# Development Models

Sandro Morasca
Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate
Via Ottorino Rossi 9 – Padiglione Rossi
I-21100 Varese, Italy
sandro.morasca@uninsubria.it

# Software Lifecycle Models

cosa?
conviene?
tempo?
come?

- In some cases, no reference model
  - code&fix

ho gia un idea del progetto se trovo un bug correggo

- When is it appropriate not to use a reference model?
  - NEVER
  - bad practice in any case

# Software Lifecycle Models: Waterfall

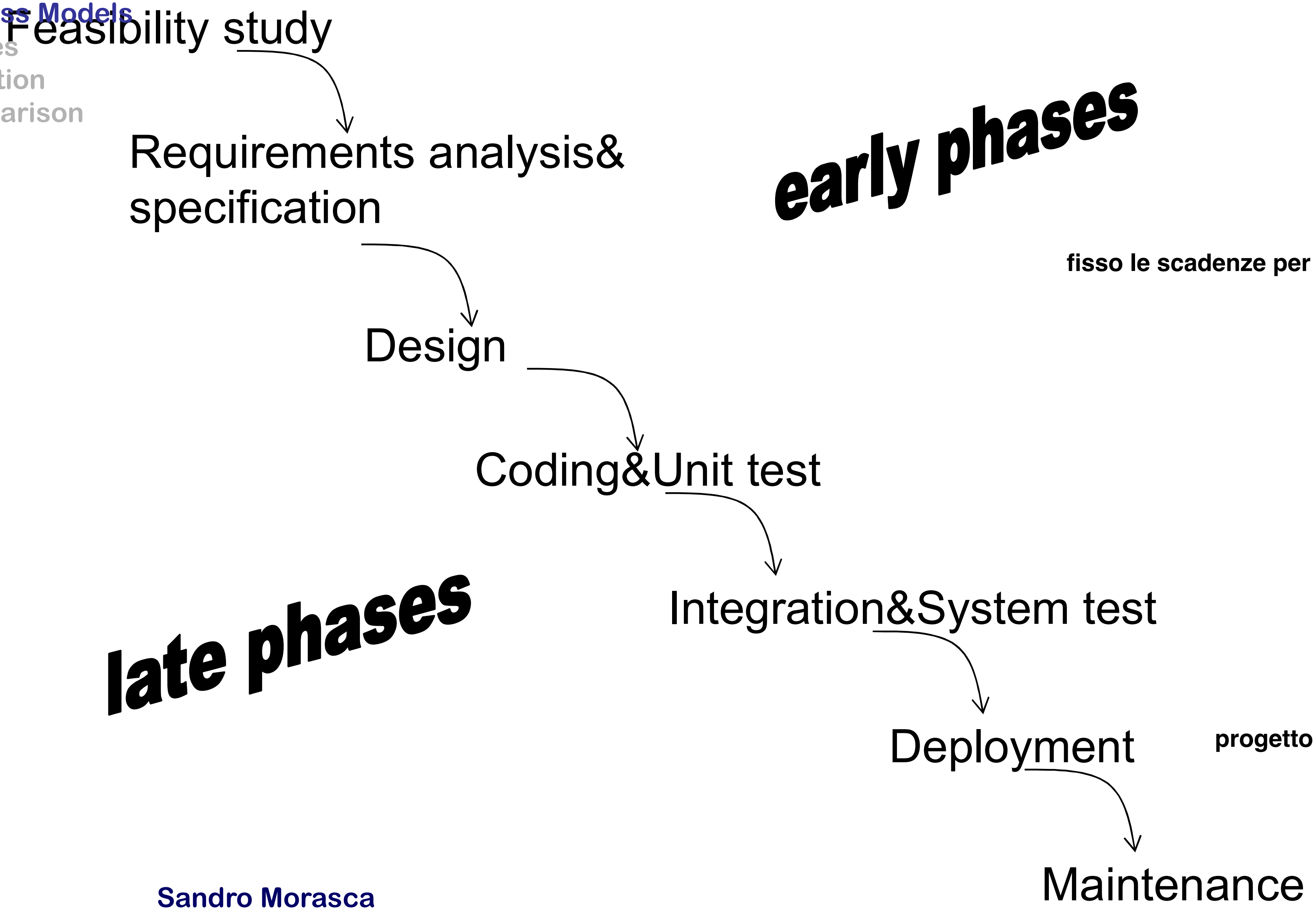
- The traditional “waterfall” model
  - identify phases and activities
  - force linear progression from a phase to the next
  - no returns (they are harmful)
    - better planning and control
  - standardize outputs (artifacts) from each phase

è un modello teorico , ha validita quando i requisiti sono congelati

- The waterfall model can be used when
  - requirements are certain and frozen
  - customers and users are not involved in the development process

cambia le documentazione ogni volta che cambio i requisiti

# Software Lifecycle Models
# Waterfall



Feasibility study

Requirements analysis& specification

Design

Coding&Unit test

Integration&System test

Deployment

Maintenance

early phases

late phases

fisso le scadenze per ogni passo

progetto finito

# Software Lifecycle Models: Agile Approaches

- **Process Models**
- Phases
- Evolution
- Comparison

vantaggi realeases piccole permettono di fare vedere al cliente un piccolo pezzettino eseguibile cosi al massimo modifco solo un pezzo

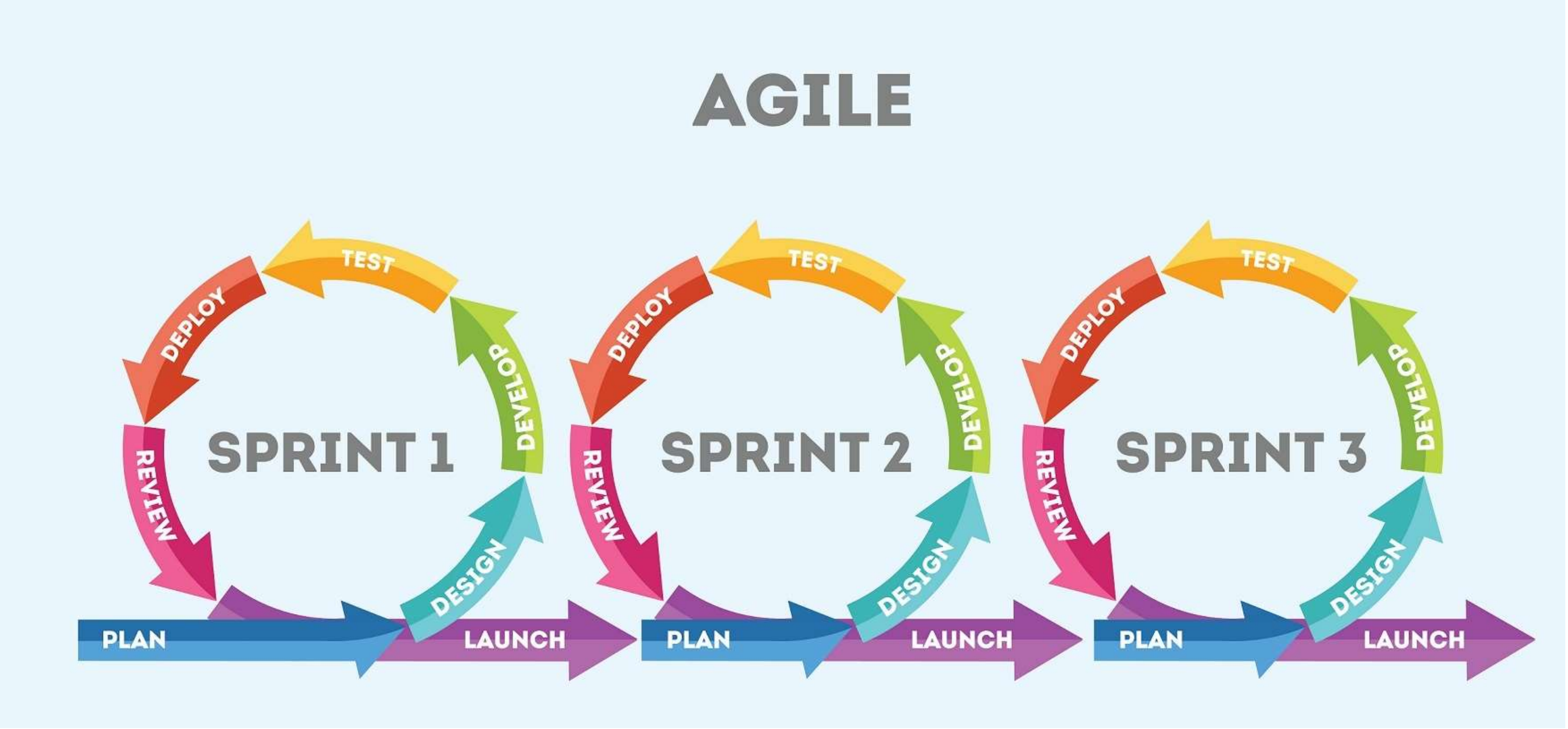
- The "Agile" model
  - Incremental (small releases,  rapid cycles)
  - Cooperative (communications between developers and customers)
  - Straightforward (method is easy to learn and modify, well documented) commento una documentazione piccola per ogni pezzo
  - Adaptive (embrace changes, even last moment)
- When we use Agile? scrivere software che sia modificabile cosi aggiorno facilmente
  - Requirements (uncertain or volatile)
  - Developers (responsible and motivated)
  - Customer (is involved and understands)

# Software Lifecycle Models: Agile Approaches

- ➢ **Process Models**
- Phases
- Evolution
- Comparison

**faccio cicli "incrementi" prodotto, faccio il piccolo progetto su un numero piccolo di funzioni o requisiti, cosi cambio solo un pezzo**

**tanti piccoli waterfall in un periodo ridotto**



# Feasibility Study

- Cost/benefits analysis   fattibilità e convenienza, mi conviene farlo? costi, tempo, ricavo
- Determines whether the project should be started (e.g., buy vs make), possible alternatives, needed resources
- Produces a Feasibility Study Document
  - Preliminary problem description   raccolta requisiti chiari
  - Scenarios describing possible solutions   come potrei farlo?
  - Costs and schedule for the different alternatives
  - Development model (Agile, Waterfall, custom, …)

# Feasibility Study

- In practice, the feasibility study is subject to
  - time pressure
  - cost pressure: we are not even sure that the customer will accept our offer

**mese * numero persone che affido sopra lo stesso lavoro**

- Consequences
  - alternatives may not be investigated
  - risks are not assessed right

**distribuzione di carichi in modo efficiente se no esce o un software di merda o sforo la scadenza**

# Requirements Analysis and Specification

fase alta

Process Models
➢ Phases
Evolution
Comparison

Requires sono dei compiti che il software deve fare es “pressione sanguigna”, concordati con il cliente

analisi requisiti

- Analyze the domain in which the application takes place
- Identify requirements
- Derive specifications for the software
  - Requires an interaction with the user
  - Requires an understanding of the properties of the domain
- Produces a Requirements Analysis and Specification Document (RASD)

documento dei requisiti, devo scrivere tutto quello che deve essere fatto e deve fare il software

se per ospedale = la sanita è un dominio applicativo, sapere come funziona da fuori e da dentro

# The 5 Wh's

a cosa deve rispondere il documento

Process Models
➢ Phases
Evolution
Comparison

- Who per chi è prodotto, che conoscenze avrà?
    - who will use the system
- Why perche un utente dovrebbe usare la mia app invece che un altra?
    - why should it be developed + why will the users use it
- What (vs How) cosa farà ?
    - what will it provide
- Where
    - where will it be used, on which architecture
- When
    - when and how long will it be used

per quanto tempo dovrà essere usato ?, quando diventerà opzoleto? quando e se mai faro una versione 2.0? quando andrà in pensionamento?

# RASD

Process Models
➢ Phases
Evolution
Comparison

- Required properties che proprietà dovrà avere questo complemento?
  - Precise non posso dire sarà molto rapido, devo dare il numero in secondi ecc
  - Complete comunque cambierà il modo di lavorare
    deve contenere tutti i requisiti
  - Consistent non deve essere contadditorio, sopratutto se scritto da molte persone
  - Understandable
  - Modifiable suddividerlo in capitoli cosi cambio solo i paragrafi ecc in base se l utente vuole modificare qualcosa
- May include
  - Preliminary User Manual puo includere il user manual versione 1.0 cosi da far vedere se va bene come verrebbe usato
  - System Test Plan
    avere tanti casi di test per vedere errori o cosa succede se faccio una determinata cosa

# RASD

- Functional requirements **cosa farà, cosa migliorerà dare il confronto su altri software**
- Non-functional requirements **requisiti funzionali tempi esecuzioni ecc..**
- Requirements on the development and maintenance process

# Design

- Defines the software architecture
  - Components (modules) classi che eredita altro da altro
  - Relations among components relazioni statiche e dinamiche ( il metodo a chiama b)
  - Interactions among components
- Goal fare vedere gia qualche esecuzione, cosa fa se faccio determinata cosa ? cosi nel caso correggo
  - Support concurrent development, separate responsibilities
- Produces the Design Document

# Coding&Unit Test

- Each module is implemented using the chosen programming language
- Each module is tested in isolation by the module’s developer
- Programs include their documentation

# Integration&System Test

- Modules are integrated into (sub)systems and integrated (sub)systems are tested
- This phase and the previous may be integrated in an incremental implementation scheme
- Complete system test needed to verify overall properties
- Sometimes we have *alpha test* and *beta test*

# Effort Distribution

- From 125 projects within HP **costo $**
  - 18% requirements and specification
  - 19% design
  - 34% coding
  - 29% testing
- typical variations around 10%

**17% finiscono in tempo e con il costo calcolato**
**molti non finiscono per mancanza fondi**
**molti non finscono in tempo**

# Deployment

- The goal is to distribute the application and manage the different installations and configurations at the clients' sites

# Maintenance

- All changes that follow delivery
- Unfortunate term: software does not wear out
  - if a failure occurs, the cause was there
- Often more than 50% of total costs
  - Recent survey among EU companies
    - 80% of IT budget spent on maintenance

# Maintenance

- It includes different types of change: correction + evolution
  - corrective maintenance ≈ 20%
  - adaptive maintenance ≈ 20%
  - perfective maintenance ≈ 50%

**manutenzione correttiva, a specifiche ferme sistemo bug**

**manutenzione adattattiva, funziona ma cambia il contesto (le leggi su tasse ecc)**

**manutenzione perfettiva, aggiungo funzionalita**

# Other Activities

- Some activities are carried out along the entire lifecycle
- Documentation
- Verification
- Management

# Folk Data on Errors

ispezioni del codice leggendolo

- Systematic inspection techniques can discover up to 50-75% of errors
- Modules with complex control flow are likely to contain more errors
- Often tests cover only about 50% of code
- Delivered code contains 10% of the errors found in testing
- Early errors are discovered late, and the cost of removal increases with time
- Eliminating errors from large and mature systems costs more (4-10 times) than in the case of small and new systems
- Error removal causes introduction of new errors
- Large systems tend to stabilize to a certain defect level

# Why Evolution?

- Context changes (adaptive maintenance)
  - EURO vs national currencies
- Requirements change
  - New demands caused by introduction of the system
  - Survey among EU companies indicates that 20% of user requirements are obsolete after 1 year
- Wrong specifications (requirements were not captured correctly or domain poorly understood)
- Requirements not known in advance

# How to Face Evolution

- Likely changes must be anticipated
- Software must be designed to accommodate future changes reliably and cheaply

*This is one of the main goals of*

*software engineering*

# Correction vs. Evolution

- Distinction can be unclear, because specifications are often incomplete and ambiguous
- This causes problems because specs are often part of a contract between developer and customer
  - early frozen specs can be problematic, because they are more likely to be wrong

# Software Changes

- Good engineering practice
  - first modify design, then change implementation
  - apply changes consistently in all documents
- Software is very easy to change
  - often, under emergency, changes are applied directly to code
  - inconsistent state of project documents
- *Software maintenance is (almost) never anticipated and planned*
  - *this causes disasters*

# Lifecycles

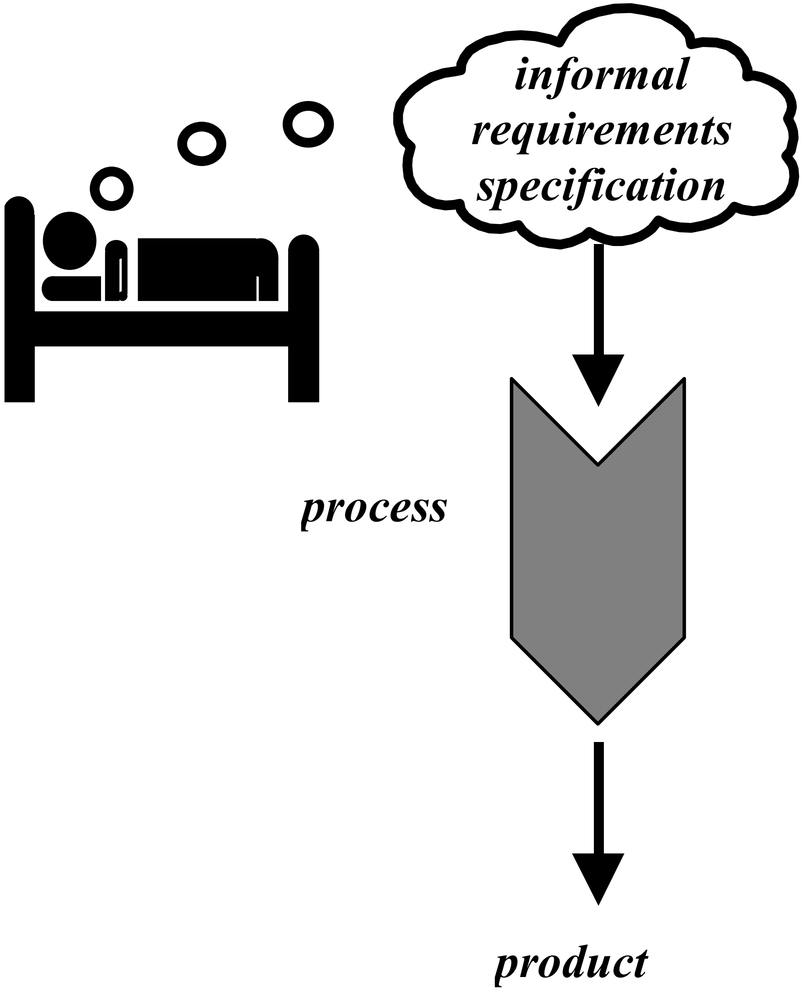
- Many variations exist
- Each organization tends to define “its own”
- Sample cases
  - software developed for personal use
  - customer (user) belongs to same organization
  - custom software developed by sw house
  - application for the market

# Waterfall Is “Black Box” until the end

- “Waterfall” requires that the domain be understood and requirements be known and stable
- This happens in only a few cases, e.g. airplane monitoring software
- Recycling cannot be eliminated
  - it is part of our problem

# Waterfall Is “Black Box” until the end

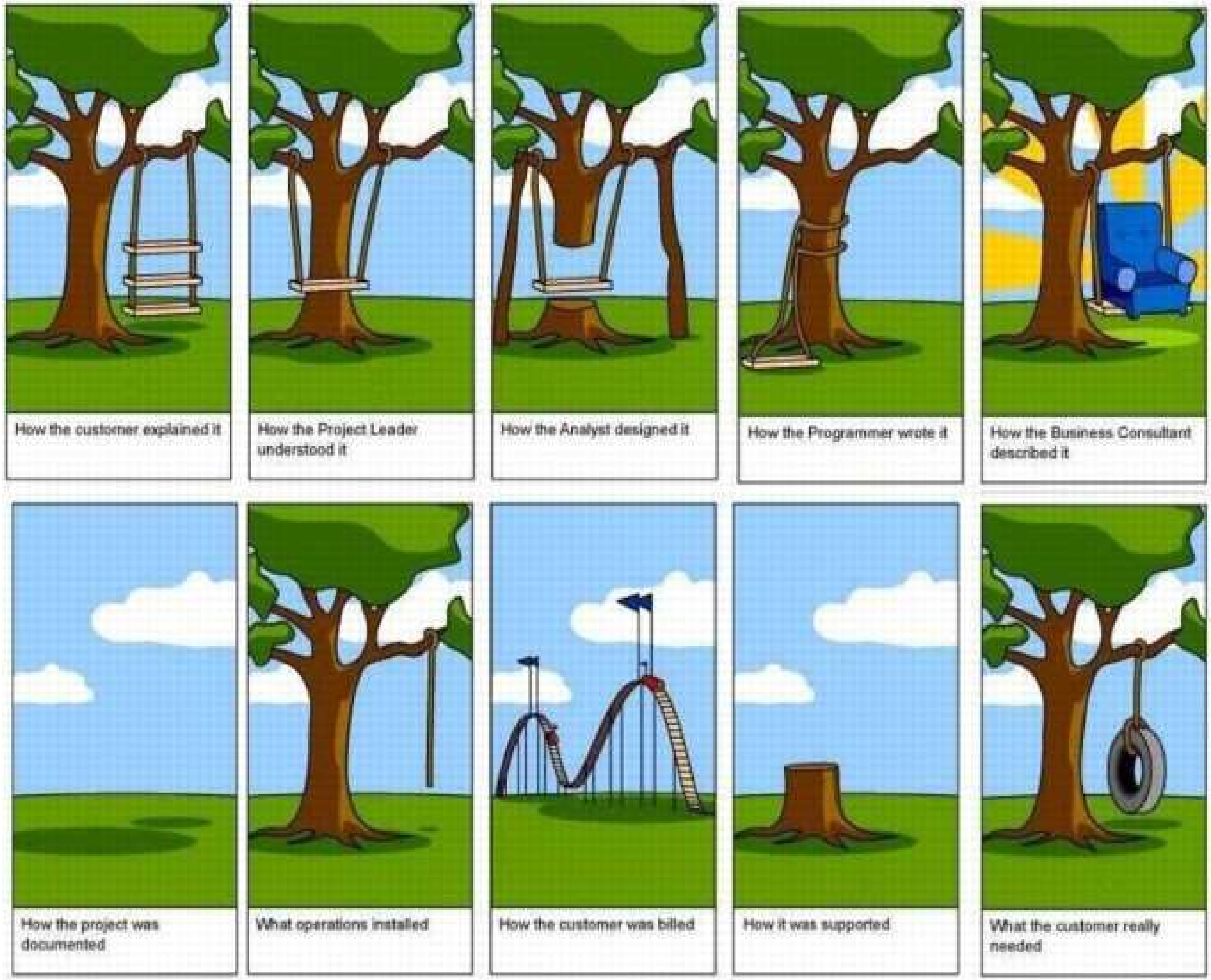
Process Models
Phases
Evolution
➢ Comparison



# Waterfall Is “Black Box” until the end

Process Models
Phases
Evolution
➢ Comparison

**avere feedback continuo dal cliente per evitare questo: nessuno ha capito un cazzo**

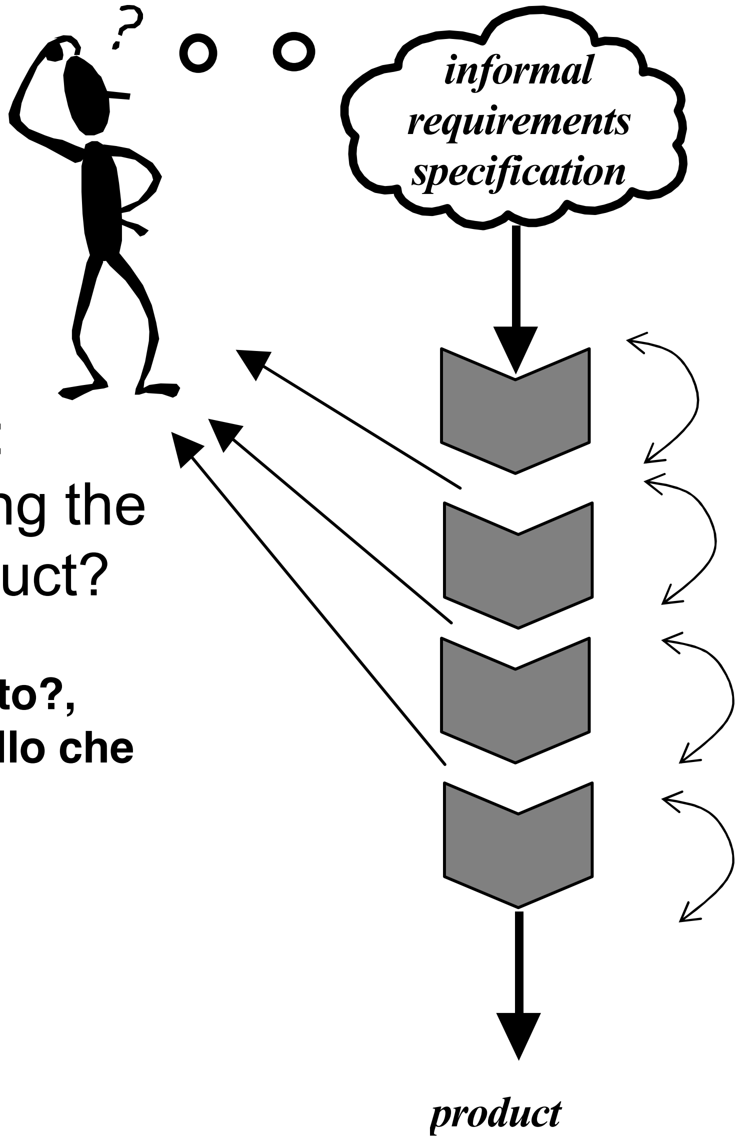

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

# Agile is for Transparency, Verification and Validation

Process Models
Phases
Evolution
➢ Comparison

- Transparency allows early check and change via feedback
- It supports flexibility
- It enables validation and verification

# Agile is for Transparency, Verification and Validation



**Validation**:
are we doing the “right” product?

sto facendo il lavoro giusto?,
secondo le specifichev (quello che
è stato richiesto)

sto construendo
in maniera giusta
il prodotto ?,
congruente con
le specifiche che
ho scritto?

**Verification**:
are we doing
the product right?

# Manifesto for Agile Software Development

https://agilemanifesto.org/iso/it/manifesto.html