Introduction to Software Engineering

Software lifecycle lecture

Philippe Lalanda

Philippe.lalanda@imag.fr

http://membres-liglab.imag.fr/lalanda/

Outline

- Introduction
- Waterfall model
- Evolutionary development
- Agile approaches
- Conclusion

Software activities (1)

- Recurrent activities
  - Requirements
  - Design
  - Implementation
  - Deployment
  - Operation and maintenance

- Activities produce various artifacts (documents, code, …) that must be evaluated

Software activities (2)

- Transverse activities
  - Project management
  - Quality management
  - Document management
  - Externalization management
  - Purchase management
  - …

- Not in the scope of this course

Linking all together

- Activities do not follow one another nicely. This is due to
  - unclear scope
  - Instability
    - inability to finish / validate activities
  - Dynamicity
    - new requirements
    - New conditions (market, available tools or COTS, …)
    - → need to backtrack

Software processes (1)

- A software process is a set of activities
  - An activity is a set of tasks, resources, constraints and ways to be performed
  - Activities are the same formost processes
- A software process defines how activities are linked together
  - Many processes have been defined; some are generic and some are specific to organizations

Software processes (2)

- Software processes are intellectual and creative processes
  - Rely on people
  - Attempts to automate processes have had limited success

- Also, big diversity of software processes
  - No ideal process
  - Must conform to available people, target domain …
  - Critical systems require very structured processes, business systems require flexible processes

Software process models

- A software process model is an abstract representation of a software process
  - Process frameworks that can be extended, tailored to specific SE processes

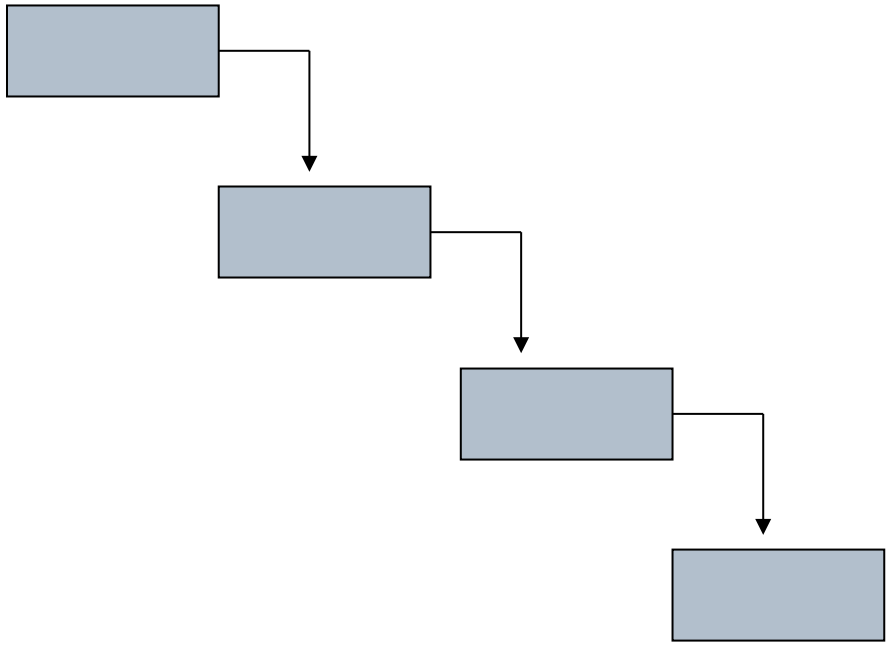
- Major process models
  - Waterfall model
  - Evolutionary development
  - Agile approaches

Outline

- Introduction
- Waterfall model
- Evolutionary development
- Agile approaches
- Conclusion

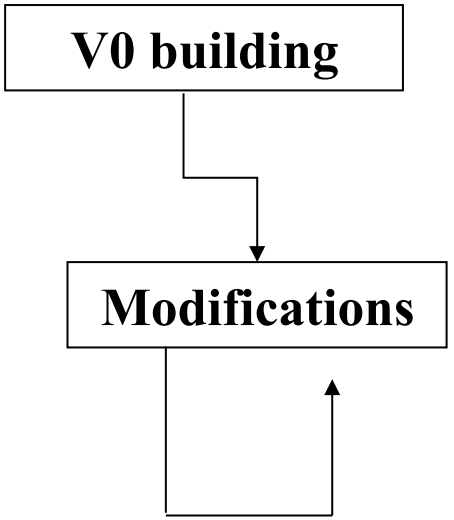
Waterfall model

- Development is seen as a succession of steps, strictly sequential
  - Each step is a base activity
  - Each step is validated
  - There is no (or little) backtrack

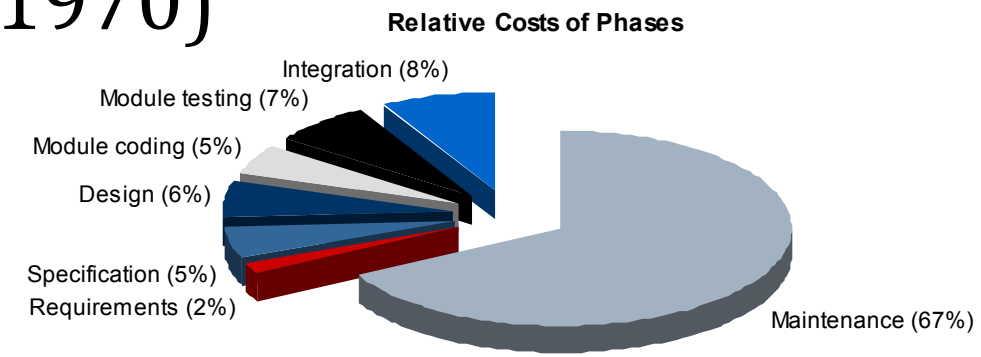


Code and fix

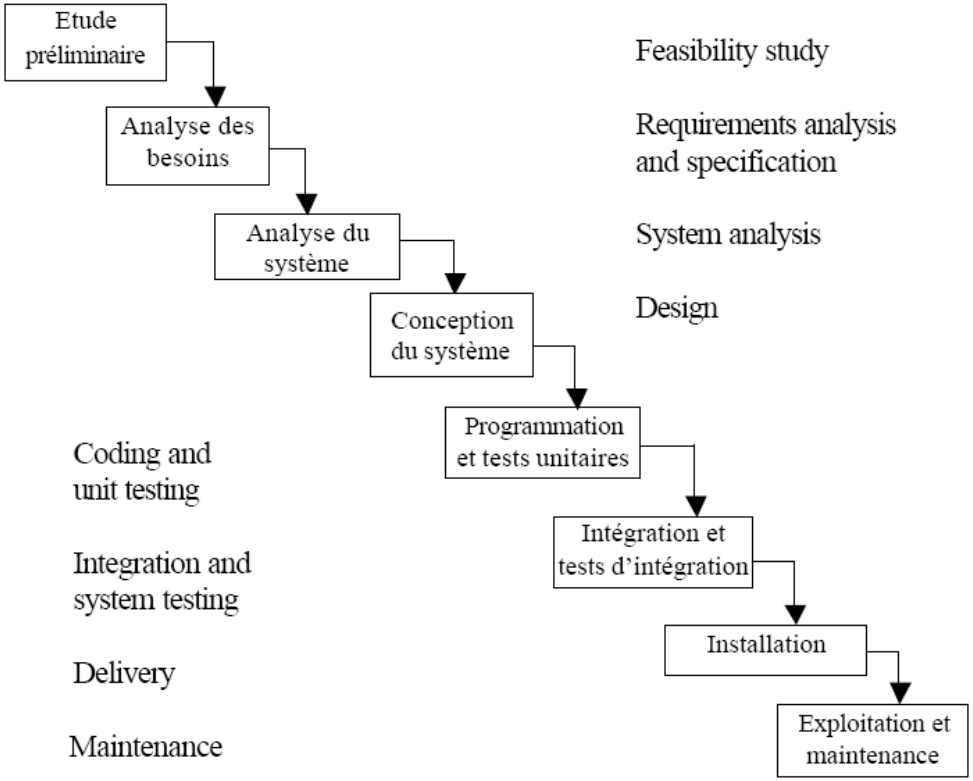
- We code first, then, we modify
  - Wild development
  - Little analysis, lot of coding
  - Your last student project ?



- Primitive model (< 1970)
  - Not adapted to large, multi team developments

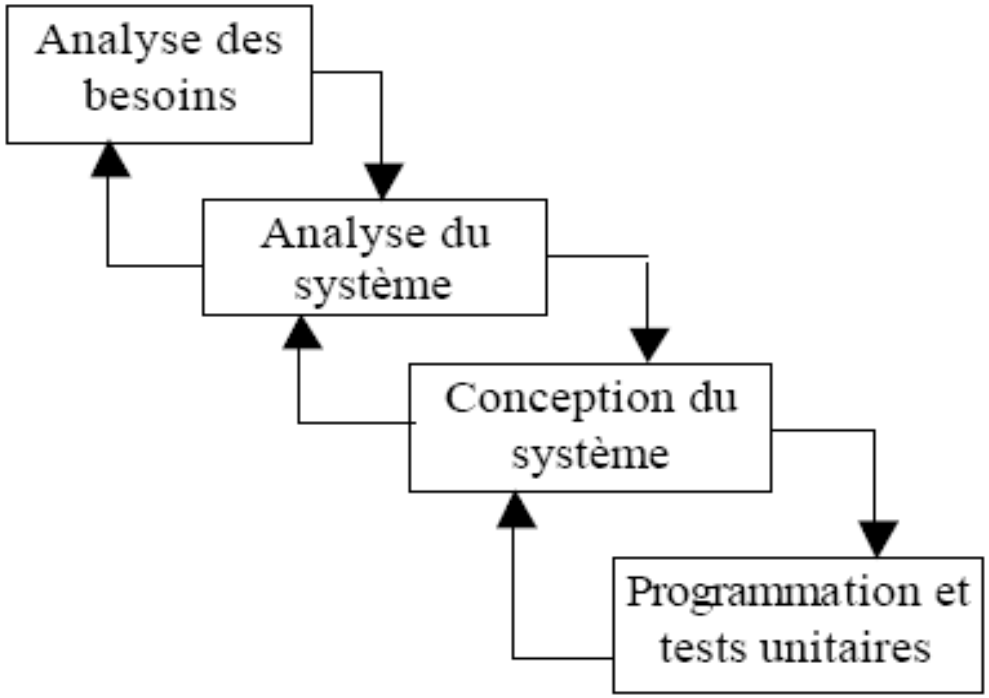


Waterfall model (1970)



- Large set of activities
- Document-oriented
- No backtrack
- Classic engineering

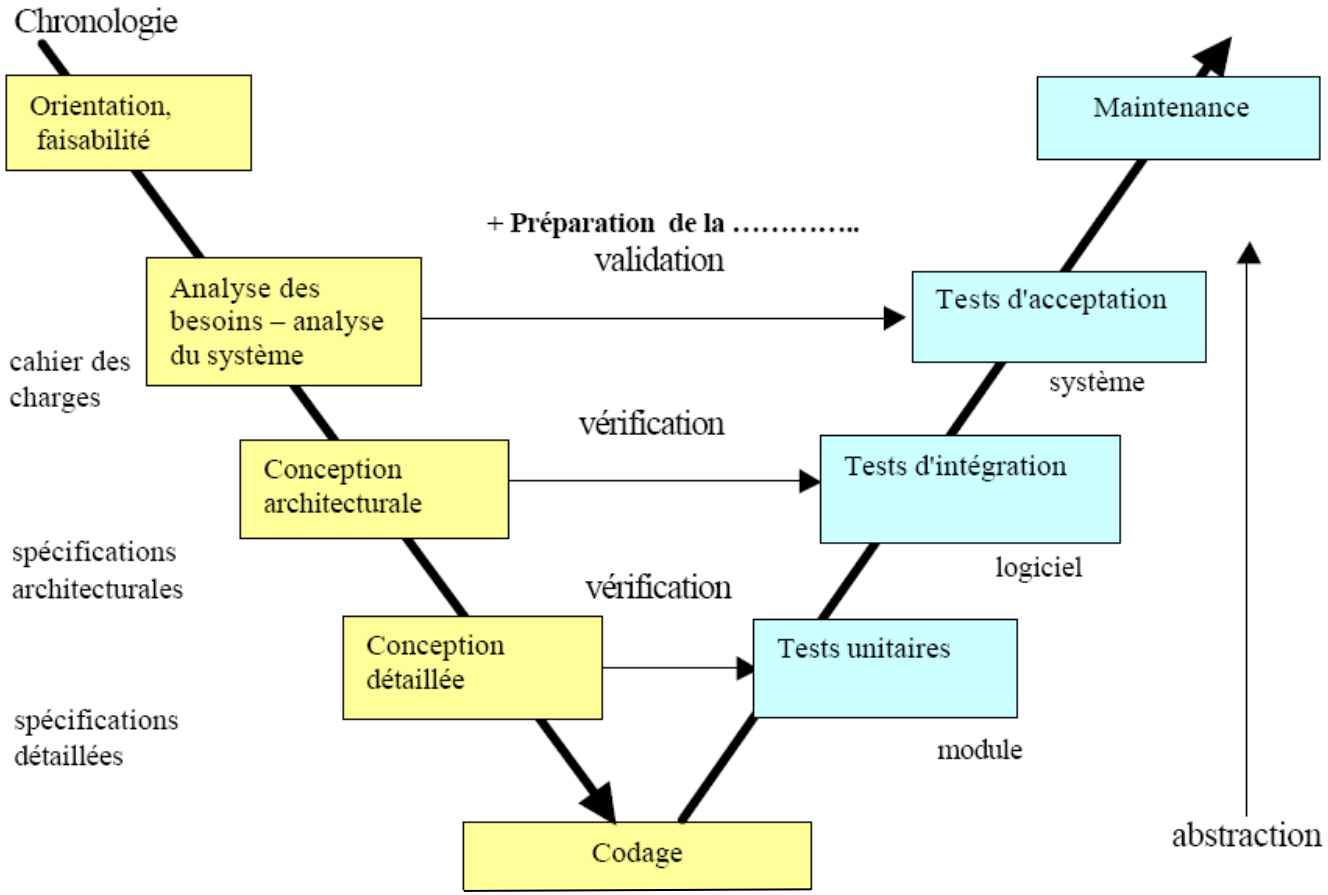
Waterfall model with iterations



- Possible backtrack
- More flexible
- Still hard to manage
- Limited number of iterations

V process

- Tests are defined at every phase



Advantages of waterfall models

- Simple, easy to grasp
- Enforce documentation
  - A step is not completed till document validation
- Test is included in every step (V)
- Progress is tangible
  - For the development team
  - For the managers, too !

Limits of waterfall models

- Document-driven
  - Meaningless to clients and users
  - Final product is the first artifact seen by users
- Makes the hypothesis the project is feasible
  - Solvable problem
  - Many problems are discovered at the end
- Not flexible
  - Stable needs, does not deal with evolutions
- Not realistic is most cases

Conclusion about waterfall models

- Applicability
  - Known and stable needs
  - Solid technologies
- Still very popular
  - Simple and similar to other engineering domains
  - Often used by non specialists or inexperienced software engineers

Outline

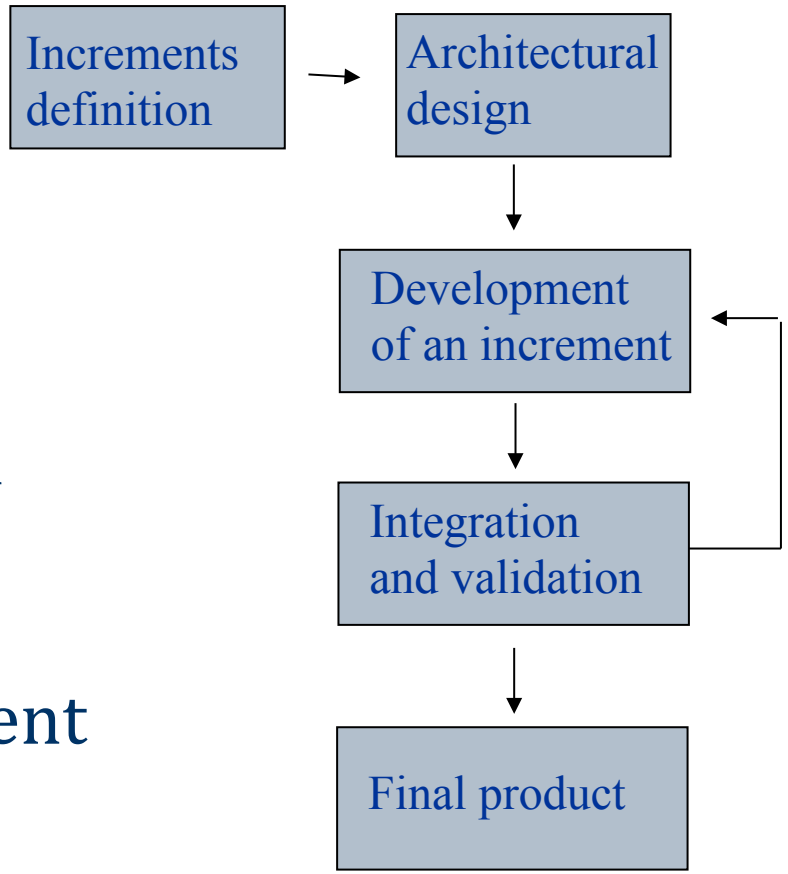
- Introduction
- Waterfall model
- Evolutionary development
- Agile approaches
- Conclusion

Changes

- Changes cannot be avoided
  - Technical and market environments change
  - Clients needs and wishes change
  - Management priorities change
    - Waterfall models are ineffective
- We cannot wait for stability to start
  - Time-to-market tyranny
  - The illusion of perfection

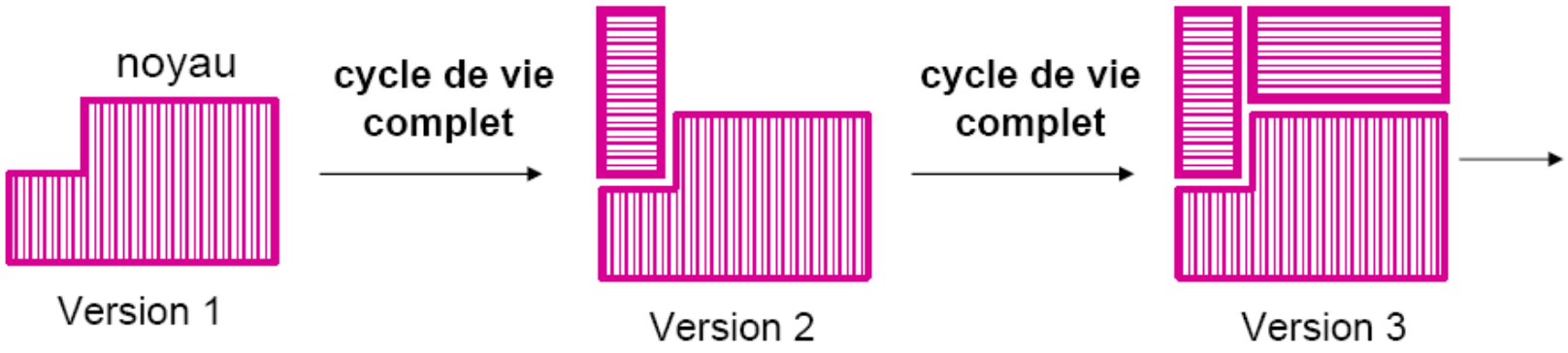
Incremental models

- Divide the projects into increments
  - An increment is a functional subpart of the final product
  - Each increment adds new functions
  - Each increment is tested
  - Increments are defined *a priori* – requirements prioritization (by the client if possible)



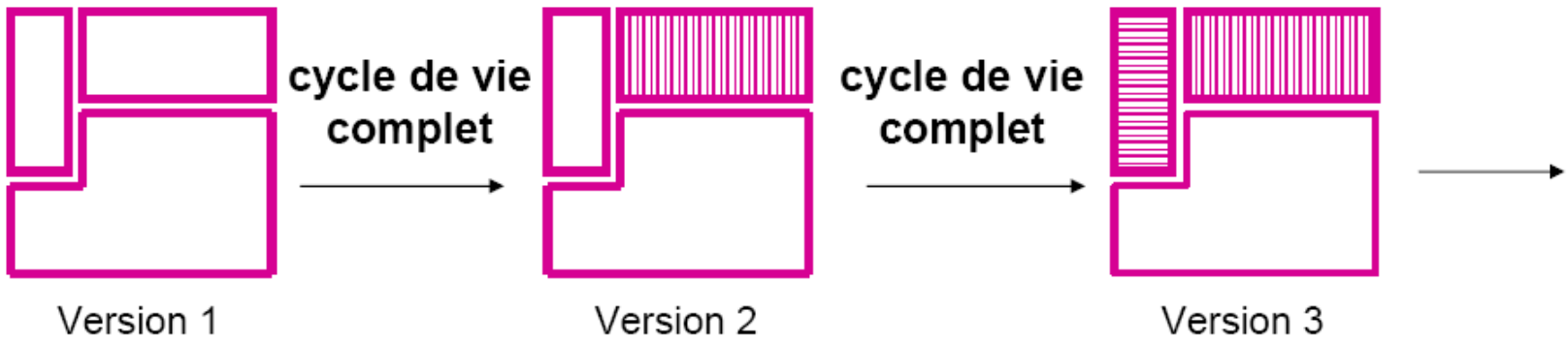
Incremental model - 1

- Evolving architecture
  - The first increment builds the core
  - Next increments build on top of the core
  - Each increment implements a complete lifecycle



Incremental model - 2

- Stable architecture
  - The first increment defines the architecture
  - Next increments provide part of the architecture
  - Parallel development is possible



Advantages of incremental models

- A first increment is rapidly delivered
  - Quick ROI
  - Management satisfaction (less stress !)

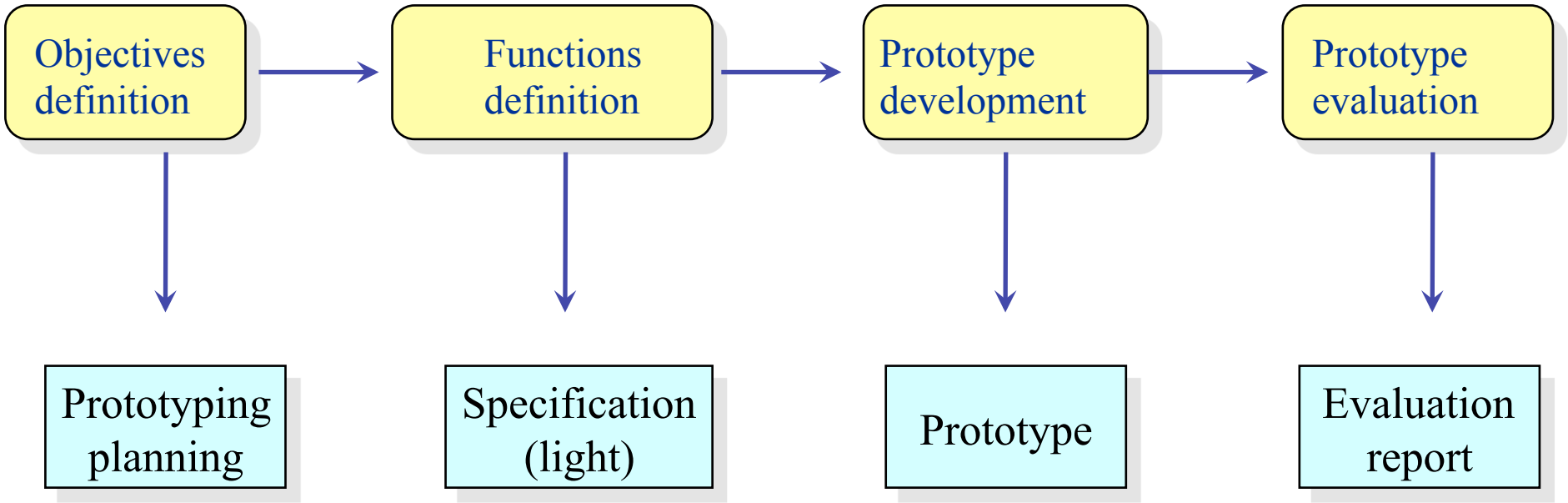
- Failure risks are decreased
  - Early detection of problems
  - Important parts are delivered at the beginning; they will be tested longer
  - Clients can add new requirements

Limits of incremental models

- Increments
  - Hard to define (the mapping requirements to increments is difficult)
  - Too many increments → unmanageable
  - Too few increments → waterfall
- Architecture
  - Hard to define stable architecture at the beginning
  - Hard to identify common services
- Does not deal with all possible evolutions (architectural evolutions)

“Special” increment : prototypes

- Building throwaway prototypes
  - A first increment to better understand the needs and challenges (technical, ergonomics, …)



Advantages of prototypes

- User involvement
- Requirements elicitation
- Technological evaluation
- Risks mitigation
- Usable with any software process
- Available supporting tools

Limits of prototypes

- Cost is badly accepted by managers
- Risk of badly focused prototype
- It is very tempting to build on top of prototype
  - Reusing non optimal design and code

Outline

- Introduction
- Waterfall model
- Evolutionary development
- Agile approaches
- Conclusion

Models presented so far

- A controlled approach to development
  - Precise planning
  - Quality insurance (at least management)
  - Analysis and design methods
  - Use of tools (CASE)
- Best applicability
  - Big projects (multi teams / sites / companies)
  - Long project (development and maintenance)

But

- Using such processes, we may end up spending more time on the process (how a project should be conducted) than on the software building itself !

Agile methods

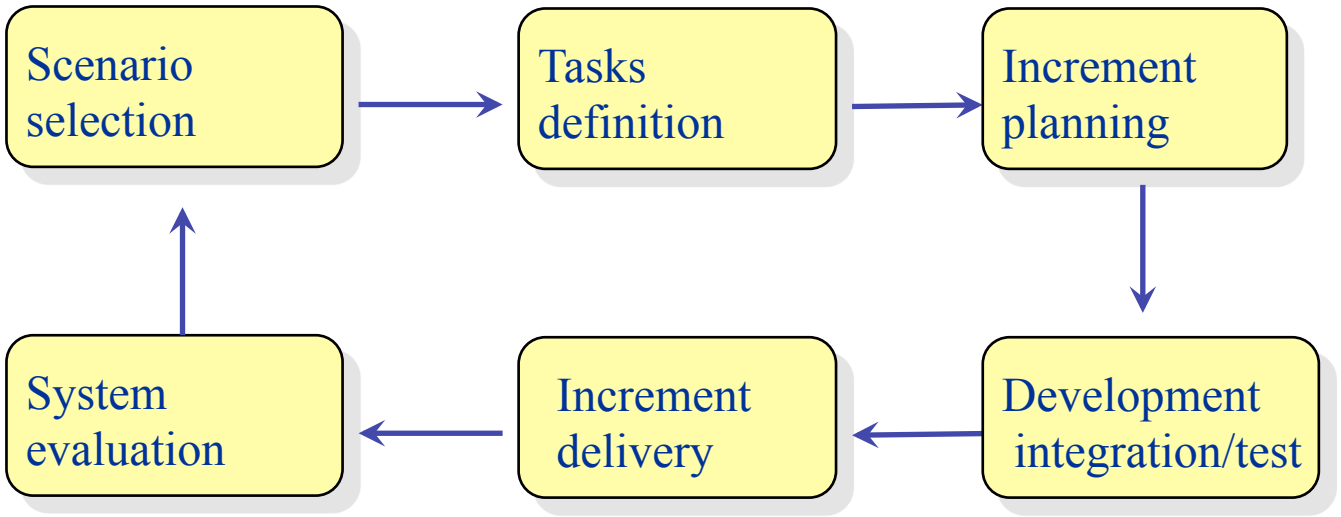
- Characteristics
  - Development focused
  - People focused
  - Iterative
  - Quick delivering of a first executable software
- Dedicated to “rapidly evolving” domains
  - Extreme programming » (Beck
  - Crystal » (Cockburn)
  - Adaptive software development » (Highsmith)
  - Feature driven development » (Palmer)

Principles

- User
  - involved in the development
  - provide requirements and priorities
  - evaluate iterations
- Increments
  - incremental delivery
- People
  - developers skill is recognized
  - no process
- Changes
  - not anticipated
- Simplicity
  - no unnecessary complexity

Extreme programming (XP)

- Frequent iterations
  - Scenario selection (cards)
  - Tasks definition and distribution
  - Planning and tests
  - Increment delivery and evaluation



XP principles

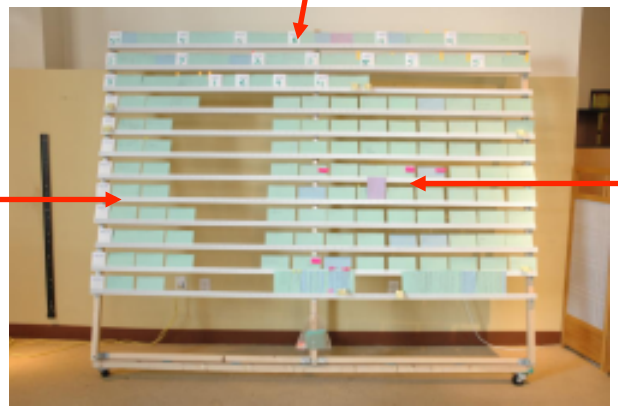
- Building an increment
  - Meeting every morning (standing)
  - Pair programming in a “war room”
  - War room preferably at the client site
  - Programmer defines and run tests
  - Minimal design
  - Code as simple as possible (constantly adapted)
  - Continuous integration
  - Intense rhythm

War room

Scenarios

Coded scenarios

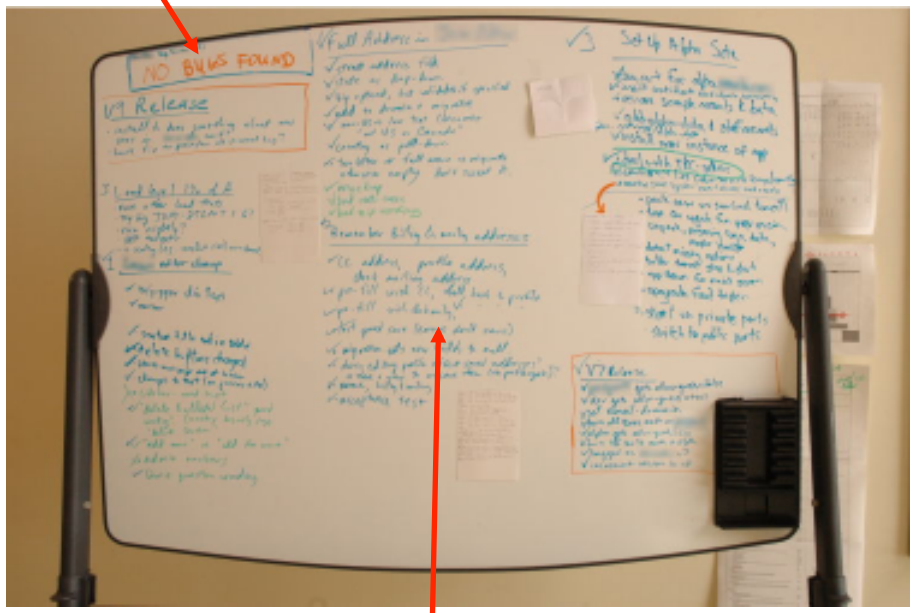
Planned scenarios

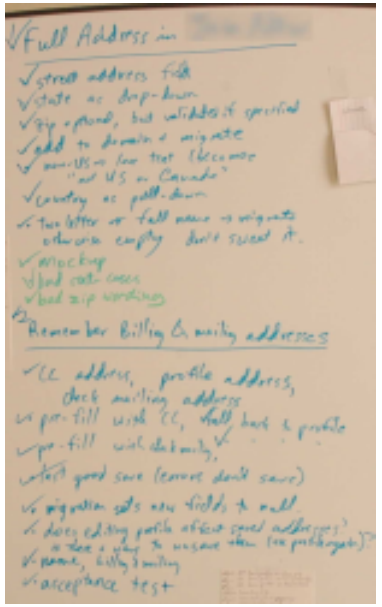


Not planned Scenarios

Ongoing scenarios (1 to 2 weeks)

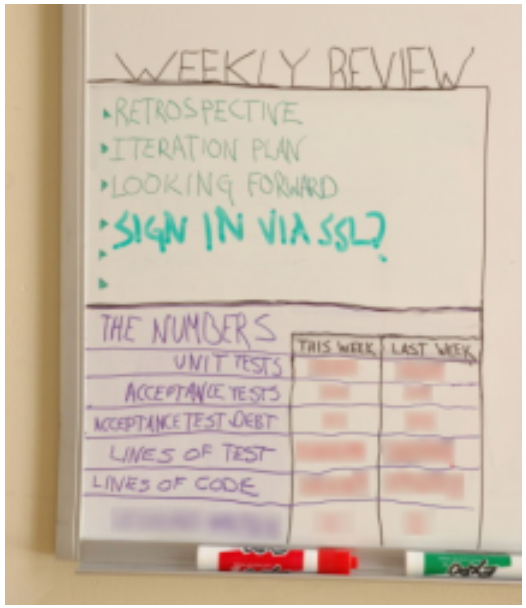
Bug list

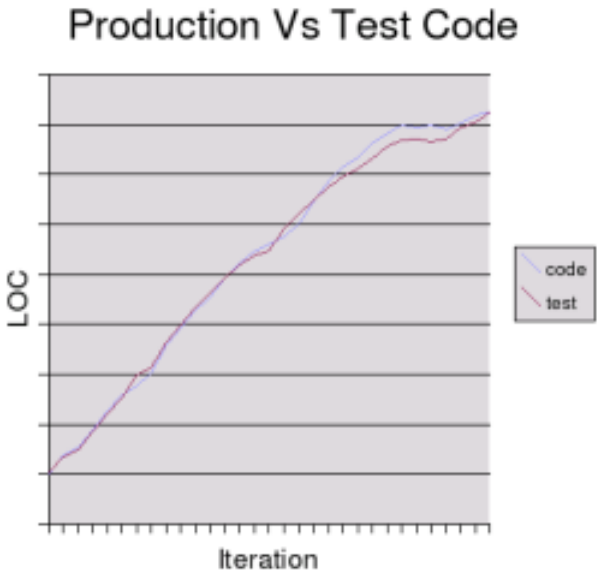




Detailed scenarion

End of iteration meeting





War rooms: more examples

Pair programming

- Egoless programming
  - Nobody owns the code
- Constant code review
  - Less expensive than formal reviews
- Favor code re-factoring
  - Towards simplicity
- As efficient as two independent programmers

Verification in XP

- Tests is not always managed in iterative approaches
  - No complete specification before the end
- Test management in XP
  - Tests are defined first
    - Every task is tested
    - Defined with clients
  - Automatic test generation

Issues in agile programming

- Client effective involvement
- Intense involvement of programmers
- Prioritization is hard (multi-clients)
- Keeping it simple requires additional work

Outline

- Introduction
- Waterfall model
- Evolutionary development
- Agile approaches
- Conclusion

Synthesis

- A process brings stability and control over an activity that can become chaotic
  - Better estimates
  - Better coordination
  - Better communication
  - Better productivity
  - Better visibility
- Using a process is a sign of maturity

To remember

- Managers love processes
  - What a pleasure to tell clients “phase x is over”
  - Some processes are required by clients
- Developers do not like them so much
  - There is gap with what happens in the trenches
  - Never deal completely with projects dynamicity
  - Phases always overlap (processes are theory !)

Which one is the best ?

- No ideal model
  - Waterfall: bad for innovative or changing projects
  - Incremental: costly for well defined, stable projects
- Small to medium projects (500 000 loc)
  - Incremental approaches are often appropriate
- Big projects
  - Mixed approaches using incremental and waterfall elements
    - Development of a prototype to identify requirements followed by a V cycle

Conclusion

Without a clear plan on how to do a project, there is little chance you get anywhere !