Release Testing
Daily Builds
Feature Debugging
Feature Intergration
Feature Complete
Code Complete
Code Stabilization
Buffer Time
Zero Bug Release
Release to Manufacturing

Modello del ciclo di sviluppo *synch-and-stabilize*



Confronto tra modelli *synch-and-stabilize* e *waterfall*

Synch-and-Stablize	Sequential Development
Product development and testing done in parallel	Separate phases done in sequence
Vision statement and evolving specification	Complete "frozen" specification and detailed design before building the product
Features prioritized and built in 3 or 4 milestone subprojects	Trying to build all pieces of a product simultaneously
Frequent synchronizations (daily builds) and intermediate stabilizations (milestones)	One late and large integration and system test phase at the project's end
"Fixed" release and ship dates and multiple release cycles	Aiming for feature and product "perfection" in each project cycle
Customer feedback continuous in the development process	Feedback primarily after development as inputs for future projects
Product and process design so large teams work like small teams	Working primarily as a large group of individuals in a separate functional department

Il Modello Netscape[2]

- Anche alla Netscape si è adottato un modello di tipo *synchronize-and-stabilize*, con opportuni adattamenti allo sviluppo di applicazioni Internet (browser e prodotti server):
  - dimensione dello staff
    - in media 1 tester ogni 3 sviluppatori (ma stessa produttività di Microsoft nello sviluppo di prodotti comparabili, ad es. *IE* vs. *Communicator*)
  - processo
    - scarso effort di pianificazione (tranne che su prodotti server)
    - documentazione incompleta
    - scarso controllo sullo stato di avanzamento del progetto (lasciato all'esperienza e all'influenza dei project manager)
    - scarso controllo su attività di ispezione del codice (code review)
    - pochi dati storici per il supporto alle decisioni

[2] M.A. Cusumano and D.B. Yoffie, Software Development on Internet Time, *IEEE Computer*, October 1999.

Staffing

Table 2. Allocation of staff in Netscape's client and server development divisions, mid-1998.

	Client products	Server products	Total
Software engineering	110	200	310
Testing (QA)	50	80	130
Product management	50	42	92
Subtotal	210	322	533
Other*	30	98	128
Total	240	420	660

*Persons in activities such as documentation, user interface design, OEM porting, internationalization and localization, and special product support.

Netscape Development Process (1)

Step 1: Product requirements and project proposal

Advance planning meeting (APM) held to brainstorm ideas (marketing, development, executives)

Product vision generated, initially by senior engineers, now mainly by product managers

Some design and coding by engineers to explore alternative technologies or feature ideas

Product requirements document compiled by product managers, with help from developers

Informal review of this preliminary specification by engineers

Functional specification begun by engineers, sometimes with help from product managers

Schedule and budget plan compiled by marketing and engineering, and informally discussed with executives

Step 2: First executive review

Executives review product requirements document and schedule and budget proposal

Plan adjusted as necessary

Netscape Development Process (2)

Step 3: Start of development phase
Design and coding of features, architecture work as necessary
Daily integration of components as they are created and checked in (builds)
Bug lists generated and fixes initiated

Step 4: Interim executive review (if necessary)
Functional specification should be complete at this point
Midcourse corrections in specification or project resources, as necessary
Coordination issues with other products or projects discussed, as necessary
Development continues

Step 5: First internal (alpha) release (takes approximately six weeks)
Development stops temporarily
Intensive debugging and testing of existing code
Alpha release for internal feedback (or possibly a developer's release)
Development continues
User feedback incorporated
Feature-complete target (rarely met, though servers especially try to be as complete as possible)
One week to stabilize beta release

Netscape Development Process (3)

Step 6: Public beta 1 or field test 1 (takes approximately six weeks)
Repeat development and testing steps in Step 5
Server groups moving to "field tests" with limited customers rather than public betas

Step 7: Public beta 2 and 3 (each beta takes approximately six weeks)
Repeat development steps as in Step 5
UI freeze milestone
Feature-complete status "mandatory," although some minor changes still allowed

Step 8: Code complete
No more code added except to fix bugs; features are functionally complete

Step 9: Final testing and release
Final debugging and stabilization of release candidate
Certification meeting(s) with senior executives for ship decision
Release to manufacturing (RTM) and commercial release

Agile Methods

- In the early 2000's, a reaction was witnessed against the importance of carefully planned software processes, stating that such processes are too restrictive on the developers
- The term *agile method* was introduced to extend the original concept of iterative and incremental development to concepts such as intensive communication within the project, fast feedback, few external rules for the way of working, etc.
- The common values and principles of agile methods are summarized in the *Agile Manifesto*

Agile Manifesto (2001)

- The Agile Manifesto defines *agile values*

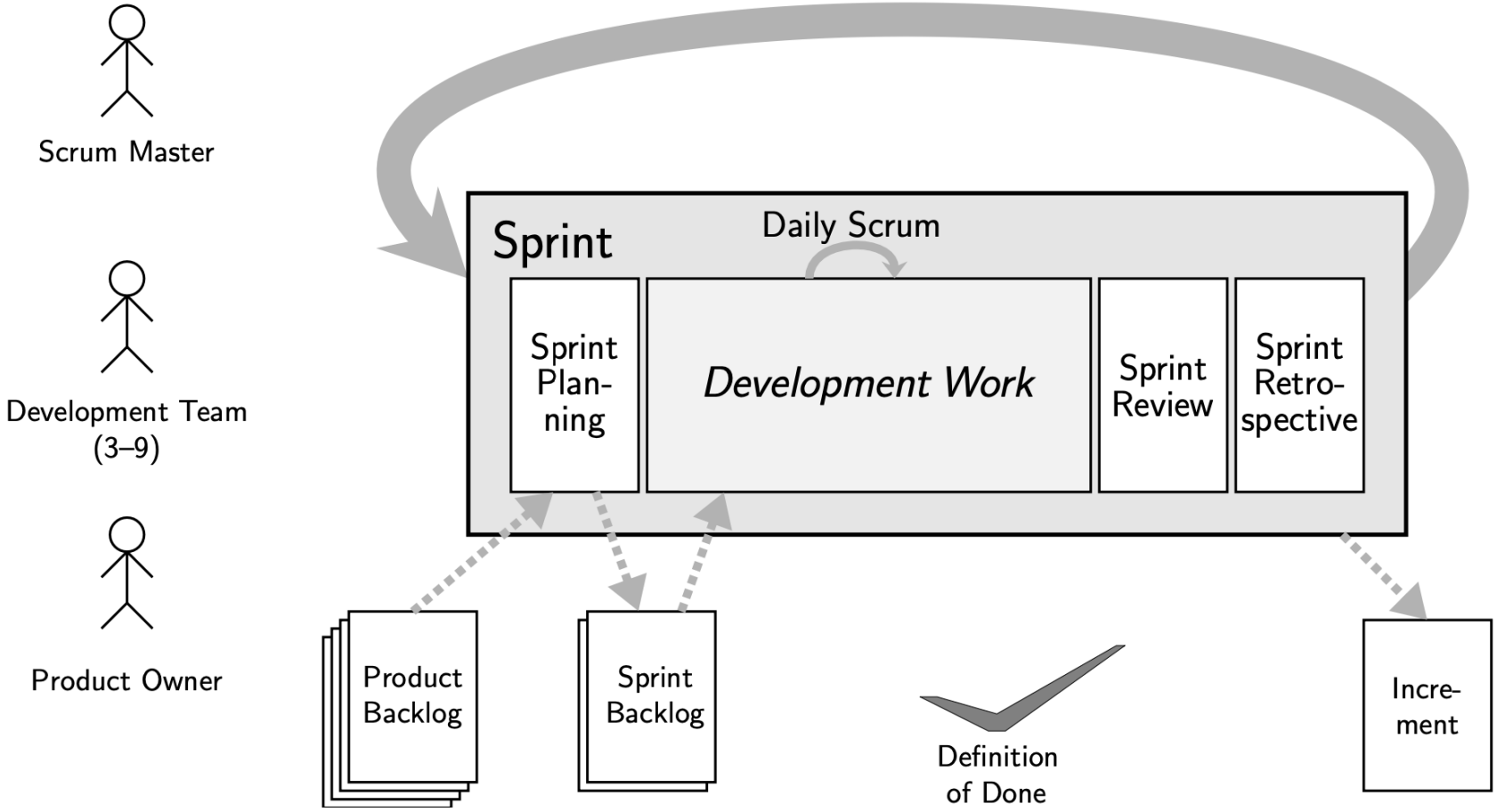
  "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

  – **Individuals and interactions** over *processes and tools*
  – **Working software** over *comprehensive documentation*
  – **Customer collaboration** over *contract negotiation*
  – **Responding to change** over *following a plan*

  That is, while there is value in the items on the right, we value the items on the left more."

- In addition to values, the Agile Manifesto contains the *twelve agile principles* that provide additional guidance on the use of agile methods
- Together, these values and principles define the basic concepts that are today known as *agile development*

Scrum


Scrum Master
Development Team
(3–9)
Product Owner
Sprint
Daily Scrum
Sprint Plan-ning
Development Work
Sprint Review
Sprint Retro-spective
Product Backlog
Sprint Backlog
Definition of Done
Incre-ment

Scrum Roles

- The *Scrum master*:
  - ensures that the methodology is understood and properly implemented by the development team and the product owner
  - supports the team by helping everyone else to interact with the team according to the Scrum rules
- The *product owner* manages and helps prioritizing the requirements to be implemented as documented in the product backlog
- The *development team* is responsible for developing the product, including all relevant tasks (e.g., designing, coding and testing)

Sprints

- A ***sprint*** is carried out to deliver a new increment of working software, and typically takes 2 to 4 weeks
- It is started with the *sprint planning* meeting where the Scrum team transfers the items to be developed from the product backlog and transfers them into the sprint backlog
- During the sprint, the development team works on the increment, with short *daily Scrum* meetings (aka stand-up meeting) of the development team to synchronize work and address any problems
- At the end of a sprint, the increment is presented to the product owner and other stakeholders in a *sprint review*
- Finally, the *sprint retrospective* meeting is performed to identify and plan any improvements for the next sprint

Definition of *Done*

- Scrum requires a “definition of done”, where the development team defines for itself what it means for a work item to be done (i.e., before it is allowed to be integrated into the main branch)
- Typical minimum requirements included in this “definition of done” include an adequate number of test cases, as well as checking the integration of the new code to ensure that it does not break the main development branch
- The definition of done also requires that the code has been documented adequately, where the team defines for itself what ”adequate” means

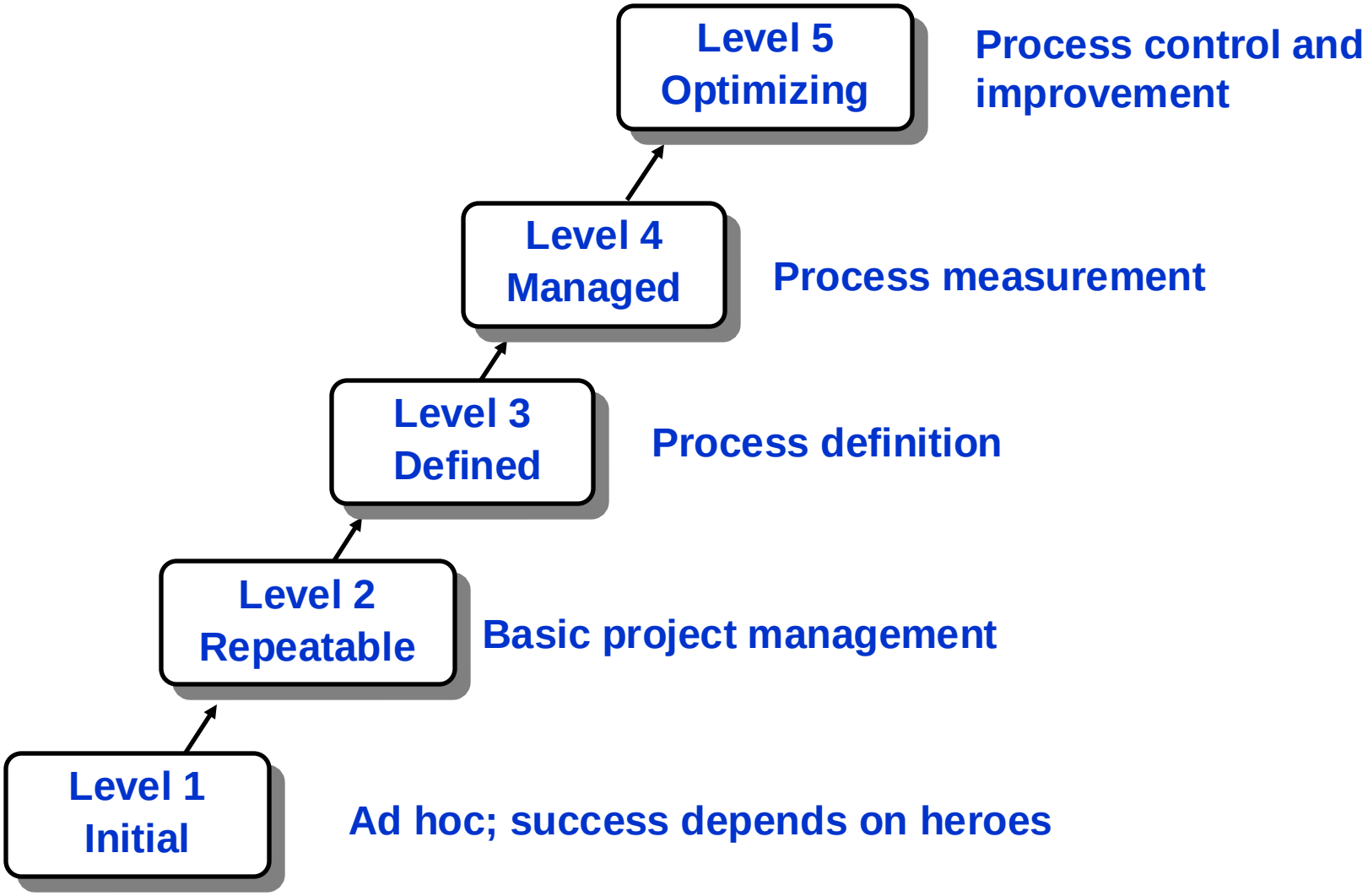
User Stories

- A common practice, used in agile development, often in combination with Scrum (but not defined in the Scrum Guide)
- A user story:
  - is a format for describing user requirements as a "story". Is should be short, typically just one sentence, and described from the user point of view
  - uses a common template, such as
    - *As a <role>, I want <goal> so that <benefit>*
      - Example: "*As a process engineer, I want to see the dependencies between different process steps so that I can easily verify and validate them*"
- Large user stories which are later broken down into smaller ones are often called *epics*

Capability Maturity Model (CMM)

- Il SEI (Software Engineering Institute) ha predisposto, a partire dal 1993, un modello per determinare il livello di maturità del processo software di un'organizzazione (ovvero una misura dell'efficacia globale dell'applicazione di tecniche di ingegneria del software)
- Il modello è basato su un questionario ed uno schema valutativo a cinque livelli
- Ogni livello comprende tutte le caratteristiche definite per il livello precedente

I 5 livelli del CMM

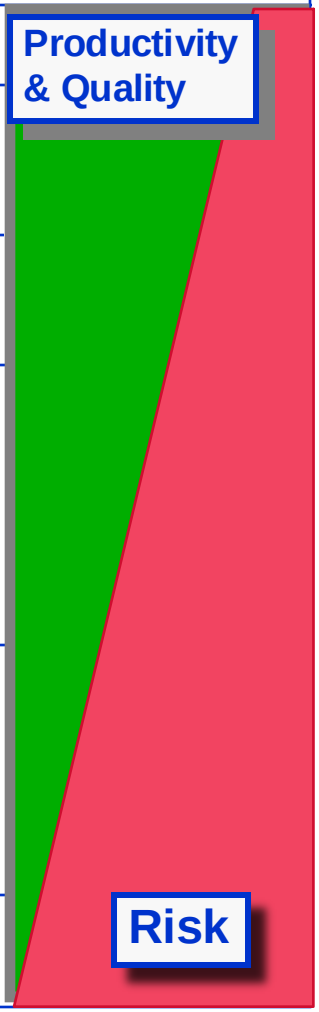


Key Process Areas

- Il CMM associa a ogni livello di maturità alcune KPA (Key Process Area), tra le 18 definite, che descrivono le funzioni che devono essere presenti per garantire l'appartenenza ad un certo livello.
- Ogni KPA è descritta rispetto a:
  - obiettivi
  - impegni e responsabilità da assumere
  - capacità e risorse necessarie per la realizzazione
  - attività da realizzare
  - metodi di "monitoring" della realizzazione
  - metodi di verifica della realizzazione

CMM KPAs

Level	Characteristic	Key Process Areas
Optimizing (5)	Continuous process capability improvement	Process change management Technology change management Defect prevention
Managed (4)	Product quality planning; tracking of measured software process	Software quality management Quantitative process management
Defined (3)	Software process defined and institutionalized to provide product quality control	Peer reviews Intergroup coordination Software product engineering Integrated software management Training program Organization process definition Organization process focus
Repeatable (2)	Management oversight and tracking project; stable planning and product baselines	Software configuration management Software quality assurance Software subcontract management Software project tracking & oversight Software project planning Requirements management
Initial (1)	Ad hoc (success depends on heroes)	"People"

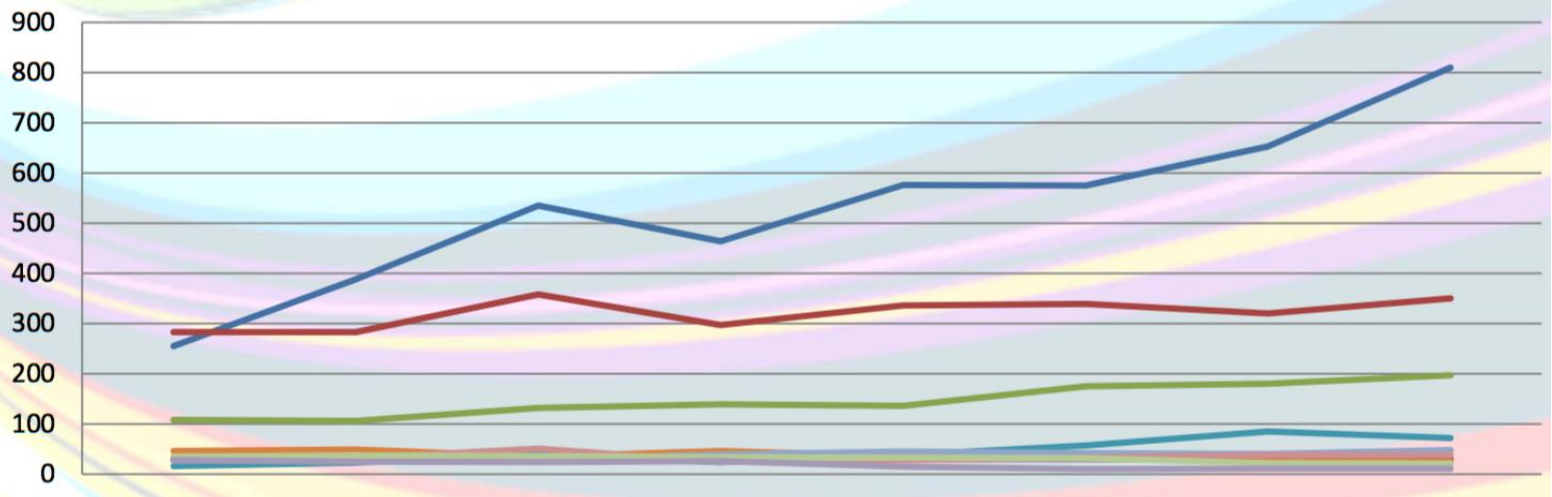


Statistiche a Febbraio 2000

La lista delle organizzazioni a livello 4 e 5 (*maturità elevata*) include:

- 71 organizzazioni negli USA
  - 44 organizzazioni a Livello 4 (tra cui Oracle, NCR, Siemens Info Systems, IBM Global Services)
  - 27 organizzazioni a Livello 5 (tra cui Motorola, Lockeed-Martin, Boeing, Honeywell)
- 25 organizzazioni al di fuori degli USA
  - 1 organizzazione a Livello 4 in Australia
  - 14 organizzazioni a Livello 4 in India
  - 10 organizzazioni a Livello 5 in India

Number of appraisals by country (06/15)



	2008	2009	2010	2011	2012	2013	2014	2015
China	255	388	535	464	576	575	653	810
United States	283	283	358	297	336	339	320	350
India	108	106	132	139	136	175	180	197
Spain	30	30	30	30	30	30	30	30
Mexico	16	23	39	24	38	57	85	72
Japan	46	49	35	46	36	32	26	25
Korea, Republic Of	27	36	29	39	45	42	40	48
Brazil	28	32	51	25	28	33	38	38
France	35	37	36	34	33	32	22	21
United Kingdom	28	25	24	26	15	10	11	11

Trends (as of June 2015)