Software Engineering

Management of IT projects

Silesian University of Technology

Agenda

Software lifecycle

- A sequence of actions and activities performed during software development

Software lifecycle

- A sequence of actions and activities performed during software development
- Main activities in software development:
  - requirements engineering
  - analysis
  - design
  - testing
  - maintenance

Software lifecycle

- A sequence of actions and activities performed during software development
- Main activities in software development:
  - requirements engineering
  - analysis
  - design
  - testing
  - maintenance
- Software development lifecycle (SDLC) models do not include the activities related with management

Methodology

- Approach to project management
  - a set of guiding principles and processes for managing a project
  - very often they are defined independently from a project domain

Methodology

- Approach to project management
  - a set of guiding principles and processes for managing a project
  - very often they are defined independently from a project domain
- Goals of defining the methodology
  - unambiguously defined cooperation principles
    - various detail level
    - roles and responsibilities
    - organization culture
    - documentation
  - clarity and transparency—everyone knows what to do
  - communication—formal and informal

IT project

Project

PRINCE2: A temporary organization that is created for the purpose of delivering one or more business products according to an agreed business case

PMBOK: A temporary endeavor undertaken to create a unique product or service

IT project

Project

PRINCE2: A temporary organization that is created for the purpose of delivering one or more business products according to an agreed business case

PMBOK: A temporary endeavor undertaken to create a unique product or service

- Features of a project
  - clear beginning and the end
  - unique outcome

IT project

Project

PRINCE2: A temporary organization that is created for the purpose of delivering one or more business products according to an agreed business case

PMBOK: A temporary endeavor undertaken to create a unique product or service

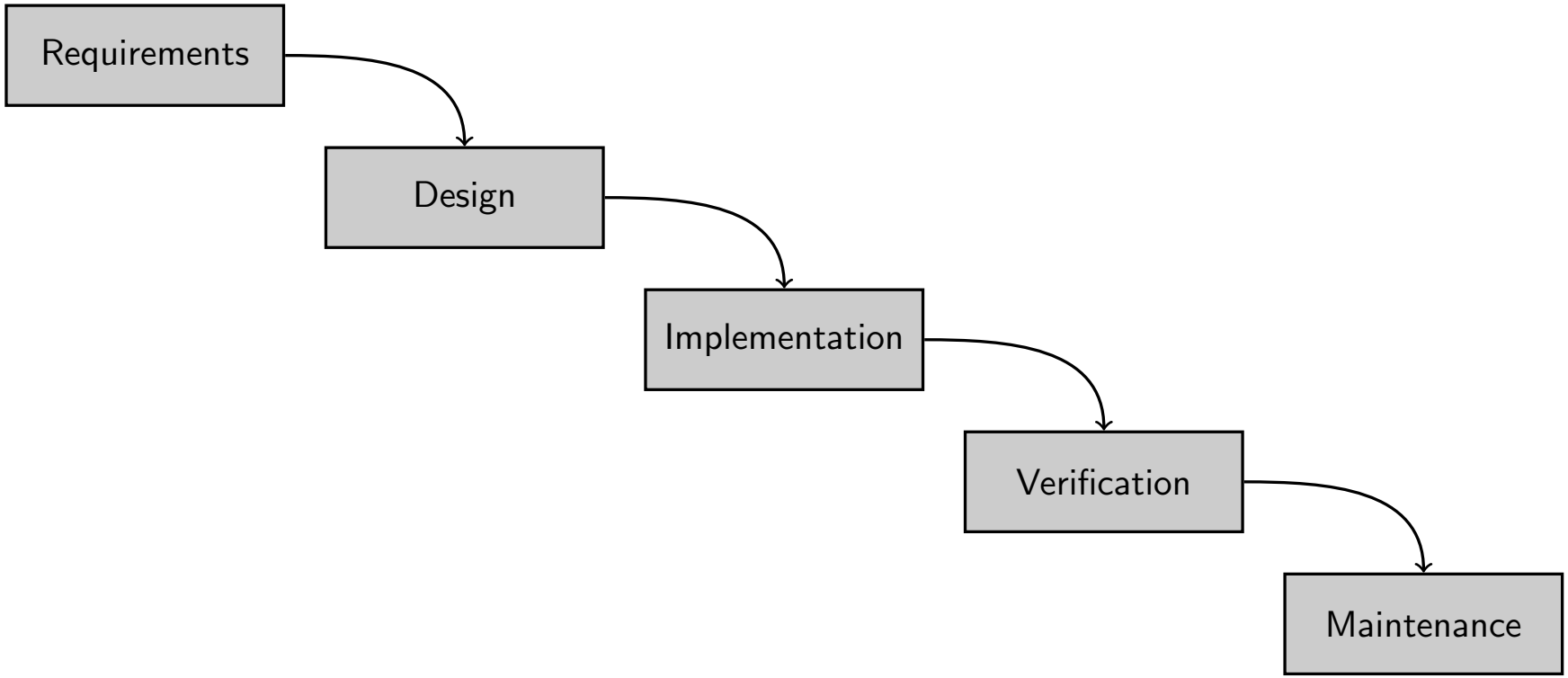
- Features of a project
  - clear beginning and the end
  - unique outcome
- What is NOT a project?
  - ongoing activity
  - repeatable activity
  - research or development without a clear goal

Agenda

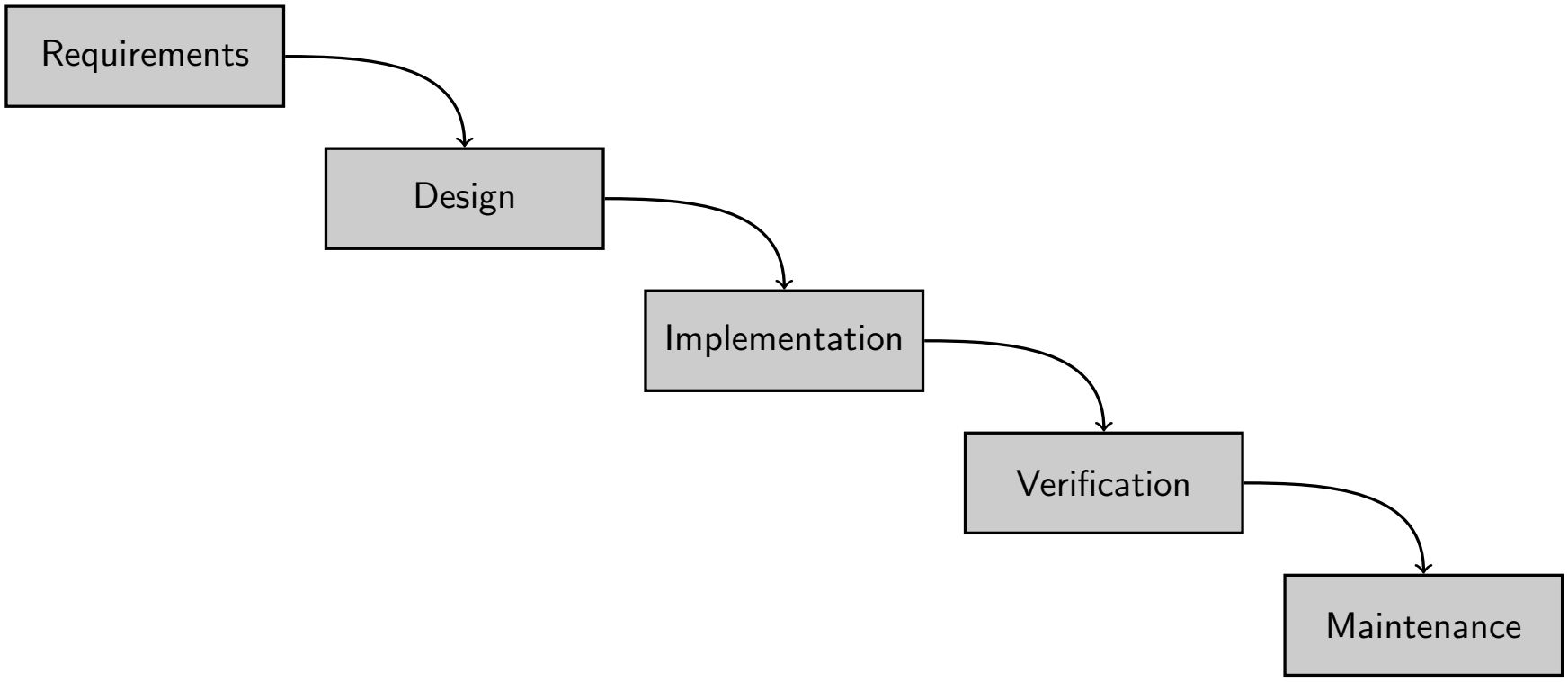
Software lifecycle models

- Basic models
  - waterfall development
  - incremental development
  - V-model
  - prototyping
  - spiral development

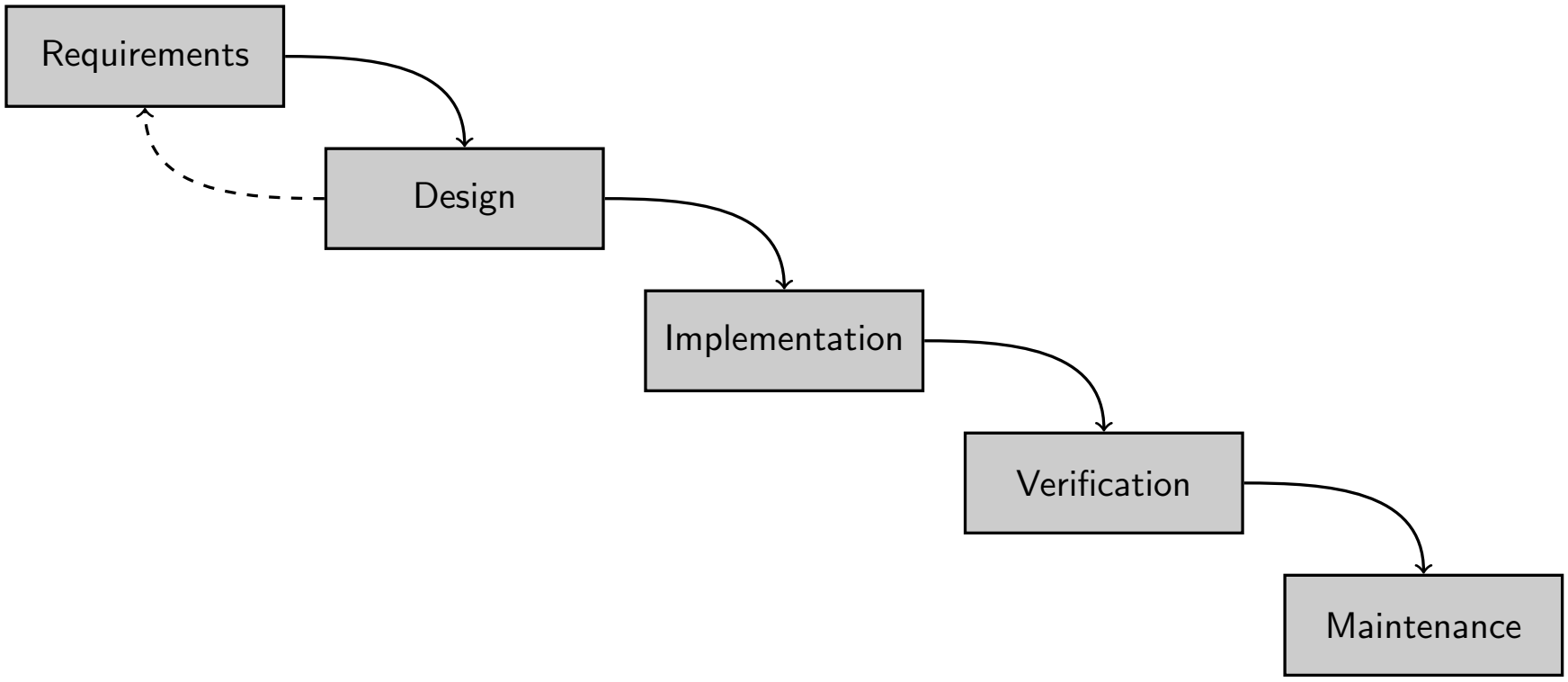
Waterfall development



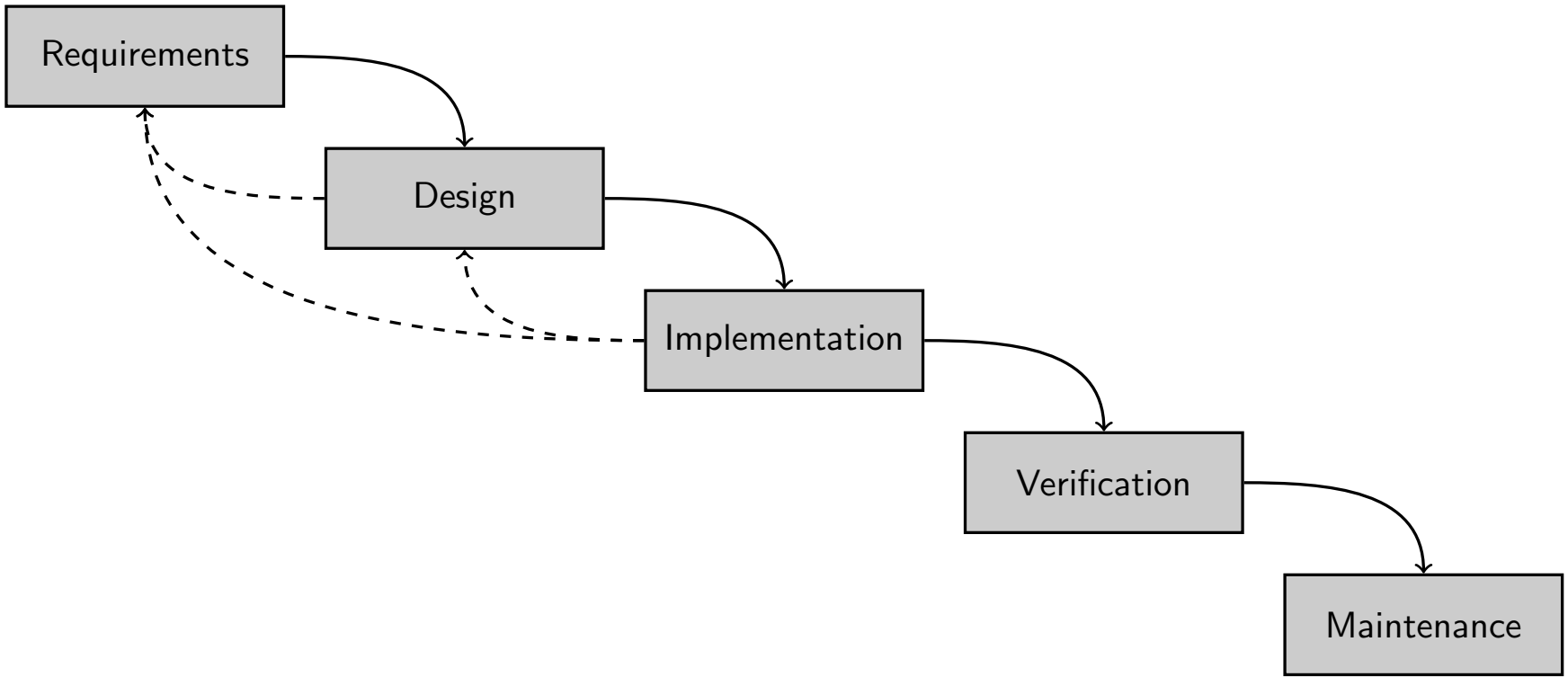
Waterfall development with iterations



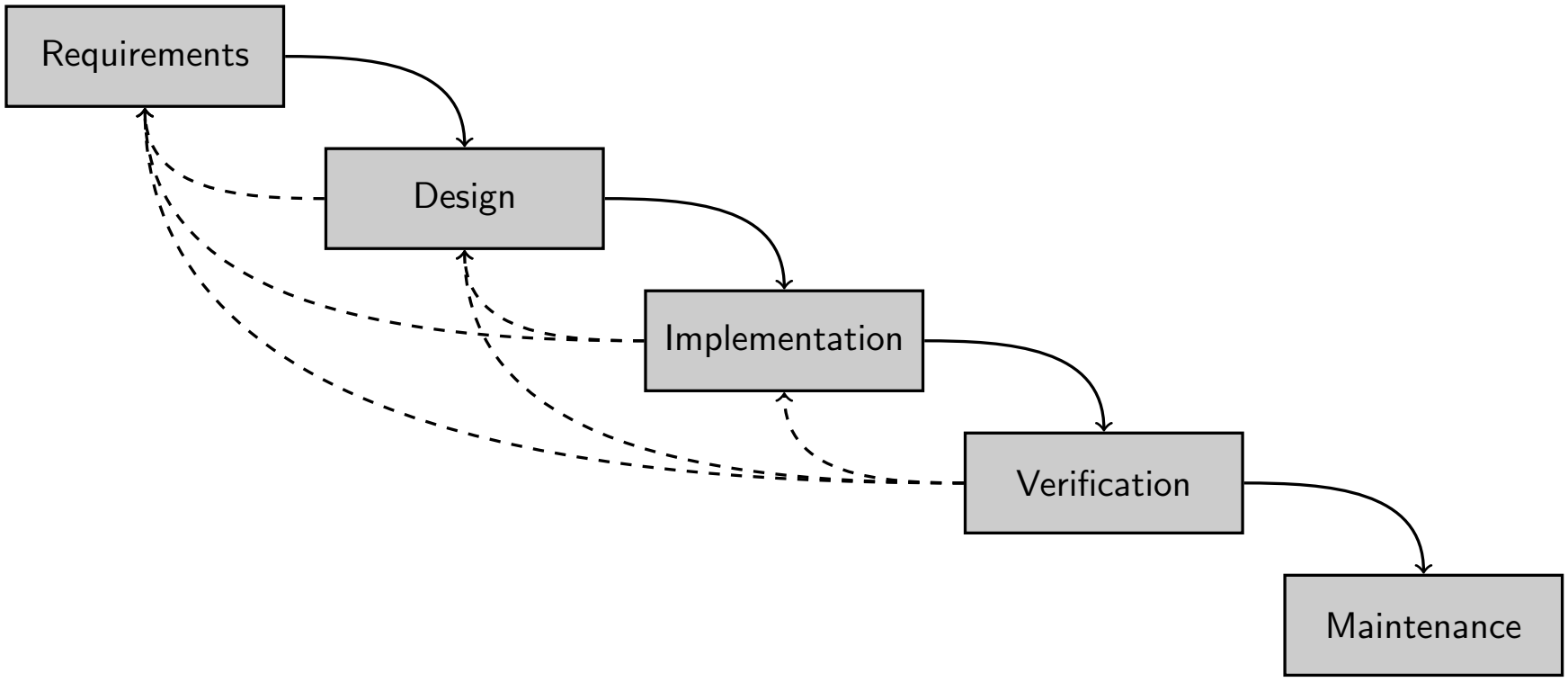
Waterfall development with iterations



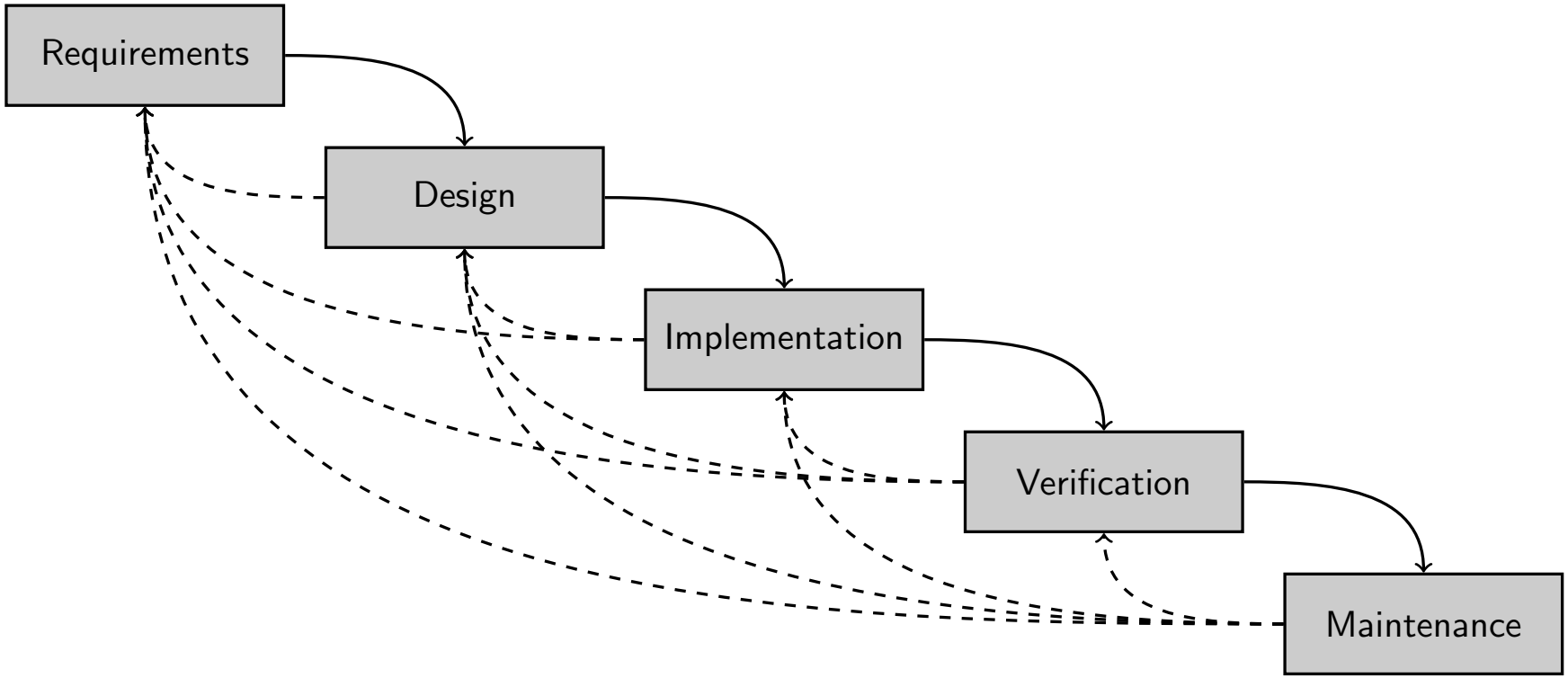
Waterfall development with iterations



Waterfall development with iterations



Waterfall development with iterations



Waterfall development

- Waterfall—disadvantages
  - low flexibility
  - high cost of making a change (especially requirements)

Waterfall development

- Waterfall—disadvantages
  - low flexibility
  - high cost of making a change (especially requirements)

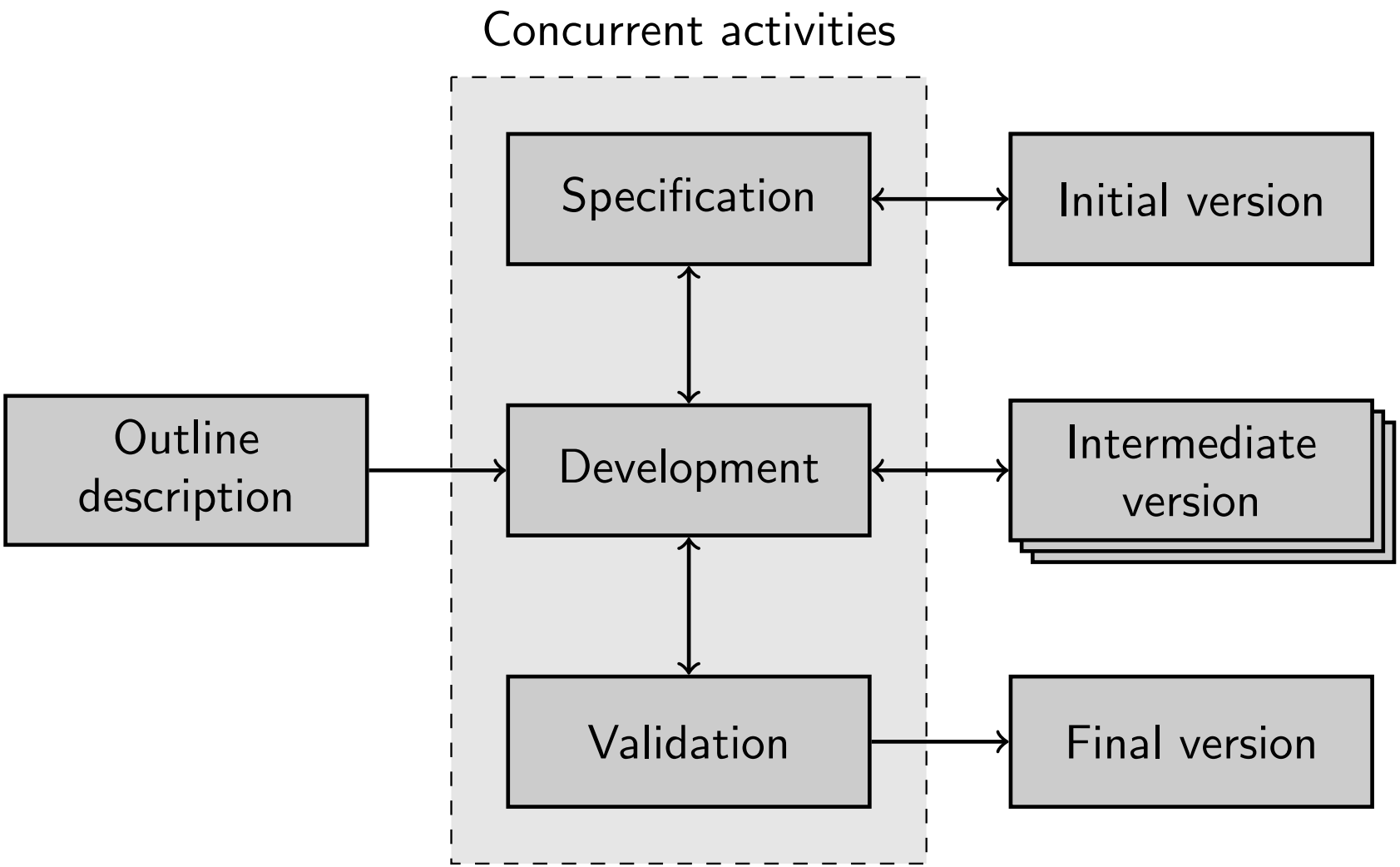
Waterfall development

- Waterfall—disadvantages
  - low flexibility
  - high cost of making a change (especially requirements)
- Waterfall—advantages
  - transparency of the process
  - encourages taking more effort in the early stages $\rightarrow$ may result in clearer requirements and better design

Waterfall development

- Waterfall—disadvantages
  - low flexibility
  - high cost of making a change (especially requirements)
- Waterfall—advantages
  - transparency of the process
  - encourages taking more effort in the early stages $\rightarrow$ may result in clearer requirements and better design
- When to use?
  - stable and well-understood requirements
  - large-scale projects that require careful planning

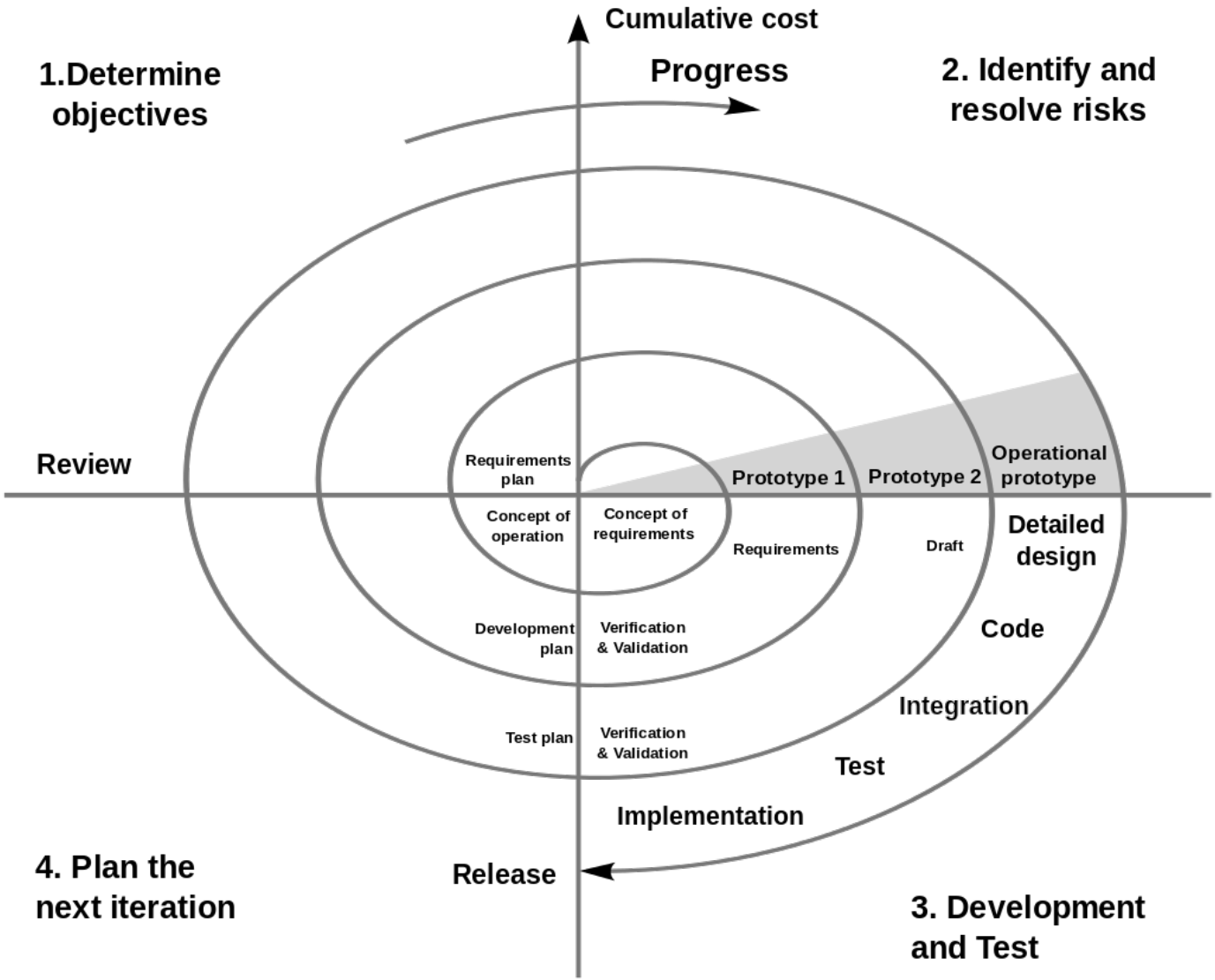
Incremental development



Incremental development

- Incremental development—benefits
  - lower cost of changes
  - mitigates the risk of missing the user requirements (frequent delivery)
  - faster delivery—the users may start using an early version of the system if they find it useful
- Incremental development—disadvantages
  - poor transparency of the process
  - regular deliverables are necessary to measure progress—creating new versions is not cost effective
  - the code structure tends to degrade with frequent changes—need for refactoring

Spiral model



Spiral model

- Spiral model—waterfall combined with iterative approach

Spiral model

- Spiral model—waterfall combined with iterative approach
- Spiral model—benefits
  - requirements may be defined on the fly—changes easier to accommodate
  - a space for customer feedback
  - easier risk management
  - development is fast and systematic
- Spiral model—disadvantages
  - difficult to estimate the total cost—risk of not meeting the schedule or budget
  - not appropriate for small projects

Spiral model

- Spiral model—waterfall combined with iterative approach
- Spiral model—benefits
  - requirements may be defined on the fly—changes easier to accommodate
  - a space for customer feedback
  - easier risk management
  - development is fast and systematic
- Spiral model—disadvantages
  - difficult to estimate the total cost—risk of not meeting the schedule or budget
  - not appropriate for small projects
- When to use:
  - large projects
  - frequent delivery required
  - unclear and complex requirements
  - high-risk projects

Lifecycles—comparison

Properties of Model	Water-Fall Model	Incremental Model	Spiral Model	Rad Model
Planning in early stage	Yes	Yes	Yes	No
Returning	No	Yes	Yes	Yes
Handle Large-Project	Not Appropriate	Not Appropriate	Appropriate	Not Appropriate
Detailed doc.	Necessary	Yes but not much	Yes	Limited
Cost	Low	Low	Expensive	Low
Requirement spec.	Beginning	Beginning	Beginning	Time boxed release
Flexibility to change	Difficult	Easy	Easy	Easy
User Involvement	Only at beginning	Intermediate	High	At the beginning
Maintenance	Least	Promoted	Typical	Easily Maintained
Duration	Long	Very long	Long	Short
Risk Involvement	High	Low	Medium to high risk	Low
Framework Type	Linear	Linear + Iterative	Linear + Iterative	Linear
Testing after:	Coding	Every iteration	Engineering phase	Coding
Overlapping Phases	No	Yes	No	Yes
Maintenance	Least Maintainable	Maintainable	Yes	Easily Maintainable
Re-usability	Least possible	To some extent	To some extent	Yes
Time-Frame	Very Long	Long	Long	Short
Working software	Life-cycle end	Iteration end	Iteration end	Life cycle end
Objective	High Assurance	Rapid Development	High Assurance	Rapid development
Team size	Large Team	Not Large Team	Large Team	Small Team
Customer control	Very Low	Yes	Yes	Yes

Source: `https://www.guru99.com/compare-waterfall-vs-incremental-vs-spiral-vs-rad.html`

Agenda

1. General terms
2. Software lifecycle models
3. Methodologies
   - PRINCE2
   - Agile techniques
   - Extreme programming
   - Crystal Clear
   - Scrum
   - Kanban
4. Team roles and team building
5. Software cost estimation

Methodologies—taxonomy (level of formalism)

- "Heavyweight" methodologies
  - PRINCE2
  - PMI / PMBOK
  - Rational Unified Process (RUP)

Methodologies—taxonomy (level of formalism)

- "Heavyweight" methodologies
  - PRINCE2
  - PMI / PMBOK
  - Rational Unified Process (RUP)
- "Lightweight" methodologies
  - agile methodologies
    - extreme programming
    - SCRUM
    - Crystal Clear
  - Kanban (a lean methodology)

Methodologies—taxonomy (application-based)

- Management methodologies
  - PRINCE2
  - PMI / PMBOK
- Development methodologies
  - SCRUM
  - Rational Unified Process
  - Enterprise Unified Process
  - Agile Unified Process

PRINCE2—general information

- A heavyweight (management) methodology
  - PRojects IN Controlled Environments (PRINCE)
  - controlled project environment
  - clearly defined competencies and responsibilities
- Key features:
  - product-based planning
  - continued business justification
  - defined structure of organisation
  - project divided into stages

PRINCE2—general information

- Elements:
  - principles—good practices, universal rules...
  - themes—key aspects of managing a project
  - processes—7 main actions
  - tailoring PRINCE2 to the project environment

PRINCE2—general information

- Elements:
  - principles—good practices, universal rules...
  - themes—key aspects of managing a project
  - processes—7 main actions
  - tailoring PRINCE2 to the project environment
- Processes
  1. Starting up a project
  2. Initiating a project
  3. Directing a project
  4. Controlling a stage
  5. Managing a stage boundary
  6. Managing product delivery
  7. Closing a project

PRINCE2—management

- Project management—control cycle
  - planning
  - delegating
  - monitoring
  - control

PRINCE2—principles

- Seven principles of PRINCE2:
  1. Continued business justification...
  2. Learn from experience—a lessons log
  3. Defined roles and responsibilities...
  4. Manage by stages...
  5. Manage by exception...
  6. Focus on products...
  7. Tailor to suit the environment...

PRINCE2—principles

- Continued business justification
  - a project must make sense
  - foreseen benefits of a project
  - business justification must be documented
  - ...and continuously being updated
    - what may change the business justification of a project?

PRINCE2—principles

- Defined roles and responsibilities
  - what is expected from me?
  - what can I expect from others?
  - who takes decisions on what?

PRINCE2—principles

- Manage by exception
  - delegate the power
  - define the tolerance
  - no formal management required unless and exception occurs
- Six effectiveness criteria
  - budget
  - time
  - scope
  - quality
  - risk
  - benefits

PRINCE2—principles

- Manage by stages
  - the project split into smaller chunks that are easier to manage
  - a checkpoint after each stage—to continue or not to continue
  - two stages at minimum:
    - initiation stage
    - execution stage

PRINCE2—principles

- Focus on products
  - named and defined products that are to be produced
  - acceptance criteria
  - product breakdown structure
- Tailor to suit the environment
  - adapting PRINCE2 to the needs of a project
  - environment
  - scale
  - complexity
  - risk
  - significance

PRINCE2—themes

- Themes in PRINCE2:
  1. Business Case
  2. Organisation
  3. Quality
  4. Plans
  5. Risk
  6. Change
  7. Progress

PRINCE2—themes

- Themes in PRINCE2:
  1. Business Case
     - Related to the *continued business justification* principle. This theme provides knowledge about whether a project is worthwhile and achievable.
  2. Organisation
  3. Quality
  4. Plans
  5. Risk
  6. Change
  7. Progress

PRINCE2—themes

- Themes in PRINCE2:
  1. Business Case
  2. Organisation
     - Related to the *defined roles and responsibilities* principle. The organisation theme requires project managers to have everyone's roles and responsibilities on record.
  3. Quality
  4. Plans
  5. Risk
  6. Change
  7. Progress

PRINCE2—themes

- Themes in PRINCE2:
  1. Business Case
  2. Organisation
  3. Quality
     - Related to the *focus on products principle*. Quality can be an abstract concept, so defining it at the beginning of a project is vital to keeping the work on track.
  4. Plans
  5. Risk
  6. Change
  7. Progress

PRINCE2—themes

- Themes in PRINCE2:
  1. Business Case
  2. Organisation
  3. Quality
  4. Plans
     - A plan describes how targets will be achieved. It focuses on the products, timescale, cost, quality and benefits.
  5. Risk
  6. Change
  7. Progress

PRINCE2—themes

- Themes in PRINCE2:
  1. Business Case
  2. Organisation
  3. Quality
  4. Plans
  5. Risk
     - The purpose of this theme is to identify, assess and control uncertain events during a project. These are recorded in a risk log. Negative risks are called threats and positive ones are called opportunities.
  6. Change
  7. Progress

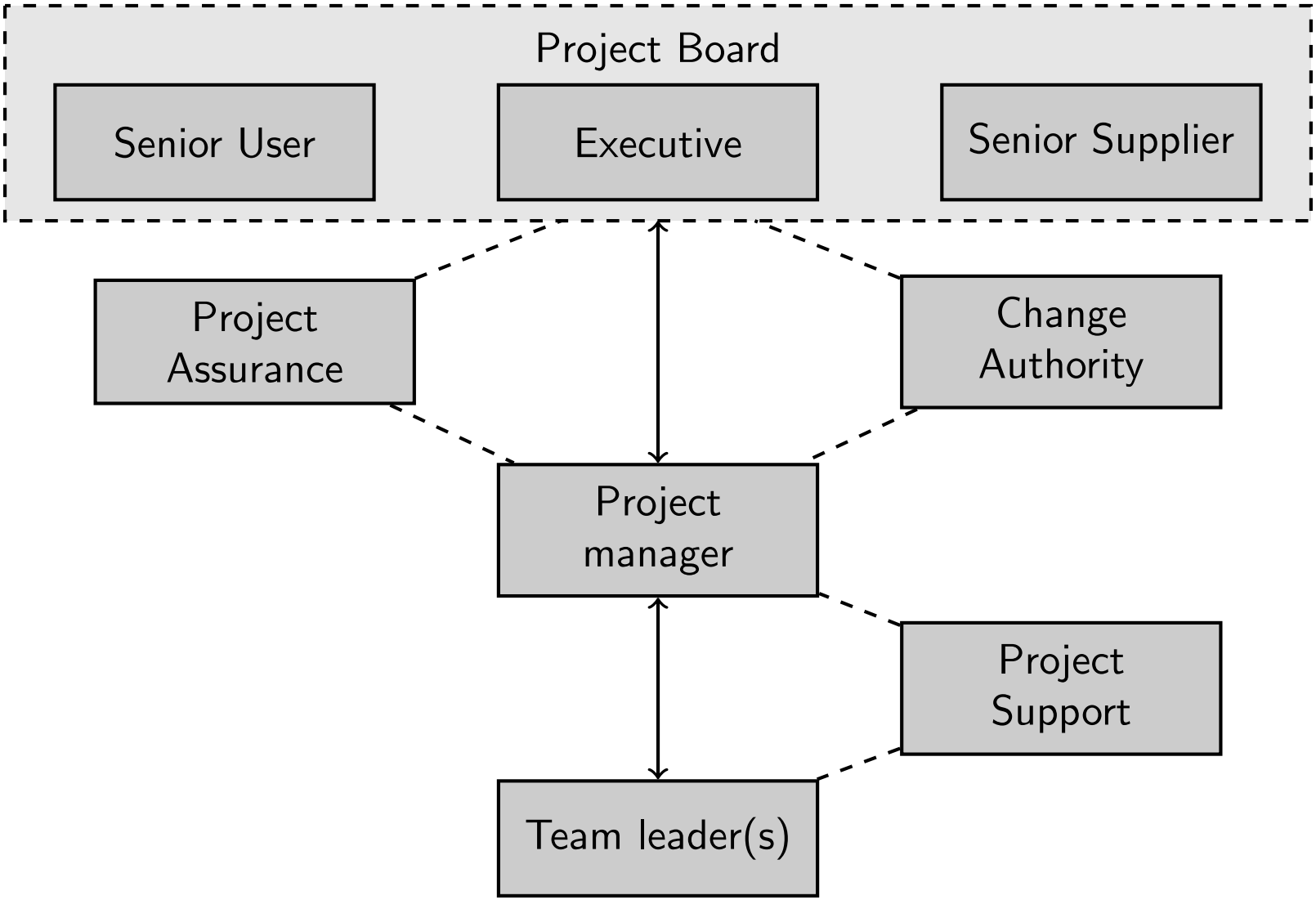
PRINCE2—themes

- Themes in PRINCE2:
  1. Business Case
  2. Organisation
  3. Quality
  4. Plans
  5. Risk
  6. Change
     - This theme is about handling change requests and issues that arise during the project. The idea is not to prevent changes, but to get them agreed on before they're executed.
  7. Progress

PRINCE2—themes

- Themes in PRINCE2:
  1. Business Case
  2. Organisation
  3. Quality
  4. Plans
  5. Risk
  6. Change
  7. Progress
     - Progress is about tracking the project. This allows project managers to check and control where they are relative to the plan. Not only can projects go off the rails without this—or any one—of the themes, but by not tracking, you may not even be aware that it's happening.

PRINCE2—project management structure



PRINCE2—project management structure

- PRINCE2—four management levels
  1. Corporate or Programme Management
     - outside the project structure
     - initiates the project
  2. Direction
     - the level of Project Board
  3. Management
     - the level of Project Manager
  4. Delivery
     - the production level

PRINCE2—project management structure

Project Board

- represents the most important entities
  - Executive
    - responsible for the project
    - takes financial decisions
    - represents the business needs
    - solves the conflicts within the board
  - Senior User
    - represents future users
    - ensures the user needs
  - Senior Supplier
    - represents the production team(s)
    - responsible for defining feasible requirements
    - manages the supplier's resources
- can assign people to the project
- strategic management (resource assignment)
- accepts the plans

PRINCE2—project management structure

- Project Manager
  - responsible for the day to day management
  - reports to the Project Board
  - Project Board defines the scope of competencies

PRINCE2—project management structure

- Project Manager
  - responsible for the day to day management
  - reports to the Project Board
  - Project Board defines the scope of competencies
- Team Leader / Team Manager
  - responsible for delivering the products of appropriate quality
  - reports to Project Manager

PRINCE2—project management structure

- Project Manager
  - responsible for the day to day management
  - reports to the Project Board
  - Project Board defines the scope of competencies
- Team Leader / Team Manager
  - responsible for delivering the products of appropriate quality
  - reports to Project Manager
- Team
  - develops the product
  - reports to Team Leader

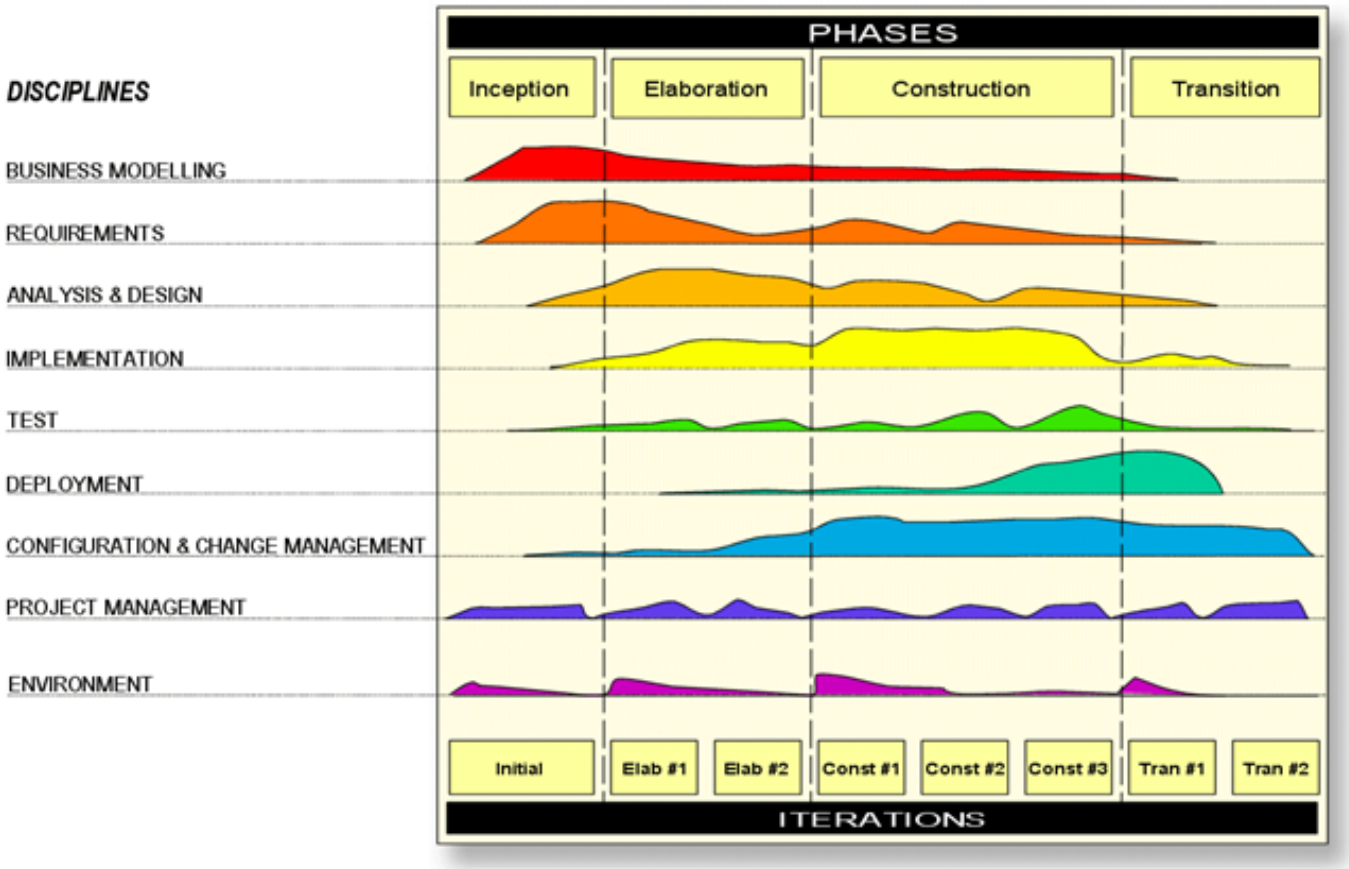
PRINCE2

- When to use PRINCE2?
  - large projects (but can be adapted to smaller ones)
  - detailed documentation required (certification purposes)
  - collaboration with many entities

PMI / PMBOK

- Heavyweight (management) methodology
  - collection of good practices—can be implemented gradually
  - 47 processes, two grouping criteria
    - area of knowledge (ten areas)
    - management cycle (five groups)
  - emphasis on work needed to achieve the project objective (work breakdown structure - WBS)
  - focused on the project manager (the methodology does not go any further)
  - success is a product that meets the requirements (collected during the project)

Rational Unified Process



Agile Manifesto

- Agile Manifesto—February 2001, Utah

Agile	**Traditional methodologies**
Individuals and interactions	Processes and tools
Working software	Comprehensive documentation
Customer collaboration	Contract negotiation
Responding to change	Following a plan

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential

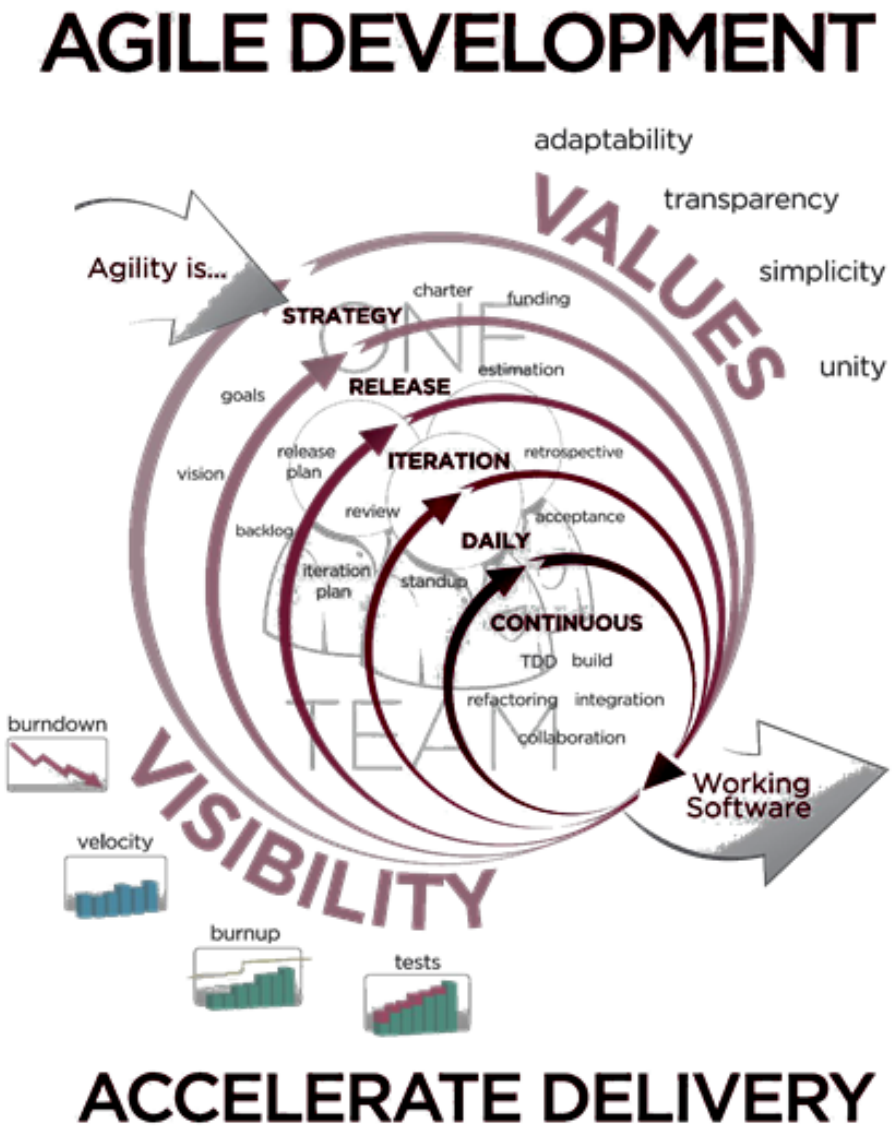
Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Self—organizing teams

Agile Manifesto

1. Customer satisfaction by early and continuous delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Self—organizing teams
12. Regular adaptation to changing circumstance

Agile development

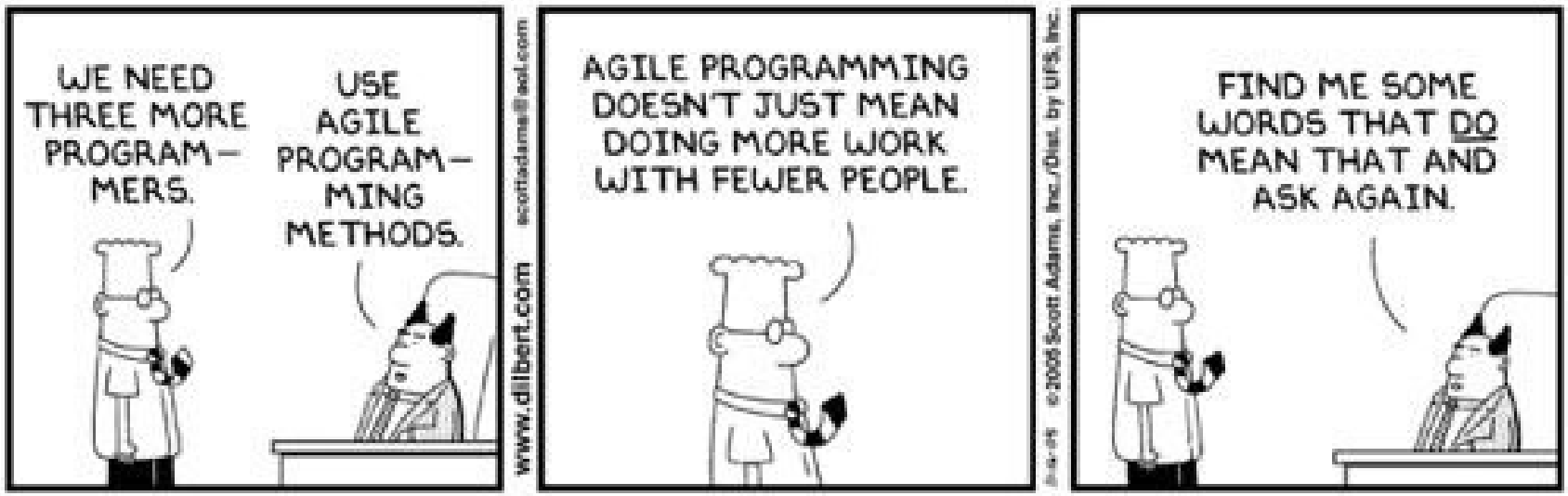


Agile development

- Agile—common features
  - responsibility passed to the team
  - high success ratio given unclear requirements
  - stress on values and communication

Agile development

- Agile—common features
  - responsibility passed to the team
  - high success ratio given unclear requirements
  - stress on values and communication



© Scott Adams, Inc./Dist. by UFS, Inc.

Is agile just a buzzword?

Is agile just a buzzword?

Cliff G., Agile Practitioner for Almost 20 Years

Answered Jan 30, 2019

No, "agile" is a product development philosophy that is based on direct customer feedback (or as close as you can get to it) and regular delivery of customer-facing value as a measure of success.

Many organizations don't take the time to understand this, even though it's very simply laid out in 4 statements and 12 principles. Instead, they just adopt some methodology loosely related to the Agile philosophy, implement it poorly and without understanding the underpinnings, and then blame the consultant that they hired without paying attention to their credentials for screwing things up.

Agility is a real thing that companies can harness to deliver better products, usually faster. Unfortunately, most are unwilling to change some of their core beliefs and cultural infrastructure sufficiently to realize those rewards.

Extreme programming—the characteristics

- The "extreme" approach to iterative development:
  - new versions may be built several times per day
  - increments are delivered to customers every 2 weeks
  - all tests must be run for every build, the build is only accepted if tests run successfully

XP and agile principles

- "Incremental development"
  - small, frequent system releases

XP and agile principles

- "Incremental development"
  - small, frequent system releases
- "Customer involvement"
  - full-time customer engagement with the team

XP and agile principles

- "Incremental development"
  - small, frequent system releases
- "Customer involvement"
  - full-time customer engagement with the team
- "People not process"
  - pair programming
  - collective ownership
  - a process that avoids long working hours

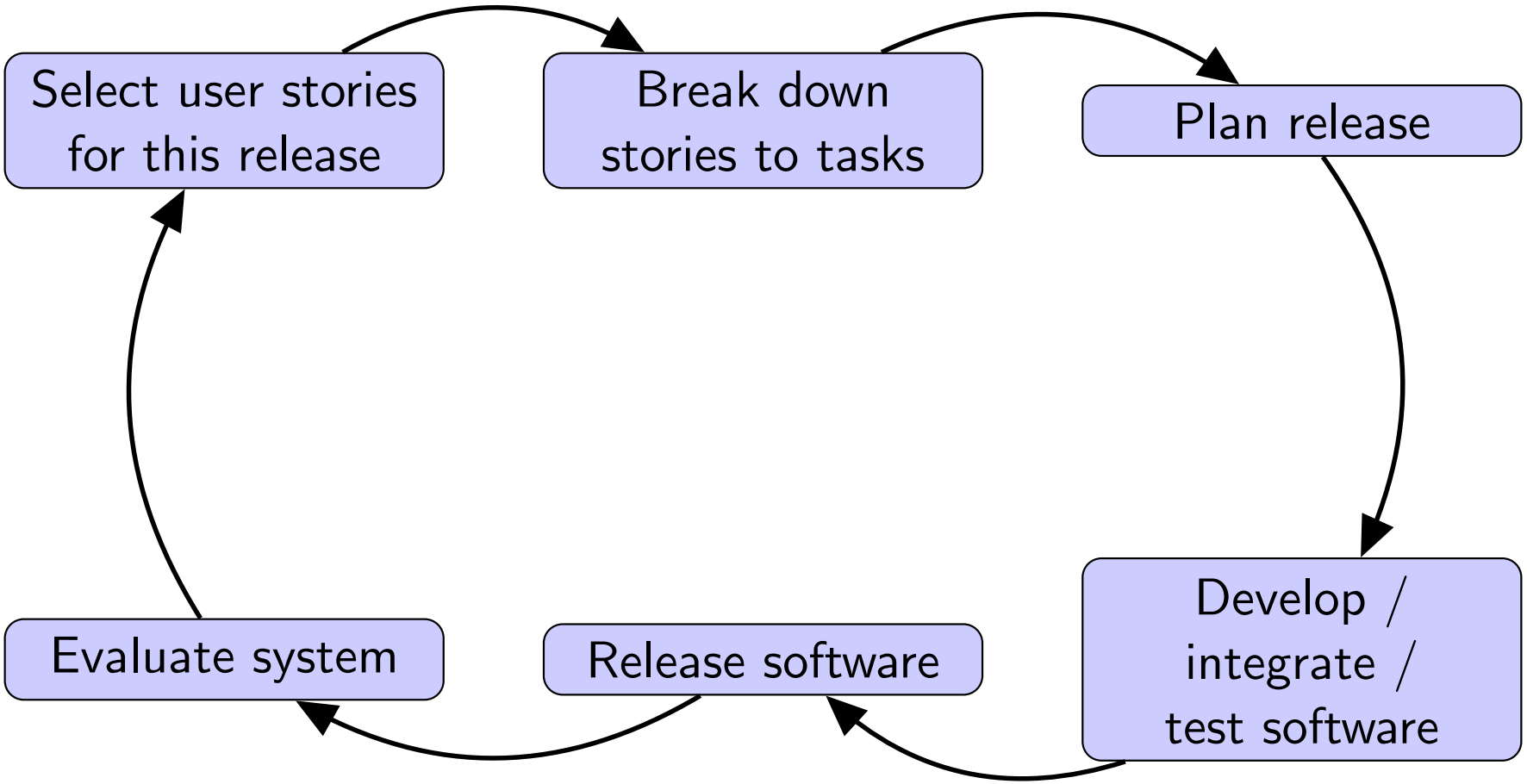
XP and agile principles

- "Incremental development"
  - small, frequent system releases
- "Customer involvement"
  - full-time customer engagement with the team
- "People not process"
  - pair programming
  - collective ownership
  - a process that avoids long working hours
- "Change support"
  - regular system releases

XP and agile principles

- "Incremental development"
  - small, frequent system releases
- "Customer involvement"
  - full-time customer engagement with the team
- "People not process"
  - pair programming
  - collective ownership
  - a process that avoids long working hours
- "Change support"
  - regular system releases
- "Maintaining simplicity"
  - constant refactoring of code

The extreme programming release cycle



Extreme programming principles

Incremental planning

Small releases

Simple design

Test-first development

Refactoring

Extreme programming principles

Incremental planning

- requirements are recorded on “user story” cards
- the stories to be included in a release are determined by the time available and their relative priority
- the stories are broken into development “Tasks”
- the stories for inclusion in the next release are chosen based on their priorities and the schedule estimates
- responsibility for making decisions on requirements: customer / user which is part of the XP team

Small releases

Simple design

Test-first development

Refactoring

Extreme programming principles

Incremental planning

Small releases

Simple design

Test-first development

Refactoring

Extreme programming principles

Incremental planning

Small releases

- start from the minimal useful set of functionality that provides business value
- frequent releases of the system
- incremental extension of the first release functionality

Simple design

Test-first development

Refactoring

Extreme programming principles

Incremental planning

Small releases

Simple design

- design carried out to meet the current requirements and no more

Test-first development

Refactoring

Extreme programming principles

Incremental planning

Small releases

Simple design

Test-first development

- before a new piece of functionality is implemented tests for it are written
- an automated unit test framework is used for that
- tests get the form of executable code (not the data to be read)

Refactoring

Extreme programming principles

Incremental planning

Small releases

Simple design

Test-first development

Refactoring

- code refactored as soon as possible
  after (code) improvements are found
- (quick) refactoring keeps the code simple
  and maintainable $\rightarrow$ the way of dealing with change
- possible changes to the code are cheap,
  to the architecture – more costly

Extreme programming principles

Pair programming

Collective ownership

Continuous integration

Sustainable pace

On-site customer

Extreme programming principles

Pair programming

Developers work in pairs (dynamically created)

- checking each other's work
- providing the support to always do a good job

Collective ownership

Continuous integration

Sustainable pace

On-site customer

Extreme programming principles

Pair programming

Collective ownership

Continuous integration

Sustainable pace

On-site customer

Extreme programming principles

Pair programming

Collective ownership

The pairs of developers work on all areas of the system

- no islands of expertise develop
- all the developers take responsibility for all of the code
- anyone can change anything

Continuous integration

Sustainable pace

On-site customer

Extreme programming principles

Pair programming

Collective ownership

Continuous integration

Directly after the work on a task is complete:

- integration of the results into the whole system
- (re)running all the unit tests in the system

Sustainable pace

On-site customer

Extreme programming principles

Pair programming

Collective ownership

Continuous integration

Sustainable pace

Reduction of overtime work

- protection against falling code quality and medium term productivity

On-site customer

Extreme programming principles

Pair programming

Collective ownership

Continuous integration

Sustainable pace

On-site customer

Regular membership of a representative of the end-user / customer in the XP team

- full time availability for the use of the XP team
- responsibility for bringing system requirements to the team for implementation
- involvement into the testing process (writing tests, validating results)

Crystal Clear

- Crystal Clear—7 principles
  1. Frequent Delivery
  2. Reflective Improvement
  3. Osmotic Communication
  4. Personal Safety
  5. Focus
  6. Easy Access to Expert Users
  7. Technical Environment with Automated Tests, Configuration Management and Frequent Integration

Crystal Clear

- Crystal Clear—7 principles
  1. Frequent Delivery
  2. Reflective Improvement
  3. Osmotic Communication
  4. Personal Safety
  5. Focus
  6. Easy Access to Expert Users
  7. Technical Environment with Automated Tests, Configuration Management and Frequent Integration

Crystal Clear

- Frequent Delivery
  - communication with a customer
- Reflective Improvement
  - planned meetings
- Osmotic Communication
  - minimize the hidden knowledge
  - plan of the office

Crystal Clear

- Personal Safety
  - atmosphere
- Focus
  - minimize the “context switching”
  - periods of uninterrupted work

Scrum

Scrum (rugby)

From Wikipedia, the free encyclopedia

For other uses, see Scrum.
"Scrummage" redirects here. For other uses, see Scrimmage (disambiguation).

A **scrum** (short for **scrummage**) is a method of restarting play in rugby that involves players packing closely together with their heads down and attempting to gain possession of the ball.[1] Depending on whether it is in rugby union or rugby league, the scrum is utilized either after an accidental infringement or when the ball has gone out of play

Luke Burgess (rightmost player in black) introduces the ball into the scrum.

Scrum

- Scrum—elements
  - events
  - team
  - artifacts
  - principles

Scrum

- Scrum—elements
  - events
  - team
  - artifacts
  - principles
- Three pillars:
  1. Transparency—all relative aspects of the process must be visible to those responsible for the outcome
  2. Inspection—frequent inspection of the artifacts and progress to identify and correct undesirable variances
  3. Adaptation—inspection leads to adjustments

Scrum

- Scrum team
  - Scrum Master
    - responsible for observing the rules and principles
  - Development Team
    - ca. 8 people, no less than 5
  - Product Owner
    - team customer
    - directs and prioritizes the development

Scrum Master protecting the team

Source: https://achardypm.medium.com/scrum-masters-protect-the-team-d99c35aa0fc9

Scrum Master protecting the team

As a Scrum Master, if you see any of the following, you definitely need to step in and speak up. Yes, even if they are more senior in the company than you:

- Product Owner insisting on giving the team estimates on how long he thinks the task will take
- Stakeholders insisting that functionality will be delivered by a certain date
- Stakeholders making decisions without consulting the team
- The teams reporting lines (developers reporting to one manager, QA reporting to another) telling the team they need to work on other priorities.

Source: `https://achardypm.medium.com/scrum-masters-protect-the-team-d99c35aa0fc9`

Scrum Master protecting the team

Scrum

- Events in Scrum:
  - Daily Scrum—a short meeting
    - what did I accomplish yesterday?
    - what will I do today?
    - what obstacles are impeding my progress?

Scrum

- Events in Scrum:
  - Daily Scrum—a short meeting
    - what did I accomplish yesterday?
    - what will I do today?
    - what obstacles are impeding my progress?
  - Sprint Planning
    - what is the scope of the sprint?

Scrum

- Events in Scrum:
  - Daily Scrum—a short meeting
    - what did I accomplish yesterday?
    - what will I do today?
    - what obstacles are impeding my progress?
  - Sprint Planning
    - what is the scope of the sprint?
  - Sprint Review
    - what have we accomplished?
    - have we managed to reach the sprint goal?

Scrum

- Events in Scrum:
  - Daily Scrum—a short meeting
    - what did I accomplish yesterday?
    - what will I do today?
    - what obstacles are impeding my progress?
  - Sprint Planning
    - what is the scope of the sprint?
  - Sprint Review
    - what have we accomplished?
    - have we managed to reach the sprint goal?
  - Sprint Retrospective
    - what is the condition of our scrum process?
    - what can be improved in our development process?

Scrum

- Artifacts
  - Product Backlog—requirements
  - Sprint Backlog
  - Burndown Charts
    - for a project
    - for a sprint

Scrum

- Restrictions
  - the meetings fit into time boxes
  - constant Sprint length—14-44 days
  - Sprint Goal cannot be modified
    - but... the sprint can be terminated

Kanban

- Kanban
  - methodology originally used in Toyota factories
  - “lean” approach—8 wastes:
    - wait
    - defect
    - transport
    - movement
    - excess inventory
    - excess production
    - unnecessary processing
    - unused talent

Kanban

- Kanban
  - methodology originally used in Toyota factories
  - “lean” approach—8 wastes:
    - wait
    - defect
    - transport
    - movement
    - excess inventory
    - excess production
    - unnecessary processing
    - unused talent
- Principles
  - workflow visualization
  - restricted WIP (work in progress)
  - the workflow is controlled
  - clear principles of the process
  - increased cooperation

Kanban

- Task pool
  - various stages
- Tasks assignment
  - retrieve from the pool, realize
  - the task assignment is self-managable
- Restricted number of tasks in progress
- Work continuity
- Scrum-ban

Agenda

Essential project roles

- General project roles
  - Users
  - sponsors
  - project manager
  - suppliers
  - team leaders
  - team members

Essential project roles

- General project roles
  - Users
  - sponsors
  - project manager
  - suppliers
  - team leaders
  - team members
- IT project roles
  - analyst
  - software architect
  - programmer
  - quality assurance engineer
  - graphic designer
  - user experience designer

Leader vs. manager

- Project manager
  - creates the plans
  - delegates the tasks
  - communicates with the team, users, and sponsors
  - is responsible for the project success

Leader vs. manager

- Project manager
  - creates the plans
  - delegates the tasks
  - communicates with the team, users, and sponsors
  - is responsible for the project success
- Team leader
  - reports to the project manager
  - has high soft skills

Team building

- Team building process
  - technical and psychological aspects
    - competencies
    - experience
    - culture
  - transparency
  - motivation—lead by example
  - define the success criteria

Team building

- Primary and secondary social groups

Team building

- Primary and secondary social groups
- Project team—a secondary group

Team building

- Primary and secondary social groups
- Project team—a secondary group
- Stages of team development (by Tuckman)
  - forming
  - storming
  - norming
  - performing
  - adjourning

Team building

- Motivating factors—categories
  - intrinsic and extrinsic
  - positive and negative
  - addressed to an individual and to a team
- Permanence of the motivating factors

Team building

- Motivating factors
  - financial rewards
  - non-financial rewards
  - appreciation
  - relationship—with team mates and leader(s)
  - position in a hierarchy
  - safety
  - personal development
  - meaningful work

Team building

- Motivating factors
  - financial rewards
  - non-financial rewards
  - appreciation
  - relationship—with team mates and leader(s)
  - position in a hierarchy
  - safety
  - personal development
  - meaningful work

Team building

- Analysis of the needs
- Maslow's hierarchy of needs

Team building

Team Role	Contribution	Allowable Weaknesses
Plant	Creative, imaginative, free-thinking. Generates ideas and solves difficult problems.	Ignores incidentals. Too preoccupied to communicate effectively.
Resource Investigator 	Outgoing, enthusiastic, communicative. Explores opportunities and develops contacts.	Over-optimistic. Loses interest once initial enthusiasm has passed.
Co-ordinator 	Mature, confident, identifies talent. Clarifies goals. Delegates effectively.	Can be seen as manipulative. Offloads own share of the work.
Shaper 	Challenging, dynamic, thrives on pressure. Has the drive and courage to overcome obstacles.	Prone to provocation. Offends people's feelings.
Monitor Evaluator 	Sober, strategic and discerning. Sees all options and judges accurately.	Lacks drive and ability to inspire others. Can be overly critical.
Teamworker 	Co-operative, perceptive and diplomatic. Listens and	Indecisive in crunch situations.

Team building

Teamworker 	Co-operative, perceptive and diplomatic. Listens and averts friction.	Indecisive in crunch situations. Avoids confrontation.
Implementer 	Practical, reliable, efficient. Turns ideas into actions and organises work that needs to be done.	Somewhat inflexible. Slow to respond to new possibilities.
Completer Finisher 	Painstaking, conscientious, anxious. Searches out errors. Polishes and perfects.	Inclined to worry unduly. Reluctant to delegate.
Specialist 	Single-minded, self-starting, dedicated. Provides knowledge and skills in rare supply.	Contributes only on a narrow front. Dwells on technicalities.

Team building

- Other team roles
  - scapegoat
  - outsider
  - aggressor / rioter
  - devil's advocate
  - perfectionist

Team building

- Other team roles
  - scapegoat
  - outsider
  - aggressor / rioter
  - devil's advocate
  - perfectionist
- Leadership types
  - autocratic
  - democratic
  - Laissez-Faire

Team building

- Communication in the team
  - integration, workshops
  - brainstorming sessions
  - conveying the messages
    - avoiding information cliques
    - feedback
  - transparent assessment
    - clear rules and principles
    - 360-degree feedback
  - verbal and written communication

Agenda

Software cost estimation

- Software cost estimation—the cost of completing a project

Software cost estimation

- Software cost estimation—the cost of completing a project
- Cost types:
  - direct costs—cost of project-specific tasks
  - indirect costs—shared across multiple projects (operational costs of an organization)

Software cost estimation

- Software cost estimation—the cost of completing a project
- Cost types:
  - direct costs—cost of project-specific tasks
  - indirect costs—shared across multiple projects (operational costs of an organization)
- Cost components:
  - labor
  - resources: materials, hardware, equipment, services, facilities, etc.
  - other costs: licences, certification, risk management, etc.

Software cost estimation

- Why is cost estimation important?
  - important for sponsors
  - fixed-price projects
  - incorrect estimation—reason for many project failures

Software cost estimation

- Why is cost estimation important?
  - important for sponsors
  - fixed-price projects
  - incorrect estimation—reason for many project failures

Software cost estimation

- Why is that a challenge?
  - usually software projects differ much between each other
  - it is hard to estimate something that has never been done
  - software is complex and many difficulties cannot be predicted
  - requirements are seldom fixed at the beginning
  - performance of programmers depends on many factors (including motivation)—difficult to predict
  - time and cost do not scale linearly with the team size

Software cost estimation

- Why is that a challenge?
  - usually software projects differ much between each other
  - it is hard to estimate something that has never been done
  - software is complex and many difficulties cannot be predicted
  - requirements are seldom fixed at the beginning
  - performance of programmers depends on many factors (including motivation)—difficult to predict
  - time and cost do not scale linearly with the team size

Software cost estimation

- Why is that a challenge?
  - usually software projects differ much between each other
  - it is hard to estimate something that has never been done
  - software is complex and many difficulties cannot be predicted
  - requirements are seldom fixed at the beginning
  - performance of programmers depends on many factors (including motivation)—difficult to predict
  - time and cost do not scale linearly with the team size

Software cost estimation

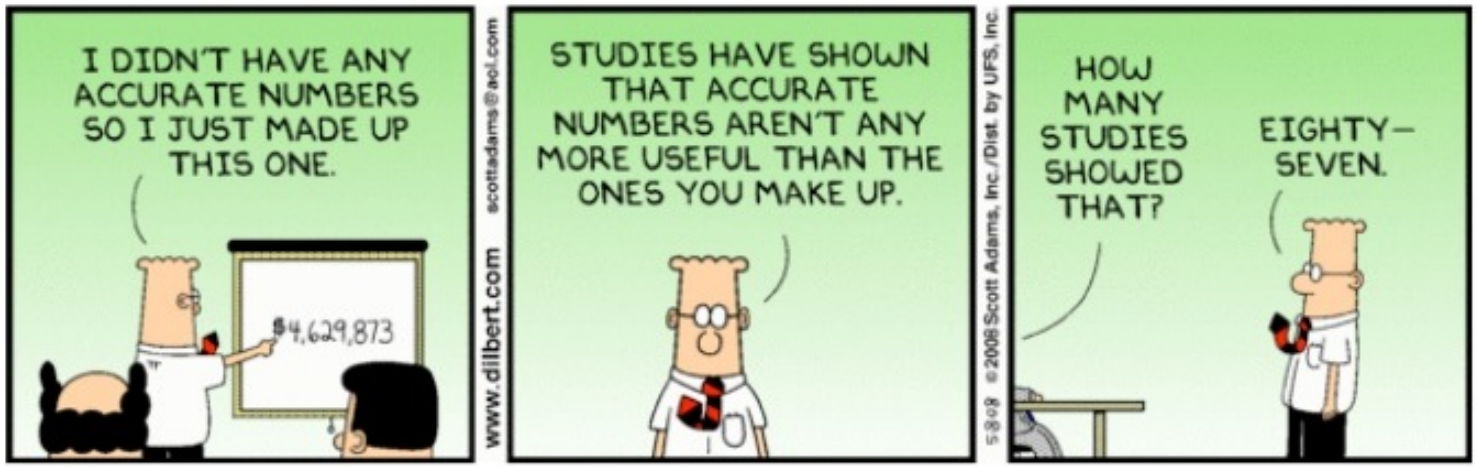
- Software cost estimation—approaches
  - algorithmic methods
  - non-algorithmic methods
  - combined methods
- Types of costs: fixed, time-dependent, work-dependent

Software cost estimation

- Algorithmic methods
  - algorithmic models (e.g., COCOMO, Function Points)
  - basis for describing the project: many numeric or descriptive attributes (e.g., lines of code)
  - result obtained by applying an adequate algorithm / mathematic formula
  - models (and their parameters) based statistical surveys (may involve machine learning)
    - benchmark databases of projects

Software cost estimation

- Algorithmic methods
  - algorithmic models (e.g., COCOMO, Function Points)
  - basis for describing the project: many numeric or descriptive attributes (e.g., lines of code)
  - result obtained by applying an adequate algorithm / mathematic formula
  - models (and their parameters) based statistical surveys (may involve machine learning)
    - benchmark databases of projects



Software cost estimation

- Non-algorithmic methods
  - expert evaluation—experience as a basis
  - brainstorming
  - estimation by analogy—similarities and differences with past projects
  - estimation to win—cost expected by the customer, costs declared by the competitors

Software cost estimation

Software Engineering

Management of IT projects

Silesian University of Technology