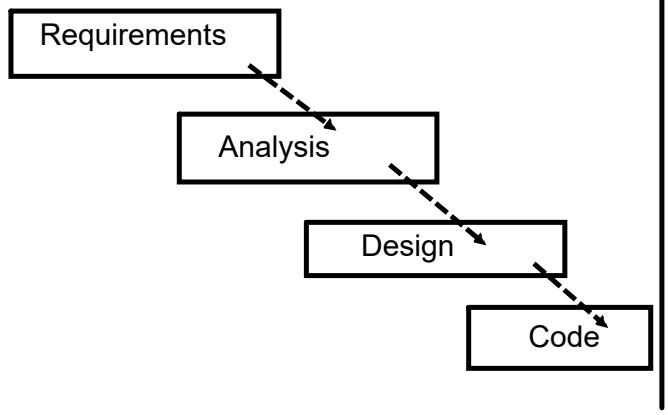
Agile Development process

#Different way of executing dev teams and project

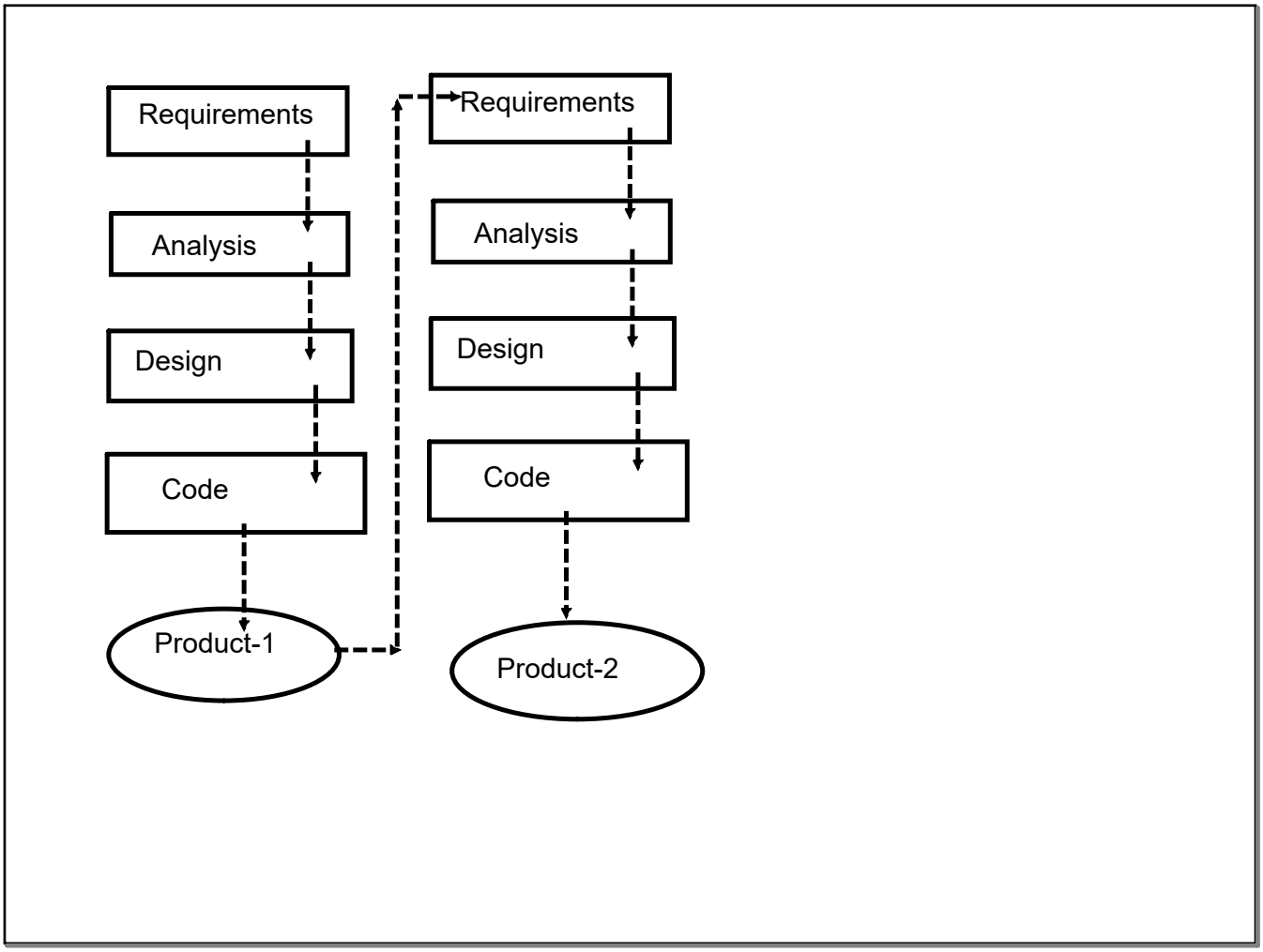
smooth process : flexible

iterations : something of value : put to world--> feedback



==> identify all req at beg

==> cannot accomodate new re


Requirements
Analysis
Design
Code
Product-1
Requirements
Analysis
Design
Code
Product-2

1. Iterative Incremental process
2. Time-Boxed (rapid/flexible)

Agile Manifesto:

=> Interaction : Individuals, process, tools

=> Focus on working software rather documentation

=> Customer collaboration Vs negotiation

=> Responding to changes

Agile embraces Unpredictability

12 principal

1. early and CD of valuable software
2. 2 weeks--2 month
3. even in late stage of dev quick welcome to changes
4. All roles must work together daily
5. trust : motivational env
6. interaction : face to face
7. primary measure of progess : working software + client acceptance
8. constant pace
9. continuos attn. on technical excellence + design
10. Simplicity
11. self-organizing teams
12. fine tuning of team org.

Misconception :

Its different

budget

unpredictability

developers only

Agile practical implementation

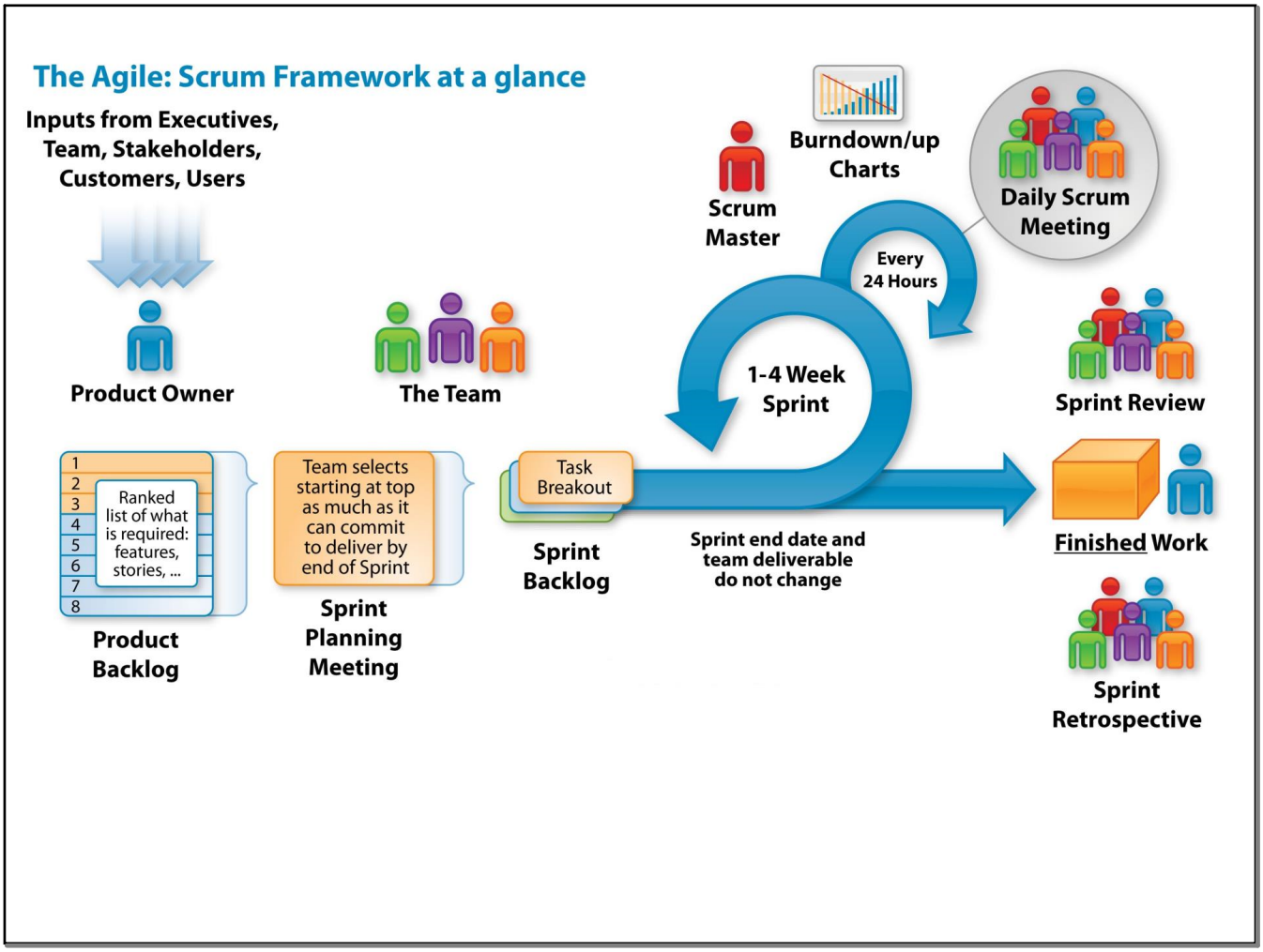
=>Scrum

=>Kanban

=>Hybrid

=> Lean

=> XP


The Agile: Scrum Framework at a glance
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective

Events : Activities

Roles : Team Organization

Artifacts : Documentation

Events :

- The sprint
- Sprint planning
- Daily Scrum meeting
- The Sprint Review
- The Spring Retrospective

Sprint : Core:

time - boxed 1-4 weeks : during which a potentially releasable product increment is created

Sprint Planning :

4 hrs for 2 week sprint

Input:

- Artifacts : Product backlog
- last product increment
- projected capacity of team
- past performance of team

Output :

- what need to be delivered
- how to achieve

Daily Scrum Meeting : 15 mins

quick explanation by each team member

Sprint review Meeting :

2hrs for 2 week sprint

wide variety of attendees

Sprint Retrospective

- 1hrs for 2 week sprint
- analysis

Team Organization

Scrum Master

Keeper of Scrum

Product Owner

single point interaction on product status

Team:

functional : (cross-functional)

anyone who is relevant to current sprint

5-10

Artifacts : Documents:

Product Backlog

- priortized list of features
- evolving artifact
- managed by Product Owner

Sprint Backlog

- features to be implemented in current sprint
- Sprint team

increment document

sum of all Product backlog items completed during recent sprint,

combined with increments in prev increments

Burnup/down chart

tracking progress of sprint

USer-Stories (standard structure)

As a <type of user>

I want <to perform some task>

So that <I can achieve some goal>

Eg:

As a "bank customer"

I want to "withdraw cash from an ATM"

So that "I don't have to wait in line in the bank"

Acceptance Criteria:

Given :

-account is valid

-card is valid

Extreme Programming (XP)

Software are organic : "grow the software"

Goals :

1. Minimize the unnecessary work
2. Maximize the communication & feedback (Customer)
3. developers do most important work
4. Make system flexible

History

Kent Back (1999) : "Extreme Programming Explained"

1960s : NASA : "Test first development" (TDD/BDD)

XP Practices

1. On-Site customer
2. Planning Game
3. Small releases
4. Simple design
5. Refactoring
6. Metaphor
7. Pair Programming
8. Collective ownership
9. Continuous integration
10. 40 hour week
11. Coding standards

XP Process

Multiple short cycles (2 weeks)

1. Meet with client to elicit requirement

   User stories + acceptance tests

2. Planning Game

   # Break stories in task, estimate cost

   # Client prioritize stories

3. Implementation

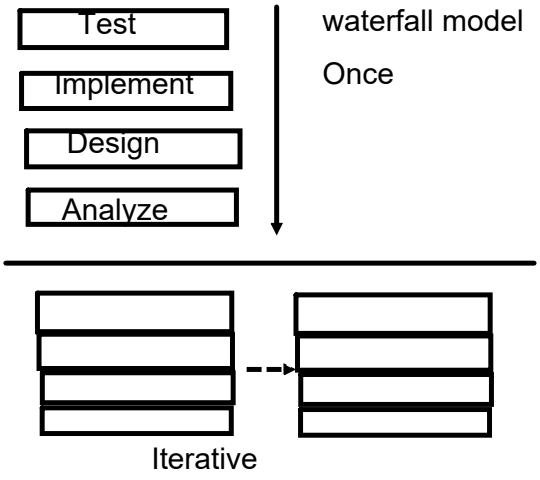
   => TDD, write the test first

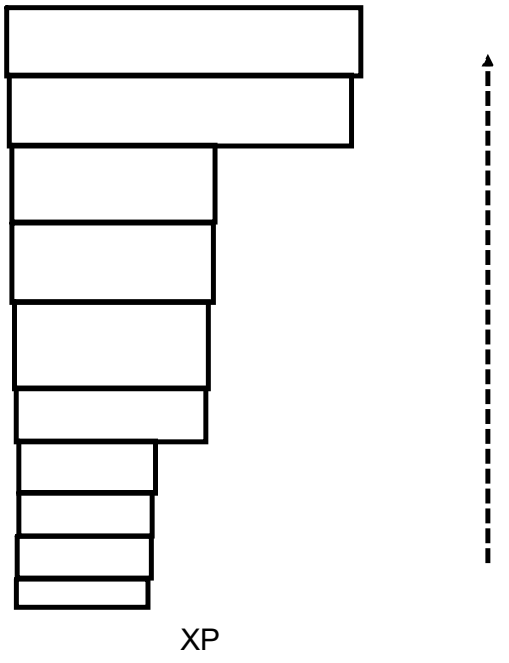
   => Simplest possible design to pass the test

   => Code in pairs

   => refactor the code

4. Evaluate progress and repeat from step 1





XP Customer

Expert Customer

On site, always available

-Clarifies req

-Negotiates with team

-Write and run the acceptance

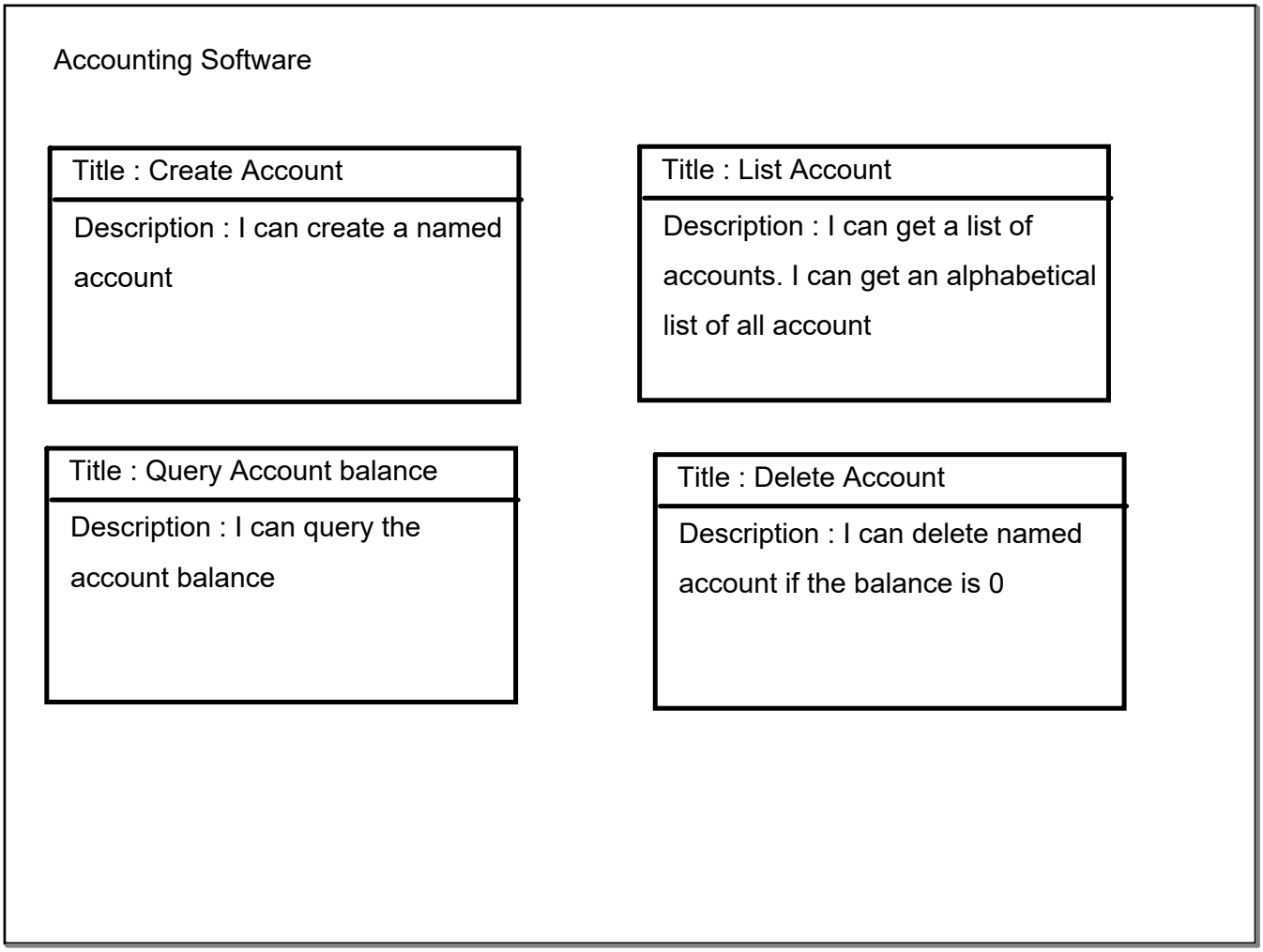
-evaluates iterative version

Planning Game

User Stories :

index cards : title

USes client language


Accounting Software
Title : Create Account
Description : I can create a named account
Title : List Account
Description : I can get a list of accounts. I can get an alphabetical list of all account
Title : Query Account balance
Description : I can query the account balance
Title : Delete Account
Description : I can delete named account if the balance is 0

Not a user story

Title: Use Ajax for UI
Description : The user interface will use AJAX tech to provide cool UX

Customer Acceptance test

Client describes the how user stories will be tested

Eg:

1. If i create an account "savings", and another account " current", then if i ask for list, i must get diffrentiated on named account
2. If i now try to create "current",--->error
3.

Test must be automated

Task (developer centered)

Story is broken into task

Eg:

"ask customer name"

"find if not exists"

"create empty account"

1. Break down only as much as needed
2. Validate the breakdown with customer

Team assign a cost to tasks

use abstract "unit"

Decide smallest task : 1 unit

Pick task => find completion date

Implementation:

acceptence test : unit

```
eg:
  addAccount("current");
if(exists("current")) throw
try{
      addAccount("current"));
}catch(----------){
}
```

Courage to refactor

Simplicity : Just in time design

No premature optimization

: You are not going to need it (YAGNI)


REfactor
feedback
Improvise
+
Regression Testing
Integrating it a official release
Hudson, CruiseControl, Jenkins, GitLab

Pair Programming:

Pilot & Co-Pilot

Evaluation:

-Run acceptance test

-Assess : completion

-Discuss the problem

-compute the speed to team

-Re-estimate the remaining user-stories

-Plan with client: next iteration

No specialized role

every XP Programmers participates in all

No up-front design

start with quick analysis

=> quickly delivering business value is the driver of XP

BDD :