INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
LYON

Software engineering

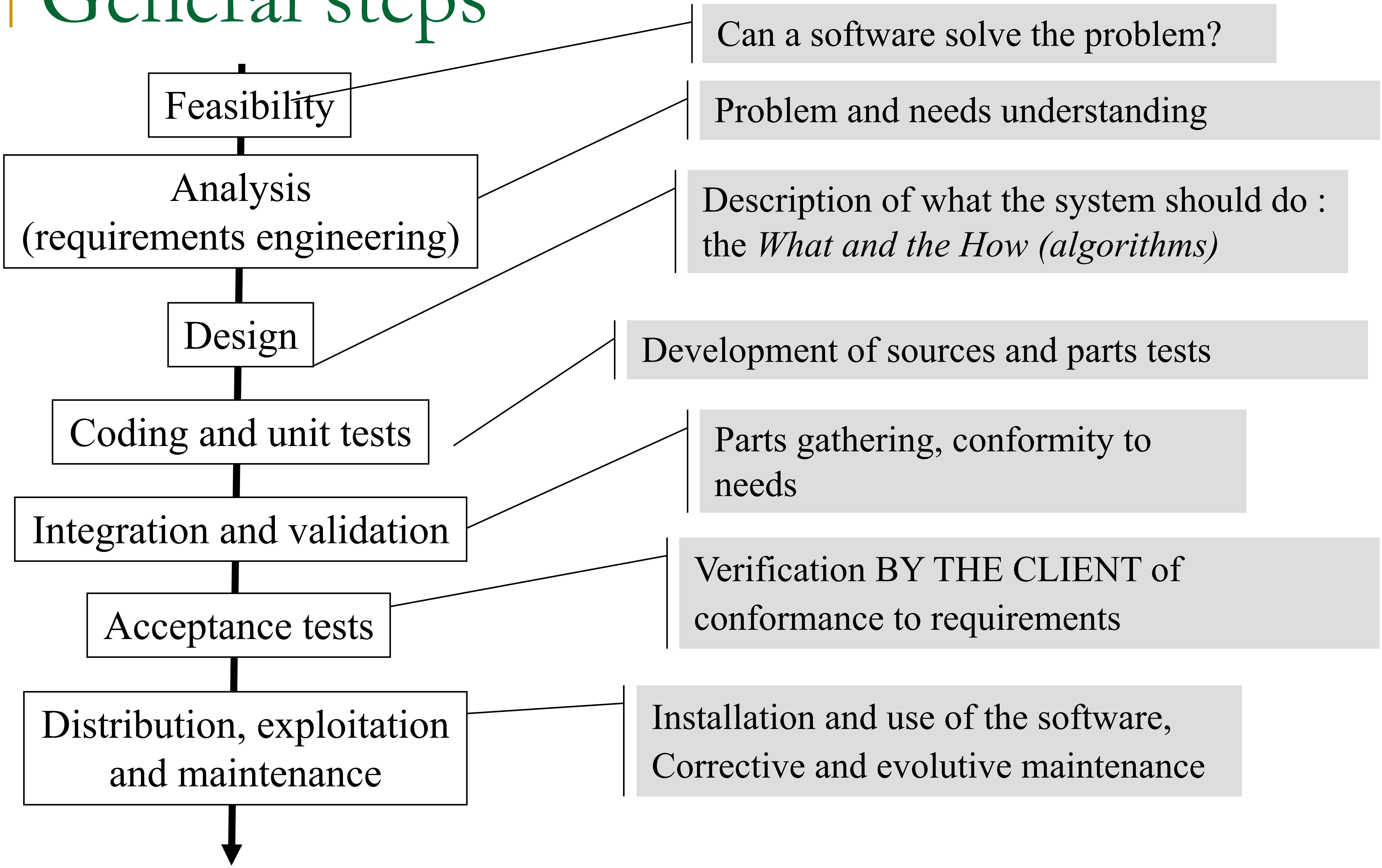
Software lifecycle

Frédérique Laforest

Software lifecycle

- Process for the development of software
  - Which steps and in which order?
  - By whom ?
  - To produce what ?
  - Who validates and how?
  - How does the software evolve in time?
- Lifecycle= structured set of activities accompanying the whole life of a quality software
  - From need request to end of use

General steps



Feasibility — Can a software solve the problem?

Analysis (requirements engineering) — Problem and needs understanding

Design — Description of what the system should do : the *What and the How (algorithms)*

Coding and unit tests — Development of sources and parts tests

Integration and validation — Parts gathering, conformity to needs

Acceptance tests — Verification BY THE CLIENT of conformance to requirements

Distribution, exploitation and maintenance — Installation and use of the software, Corrective and evolutive maintenance

Step1 : Feasibility (Etude préalable)

- 2 phases:
  - Exploration phase: should we build a software ?
    - organisational, technical and financial viability
  - Design phase: Request for proposal and project plan
    - What do stakeholders expect ?
  - General conditions
- Items built at this step
  - Interviews records
  - Decisions (to do, not to do, outsource, buy)
  - Request for proposal
  - General plan of the project
  - Budget
  - Definition of constraints

Example : Distant heating control

Example

Feasibility - Exploration phase

- Do communicating heating control modules exist? Do they follow a standard?
- Are they compatible with our heating system?
- How can we connect them with the Internet?
- How much do they cost?
- Estimation of hardware costs?
- Does it require subscriptions?
- From which type of terminal do I require to control the heating?
- Who will use this system? profile, capabilities/knowledge?
- How often will this service be used?
- Etc.

Example : Distant heating control

Example

Feasibility - Design phase

- Functional specifications

(It should be well written and organized)

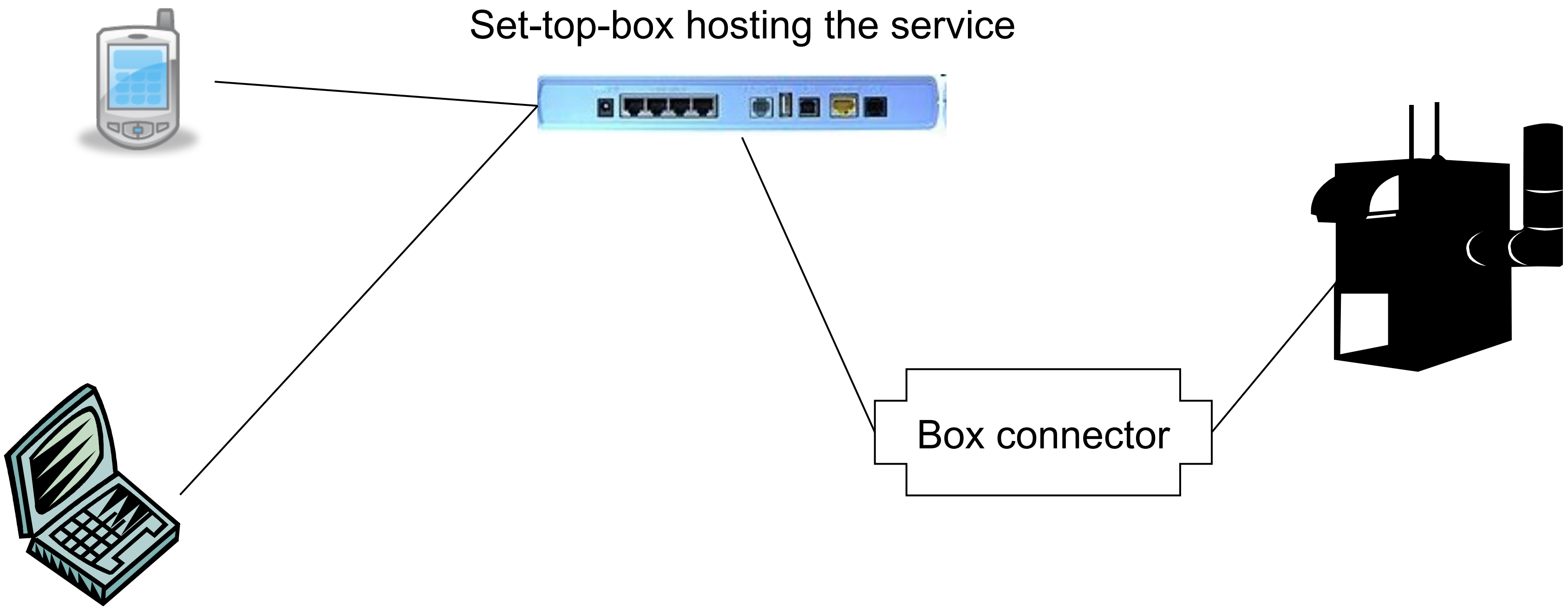
  - A distant heating control system
  - From a mobile terminal (specific app and web app)
  - Following standards xyzw
  - Secured by login/password

- Plan of project
  - Delivery at date x, tests date y…

Example: Distant heating control

Example

Feasability - General architecture



Step 2: Requirements

- Get a precise definition of the software:
  - Managed objects
  - Tasks to be applied on objects
  - Constraints
- To be built at this step
  - Requirements specifications
  - First version of user manual
  - Detailed plan of the project
  - Validation plan

Example: Distant heating control

Example

Requirements specifications

(it should be well written and organized - and precise and complete)

- From a smartphone or a standard laptop
  - Run a dedicated app or web app
  - Access secured with login/password
- Distant heating control
  - Control the central system, not each radiator independently
  - Start/stop the central system
  - Show current temperature
  - Tune expected temperature
- Service hosted on an open set-top box
  - Web service, web server…
- Command activation in less than a minute
- No more than 4 clicks for each user objective, app launching included

Example: Distant heating control

Example

Requirements specifications

- First version of user manual
  - First use
    - Download the app from store, provide login/pass, couple app and box
  - Daily use
    - Launch the app, click here and there to do this and that, screens mockups
  - List of possible errors and actions to correct
- Detailed plan of the remaining steps
  - List of tasks, durations, assigned persons, costs, intermediate deliveries,
- Validation plan
  - List of tests that will be run and checked at delivery
    - Must test all steps of the user manual, all error messages, all constraints

Step 3: Product design

- *define*
  - System architecture (hardware and software)
  - Data structures
  - Software organization of code (eg class diagram)
- Built at this step:
  - Product design document
    - System architecture
    - Definition of data structures
    - Modules and their roles, algorithms of difficult parts

Example: Distant heating control

Example

Design

- Server side
  - 1 Web server on set-top-box
  - 1 Web service hosted on the box that controls the central heating system, developed in Java/OSGi
  - 1 connector for the communication between the web server and the central heating system (with its characteristics)
- Client side
  - 1 web interface to communicate with the web service, jsp
  - 1 dedicated app for smartphones iOS + Android
  - Schemas, description of user interface, components interfaces

Example : Distant heating control

Example

Design

- Data structures
  - Temperatures history: table date, value
    - 12/12/2015 10:30:00, 15
  - Commands history: table login, date, action, parameters
    - Fla, 12/12/2015, setTemperature, 19
- Decomposition of the system into modules (architecture)
  - 1 web service, 5 functions
    - Boolean login(String login, String pass)
    - Int getTemperature()
    - Boolean setTemperature(int val)
    - Boolean start()
    - Boolean stop()
- Description of the role of each module

Step 4: Coding / implementation

  - Write programs with the selected language
  - Unit tests: individual validation of each module

- Built at this step
  - Programs code
  - Unit tests report: Launching and results of unit tests

Example: Distant heating control

Example

Implementation

- Code
  - public class xxx{…}
  - Automatic javadoc
- Tests
  - Use of JUnit
  - Launch tests batches on each funtion in an independent manner (e.g. use a simulator for heating connector to test the web service)
  - Test scenarios of all possible cases
  - Modification of code and/or of detailed design

Step 5: Integration

- Gather the different modules
- Integration tests: do inter-modules relations work well?

- Built at this step:
  - Integration tests report
  - Modifications of code and previous documents are frequent at this step. It can be necessary to go back to a previous step

Example : Distant heating control

Example

Integration

- Link the connector with the Web server
- Link the client app with the Web server
- Test end to end
- Go back to
  - Unit tests to better identify a pb,
  - Code if the pb is in code,
  - Design if the problem comes from design decisions
  - Etc.
- And re-run the whole steps!

Step 6: Recette / delivery

  - The client compares the delivery with what was expected
  - Contractual step!
- Built at this step
  - Recette report
  - Inspection and validation report

Example : Distant heating control

Example

Recette

The client takes the app in hand:

- He runs the validation plan built in step 2
  - Checks documentations
  - Follows installation manual, forces errors and checks robustness…
  - Studies the look and feel of user interfaces
  - Checks all expected functionalities, makes monkey tests…
- He checks that all constraints are respected
- He checks all versions

The client signs :>… or not :<

Step 7: Diffusion

  - Preparation and distribution of different versions (eg CD/DVD duplication, deposit in online stores, on a web page…)

- Built at this step
  - Different software versions
  - Their documentations
    - Installation document
    - User document
    - Administration document

Step 8: Exploitation

  - Put the system in its operating environment
  - Daily use
  - Software evolution
    - New functionalities and debug
- Built at this step
  - Installation report
  - Activities journal (log)
  - Incidents and corrections reports
  - Etc.

Remarks : 3 types of tests

- Unit Tests
  - Components are tested individually
  - It is run by the developer herself or by the test team
- Integration tests
  - Validation of the good inter-components collaborations
  - It is run by the test team or the project manager
- Recette Tests / validation
  - Verify the system answers the client requirements
  - Made in presence of / by the client

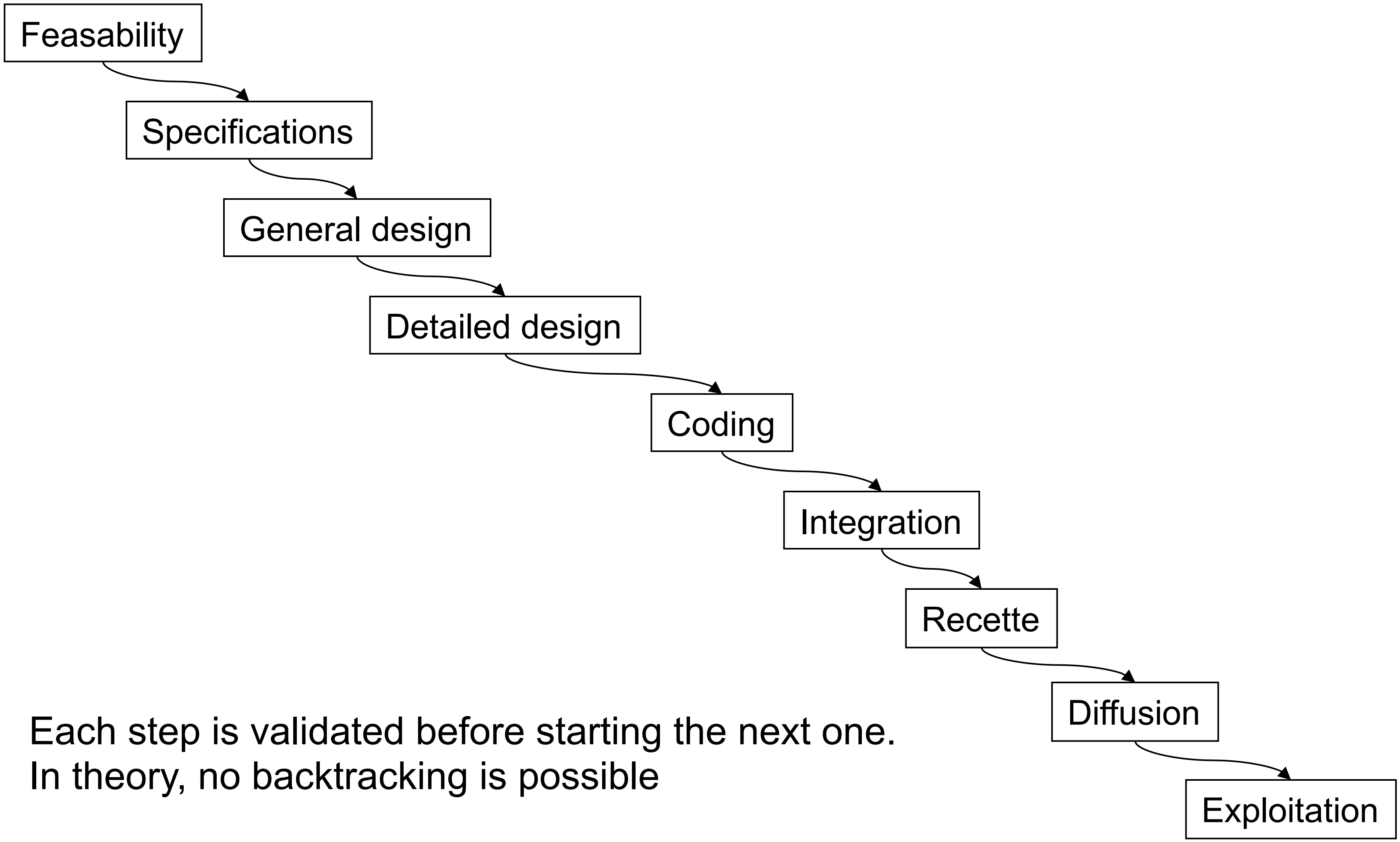
Life cycle : two « trends » today

- Classical Processes, directed by planification
  - All activities are planned in advance
  - Progress is measured regarding the plan
- Agile Processes, directed by adaptable iterations
  - Planification is decided and reviewed at each iteration
  - Changing the process is easier and allows to reflect evolutions of the project (requirements, law, hazards…)

Different models

- The first planed models
  - Cascade Model
  - Model in V shape
- The second generation of planed models
  - Incremental prototyping model
- Agile methods

Cascade model

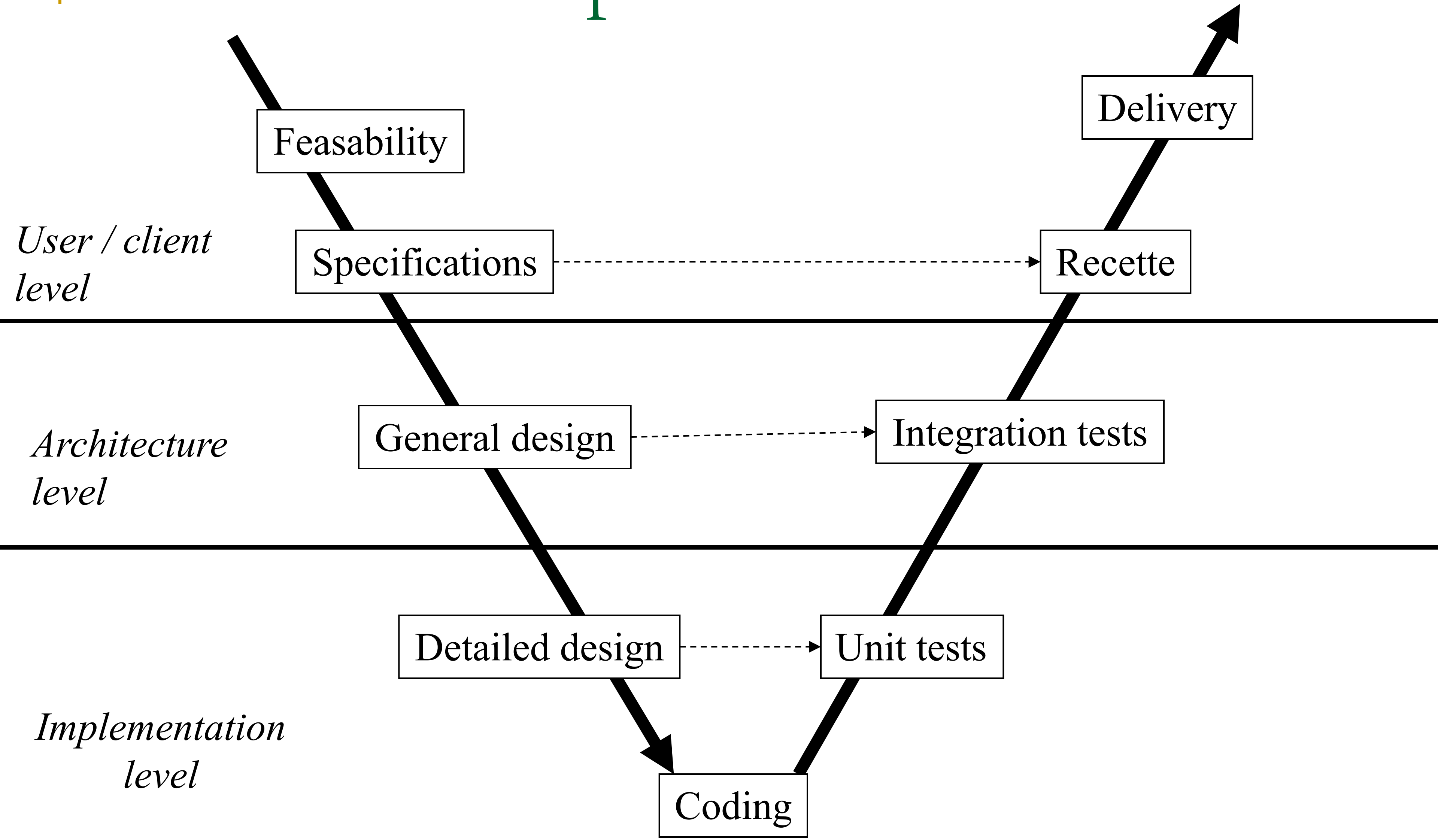


Each step is validated before starting the next one.
In theory, no backtracking is possible

Cascade model: difficulties

- Rigid decomposition of steps
  - Difficult to adapt to the evolution of users needs
  - Well adapted if specifications can be precise from the beginning and do not change
  - But it is very rare
- Tests are written very late

Model in V shape



Model in V shape

- Each level corresponds to a type of stakeholder
  - Software definition and validation
- Tests are scheduled as soon as specifications and design
  - They are then realized and validated by the corresponding people
    - E.g. : integration tests are made by the system architects, recette made by user/client

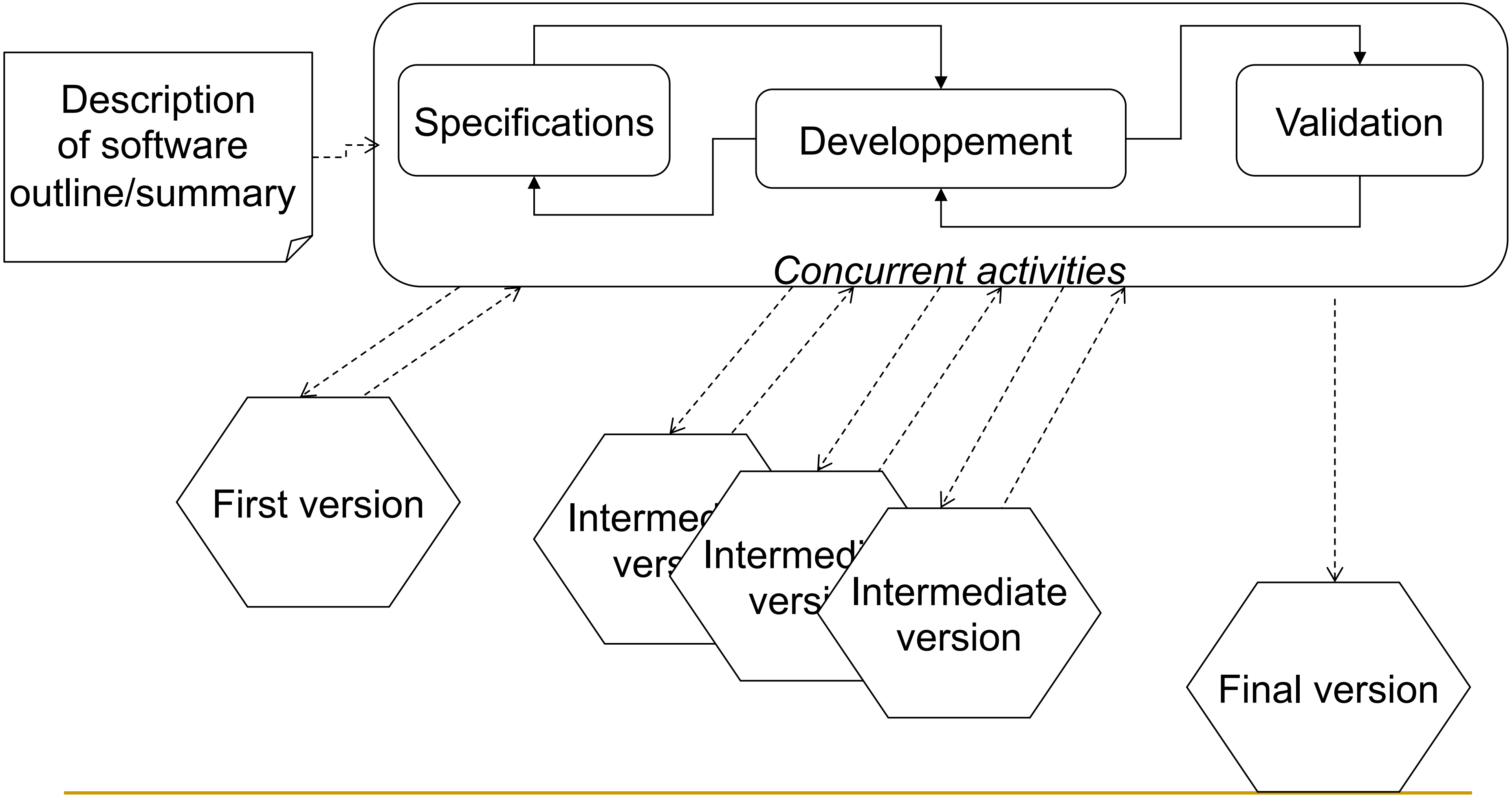
Model in V shape : synthesis

- Very similar to cascade model
  - Then similar difficulties
- The big plus
  - Tests management and stakeholders identification
- The main remaining default
  - non flexible, change is heavy to take into account

The second generation of models

- Better manage inevitable changes
  - The company changes => requirement changes
  - New technologies => improvement of implementations
- Changes = new tasks = new duties
  - Modification of design = re-design, re-coding, re-tests, re-integration, re-recette…
  - Functionalities input = re-specifications, re-design, re-coding, re-tests, re-integration, re-recette…
- The second generation methods and agile methods are specifically adapted to change management

Principles of incremental development



Principles of incremental development

- Each version can be realized independently from the others
  - In the 3 phases : specification, development and validation
- Clients can provide feedback of each version
  - Increased reactivity
  - Reduced costs for re-spec/dev/validation

Advantages of incremental development

- Reduced costs of adaptation to requirements evolution
  - Modification of a module design, not of the whole software => less documentation updates also
- Possible regular inclusion of the client / end user
  - Comments during versions presentations
  - The client/user sees the project moves forward
- Possible partial deliveries
  - The client can use pieces of the software before the end of the project

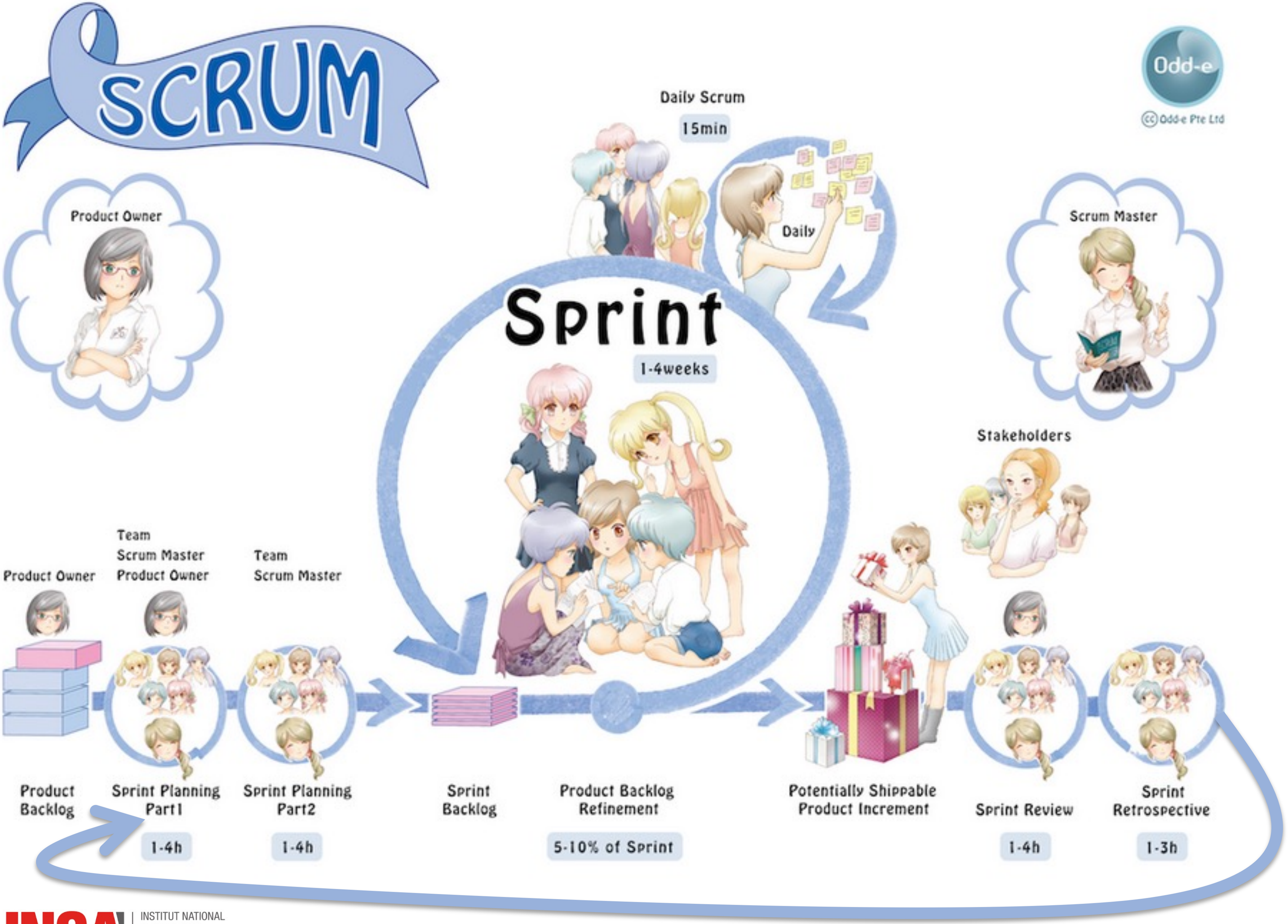
Risks of incremental development

- The global project progress is more difficult to follow
- If too many versions
  - Documentations are too fragmented
  - The structure of the global system may deteriorate
- Essential step in incremental dev., but often forgotten
  - Refactoring : reorganisation of code to make it « cleaner » without any changes in functionalities
  - NB: refactoring is also in place in agile methods!

Agile methods

More info in 4IF - PLD Agile

- Basics :
  - Have a global picture from the beginning
  - Contract on resources rather than result
  - Iterative process
    - Focus on next delivery
    - Precise requirements => design => code, test, integration => deliver
  - Collaborative team
    - All are « developers » and take part of all steps
    - End user representative inside the development team


SCRUM
Odd-e
© Odd-e Pte Ltd
Product Owner
Daily Scrum
15min
Daily
Scrum Master
Sprint
1-4weeks
Stakeholders
Product Owner
Team
Scrum Master
Product Owner
Team
Scrum Master
Product Backlog
Sprint Planning Part1
1-4h
Sprint Planning Part2
1-4h
Sprint Backlog
Product Backlog Refinement
5-10% of Sprint
Potentially Shippable Product Increment
Sprint Review
1-4h
Sprint Retrospective
1-3h