CSE 3231/5231 Software Engineering

Fundamentals of Software Engineering Processes

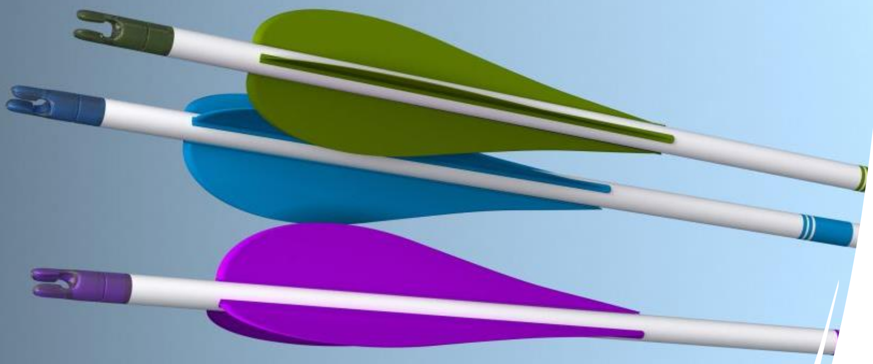
Presented by:

Praveen Kumar

Learning Outcomes

- Be familiar with software engineering process concepts and benefits
- Be familiar with plan-driven and agile/iterative software engineering processes, their differences, and suitability
- Be familiar with the principles and best practices behind software engineering practices
- Be familiar with *designing* software engineering processes

WHAT ARE SOFTWARE ENGINEERING PROCESSES?



Why do we need a process?

The goal is the software PRODUCT. Processes only ADD cost!

We follow processes to ***reduce risks***:

- ❑ Not meeting customer expectations (FR / NFR)
- ❑ Costs
- ❑ Predictability

Why follow a process?

- Repeatability
- Predictability
- Traceability
- Improved quality through standardization
- Continuous improvement
- Enables training
- Builds confidence

What is a software process?

A well-designed set of **partially ordered steps**
intended **to reach a goal**;
in software engineering
the goal is to build
a software product
or enhance an existing one.

How do you design a process?

By understanding:

- Software lifecycle phases
- Software lifecycle stages
- Characteristics of software projects
- Best practices in software processes
- Best practices within software development phases (later lectures)
- And applying a software process design FRAMEWORK!

Augh! Not another framework?!

Software Engineering Phases

PHASES OF THE SOFTWARE LIFECYCLE

Class Discussion

- What are various “types” of activities you performed for creating a new software?
  - Internship/Work experience
  - Team Project
  - Hackathon

What are the phases of the software lifecycle?

Phases are tasks grouped by common intent

1. Requirements identification
2. Analysis
3. Architecture and Design
4. Implementation
5. Testing
6. Deployment
7. Maintenance
8. Project Management

What is requirements identification?

The phase where you identify **the problem to be solved, detail the features of the solution, the business case** and **the acceptance criteria**.

What is analysis?

The phase where you come to
understand the domain (domain analysis),
understand the problem (problem analysis),
and
understand the solution (solution analysis).

What is architecture and design?

The phase where you determine

the overall structure, or architecture,

of the system, what the components of the system are, and how they fit together

What is implementation?

The phase where you actually **build the system**.

What is testing?

The phase where you check if the system **actually works!**

What is deployment?

The phase where you actually
put the system to work.

What is maintenance?

The phase where you
keep the system working
and **keep it useful** as needs evolve

What is maintenance?

The phase where you

keep the system working

and **keep it useful** as needs evolve

What is Project Management?

The phase where you plan, organize, resource, lead, control and coordinate

A note about phases

Software lifecycle phases are
groupings of activities
around a common intent.

Phases are not steps done in order!

If phases aren’t ordered, what determines the process order?

Software engineering steps
are **ordered in time** by **stages**.

Software Engineering Stages

STAGES OF A SOFTWARE LIFECYCLE

What are the stages of a software lifecycle?

- Inception
- Elaboration
- Construction
- Transition

Progression from high-level understanding and low certainty to ***detailed understanding*** and ***high certainty***

What is the inception stage?

The stage where you create
an approximate vision,
a business case,
scope,
a high-level, **potential architecture**
and **high-level estimates**.

What is the elaboration stage?

The stage where you

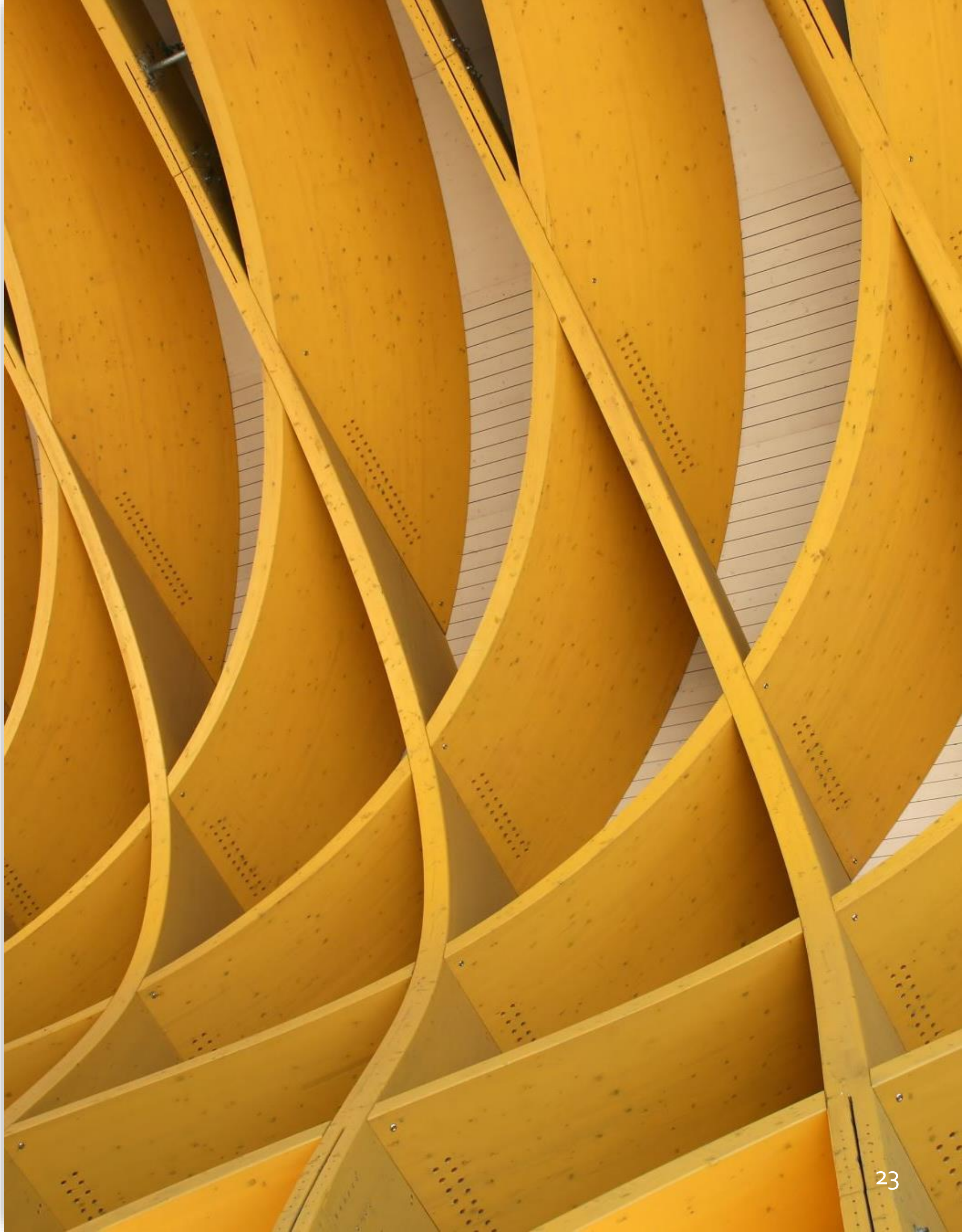
refine the vision,

validate the core architecture,

resolve high risks,

do most of the **requirement's identification**, and

create more realistic estimates.



What is the construction stage?

The stage where you

iteratively implement any remaining features

and

prepare for deployment.

What is the transition stage?

The stage where you
deploy a finished released.

Key Approaches

SOFTWARE ENGINEERING METHODOLOGIES

Progression varies because of differences in projects

Size and complexity

Personnel composition, capability, culture, proximity, uniformity, stability

Clarity of goals

Dynamism of goals

Criticality of purpose

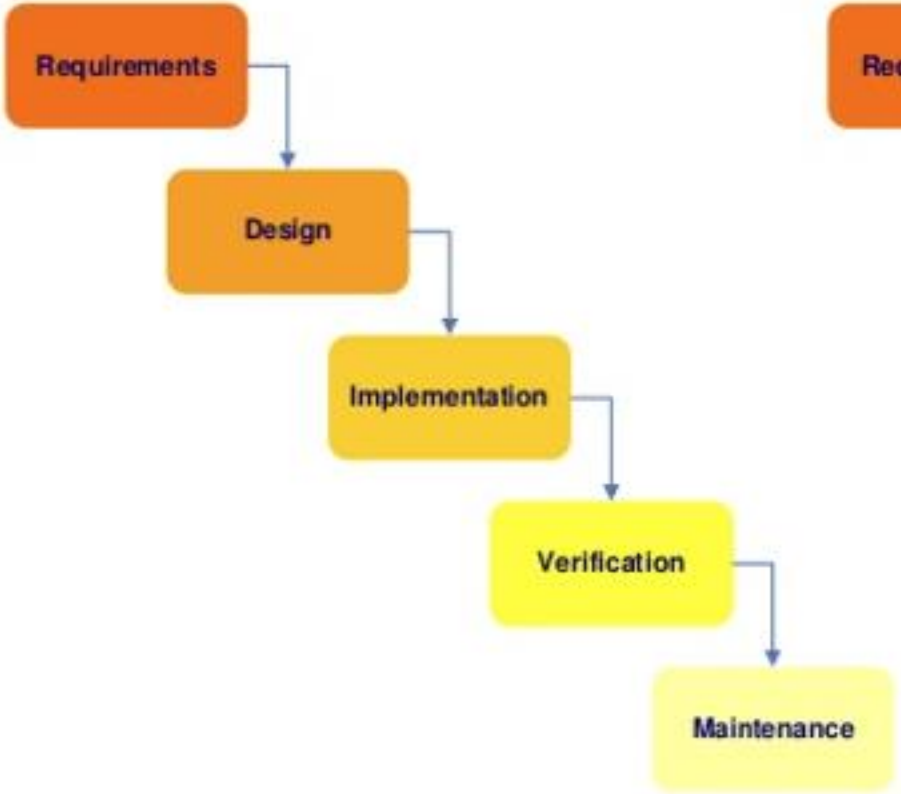
Expectations

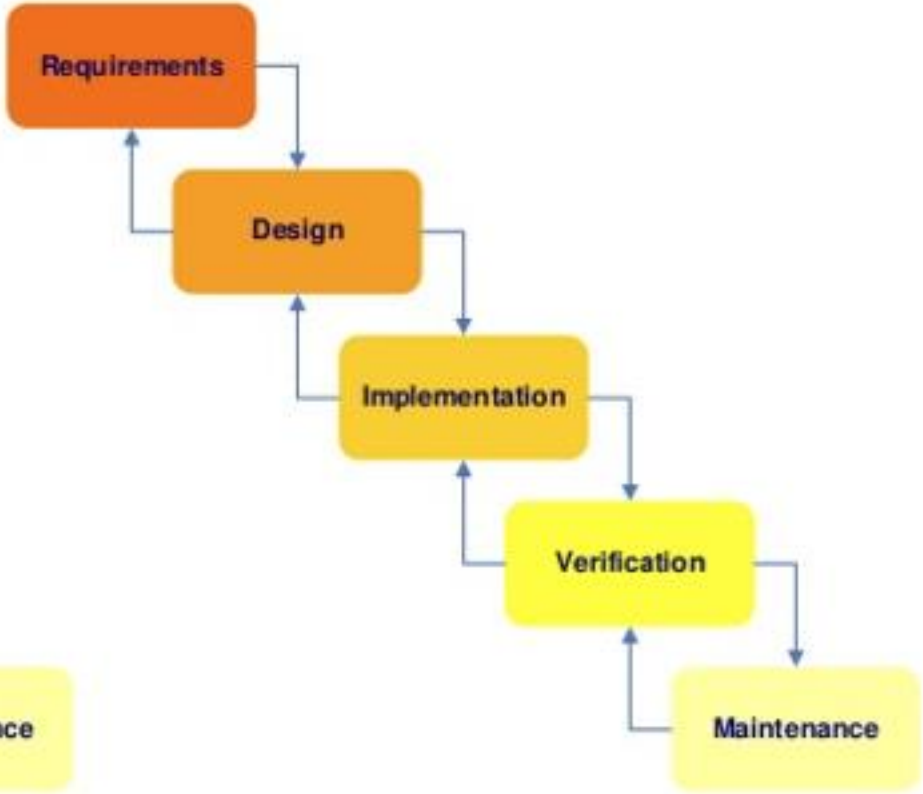
Waterfall / Plan Driven

Waterfall / Plan-Driven

1. Waterfall is **sequential** in nature: the project moves to the next phase only after the current phase is complete
2. Waterfall is good when the **scope is very well know**, and deadlines and budget are less important
3. Waterfall leverages **specialized skills**: designers at design phase, engineers at development phase
4. Used more in the physical world (e.g. Space X rockets), used less in abstract world (e.g. software)

Department of Defense: 5 Phase Model





Agile / Iterative

Agile / Iteration-driven

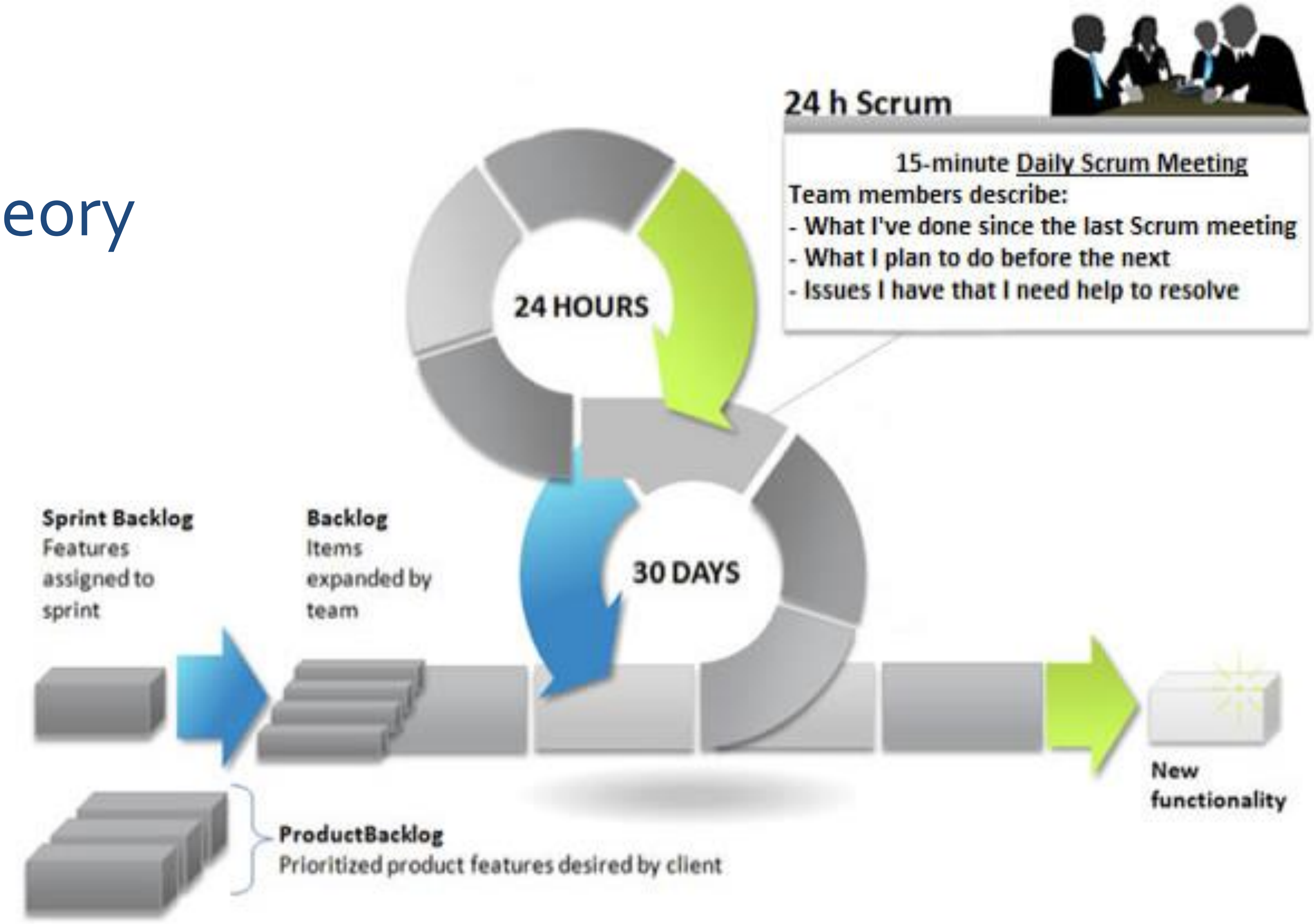
We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

1. Individuals and interactions *over* processes and tools
2. Working software *over* comprehensive documentation
3. Customer collaboration *over* contract negotiation
4. Responding to change *over* following a plan

Scrum

Scrum - Theory



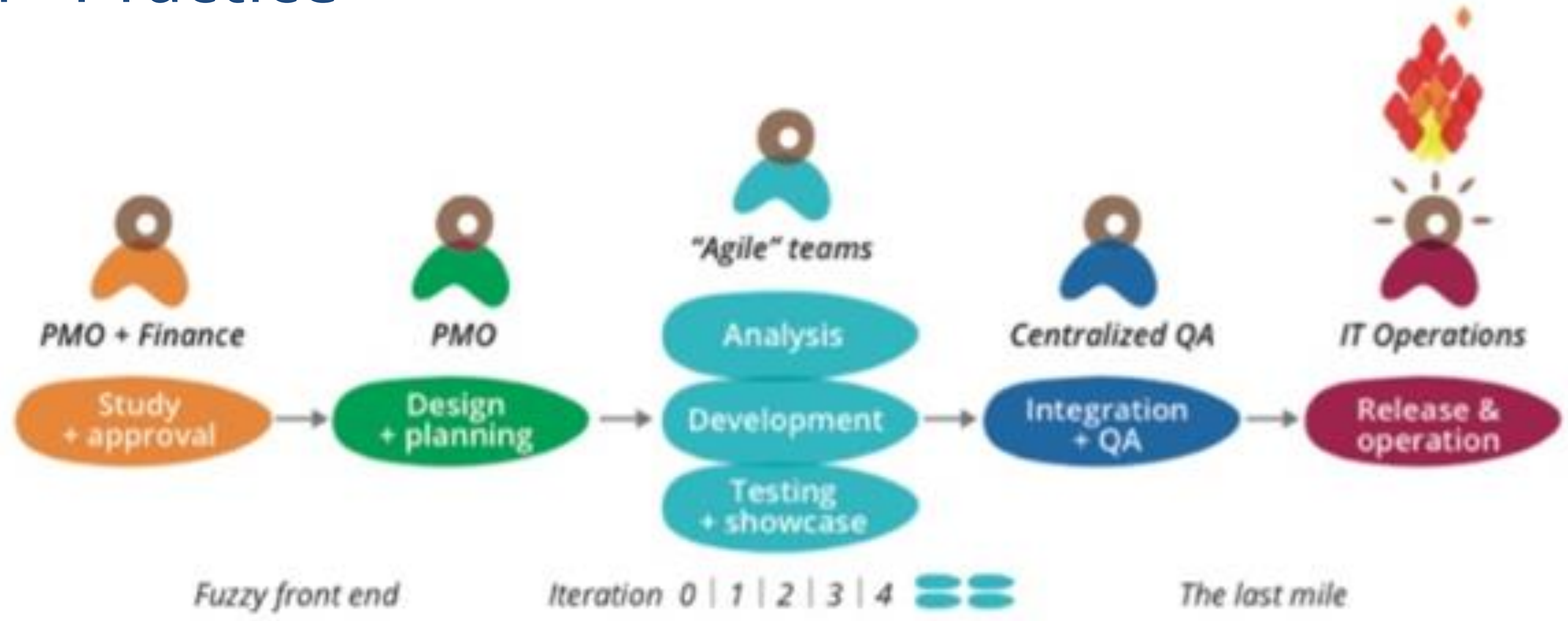
24 h Scrum

15-minute Daily Scrum Meeting

Team members describe:

- What I've done since the last Scrum meeting
- What I plan to do before the next
- Issues I have that I need help to resolve

Scrum - Practice



Scrum

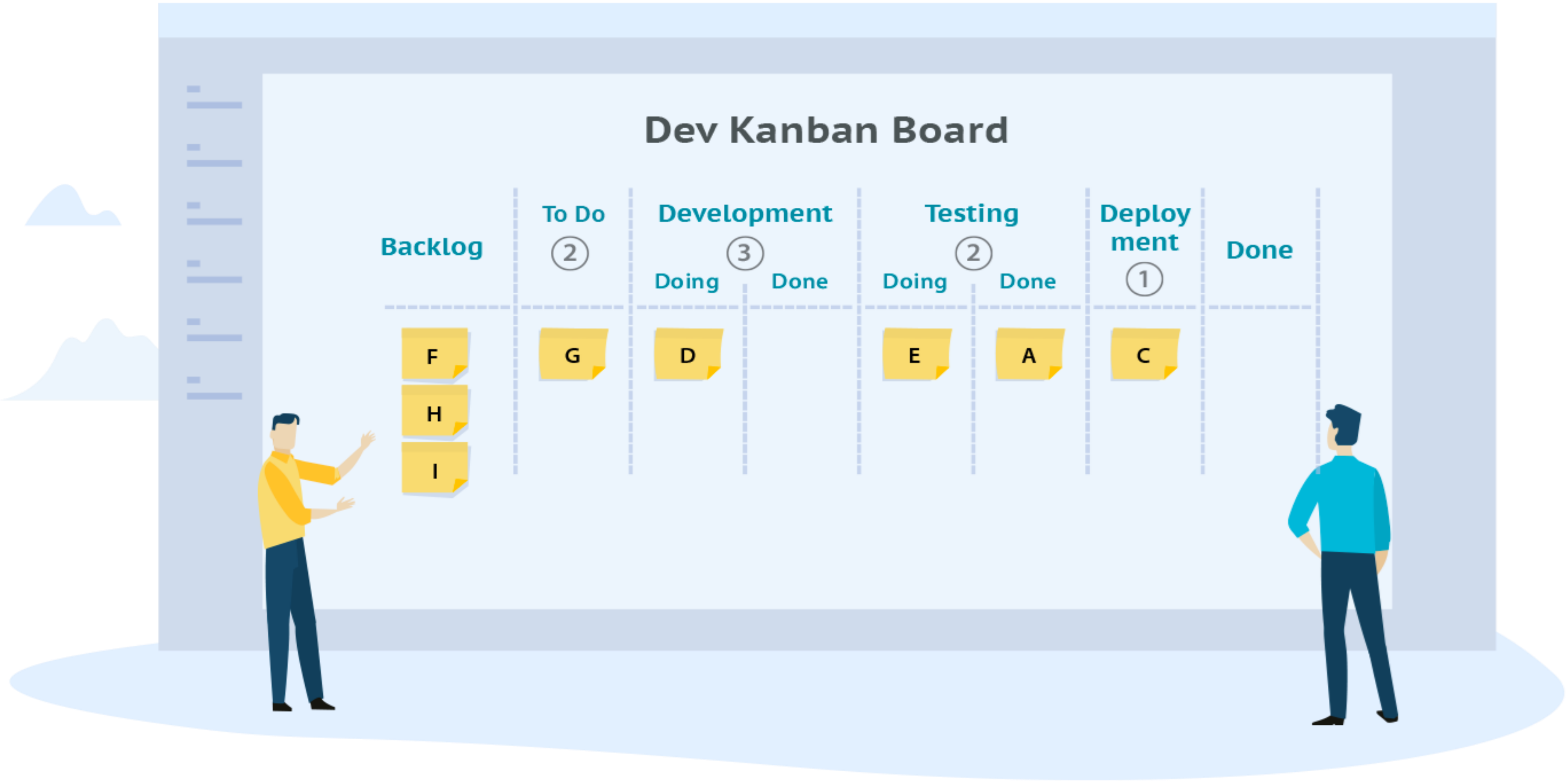
1. Fairly prescriptive with rules and procedures to follow
2. Scrum is **iterative** in nature; small units of functionality delivered **2-4 weeks**
3. Embraces that **scope changes** from sprint to sprint- but not within a sprint
4. Teams are cross-functional with **generalized** skills
5. The most common of agile practices, but most organizations are “Water-Scrummer-Fall”

Kanban

Kanban

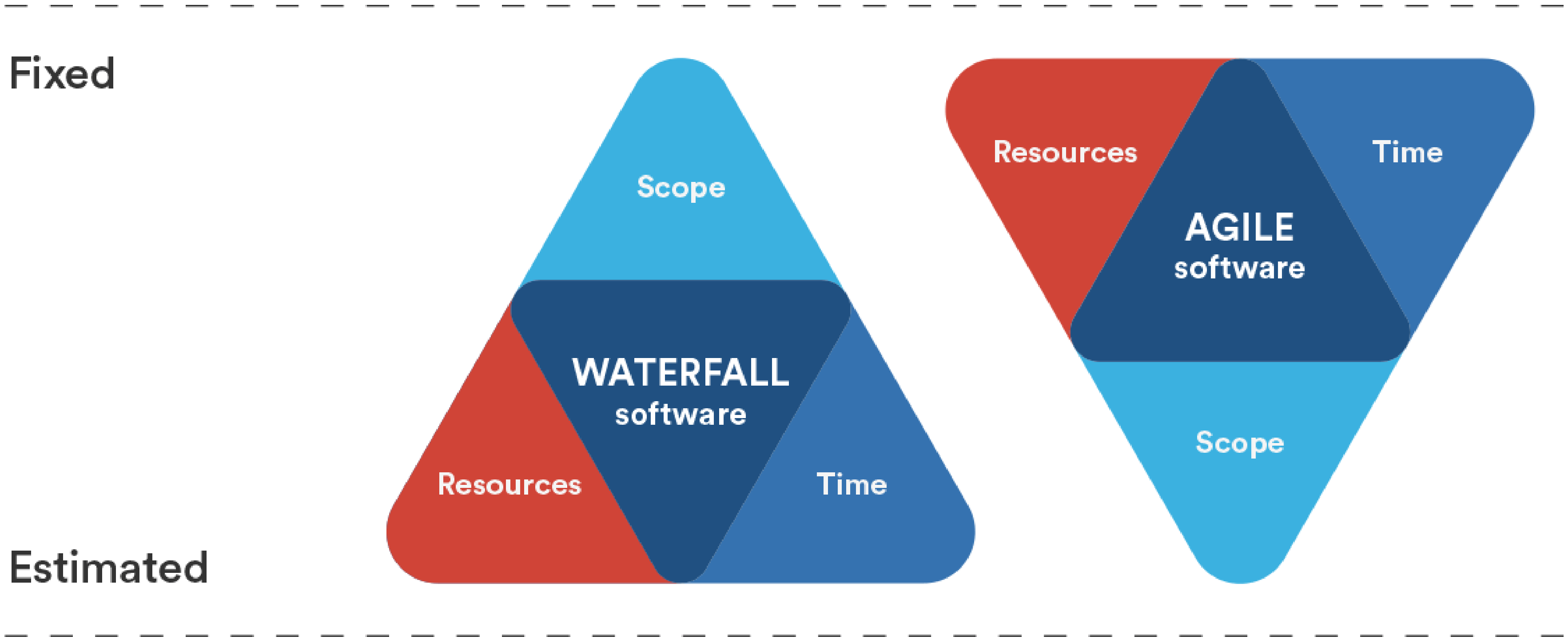
1. Phases are modeled as swimlanes, each lane **pulls** work from the top of the previous lane
2. Kanban is **iterative** in nature, and functionality can be delivered in hours, days, weeks
3. Embraces that **scope changes**, especially when constantly changing and using experimentation
4. Balance between **specialization** within a lane, **generalization** across the board
5. Measures lead time, in-process limits, flow rate, defect rate

Kanban


Dev Kanban Board
Backlog
To Do
2
Development
3
Doing
Done
Testing
2
Doing
Done
Deploy ment
1
Done
F
H
I
G
D
E
A
C

How to select which methodology?

Agile vs Waterfall



How to select agile vs. plan driven?

- Step 1 – Break overall system into “releasable parts”
- Step 2 – For each part, use spider diagram (next slide) to determine process
- Step 3 – Evaluate risks to tailor process (pure-agile or pure-plan-driven may not work)

Balancing agility & discipline

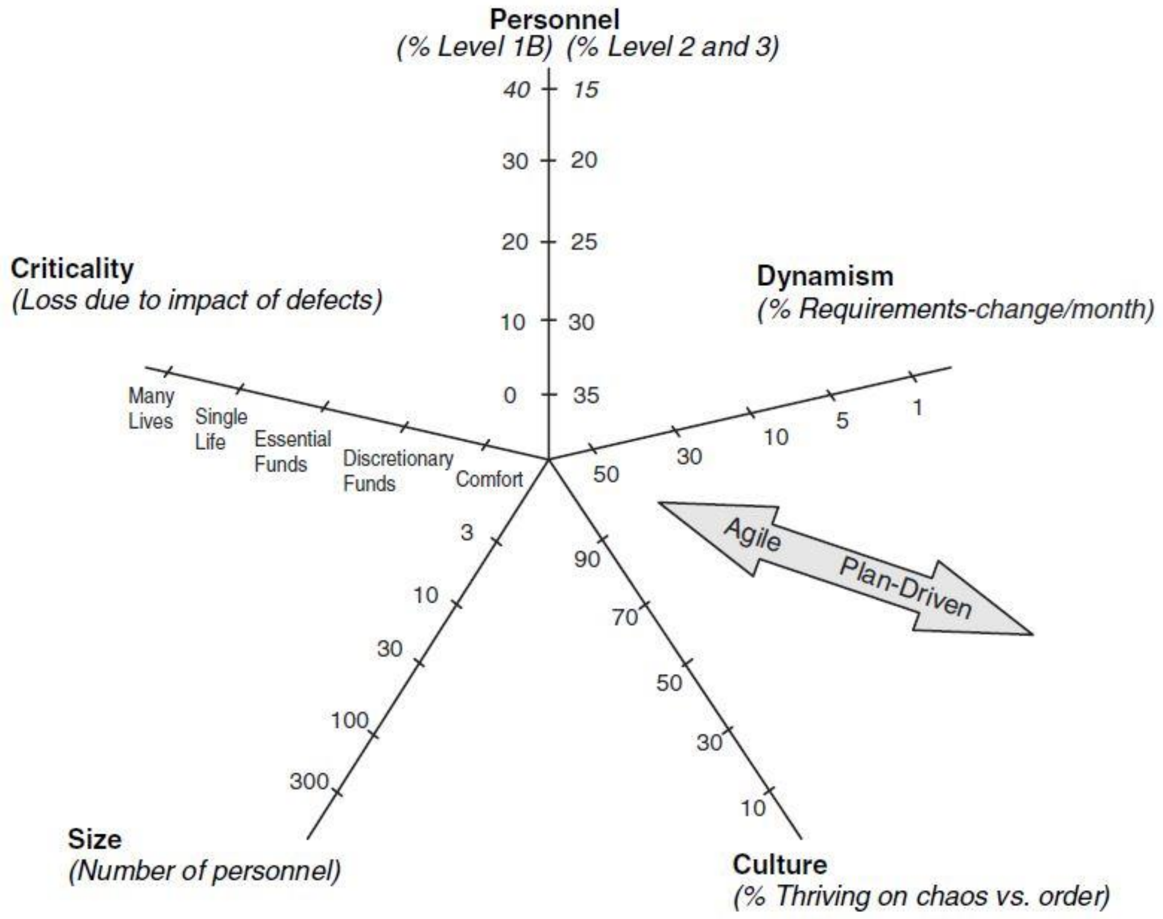


Figure 2-2 Dimensions Affecting Method Selection

Balancing agility & discipline

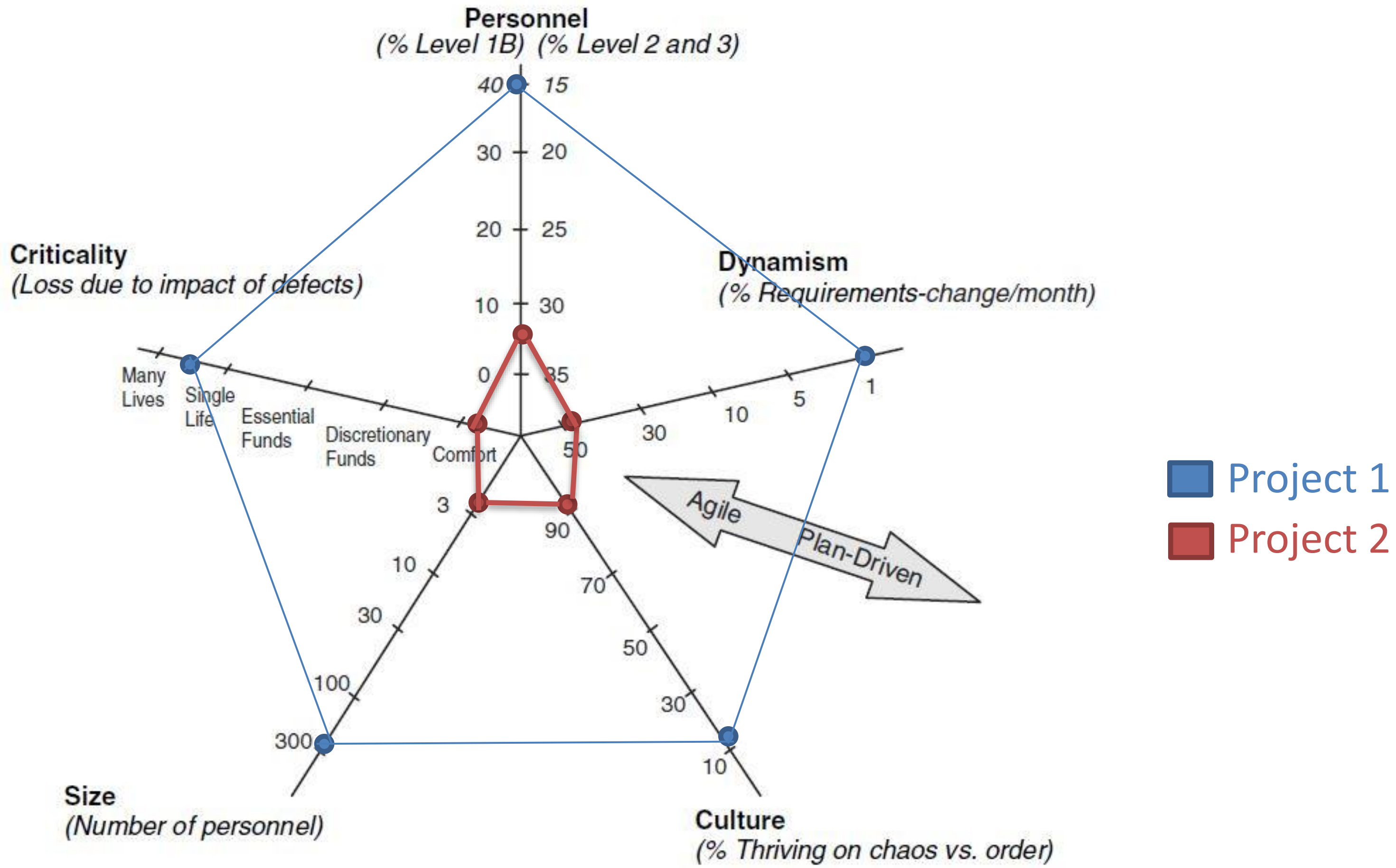


Figure 2-2 Dimensions Affecting Method Selection

Practice the concepts using

CLASS EXERCISES

Class Exercise 1 - LMS in a college

Learning Management System, something very similar to Carmen

- 30 developers, all young, inexperienced
- 15 large features
- Thrive on learning, thrive on chaos
- Not a mission critical project
- Requirements will change – about 50%

Class Exercise 2 - Credit Card Processing System for Ohio Savings Bank

- Team of 6 to develop a solution
- 5 of them with 20+ years of experience
- Mission critical software for the Bank
- It is a rewrite of an existing system
- Overall task: 1 web app with complex business logic, 6 APIs, 1 Relational database

(hint: ***estimate*** requirement volatility & other factors)

<Do not see, instructor will walk you through>

SOLUTIONS

Class Exercise #1

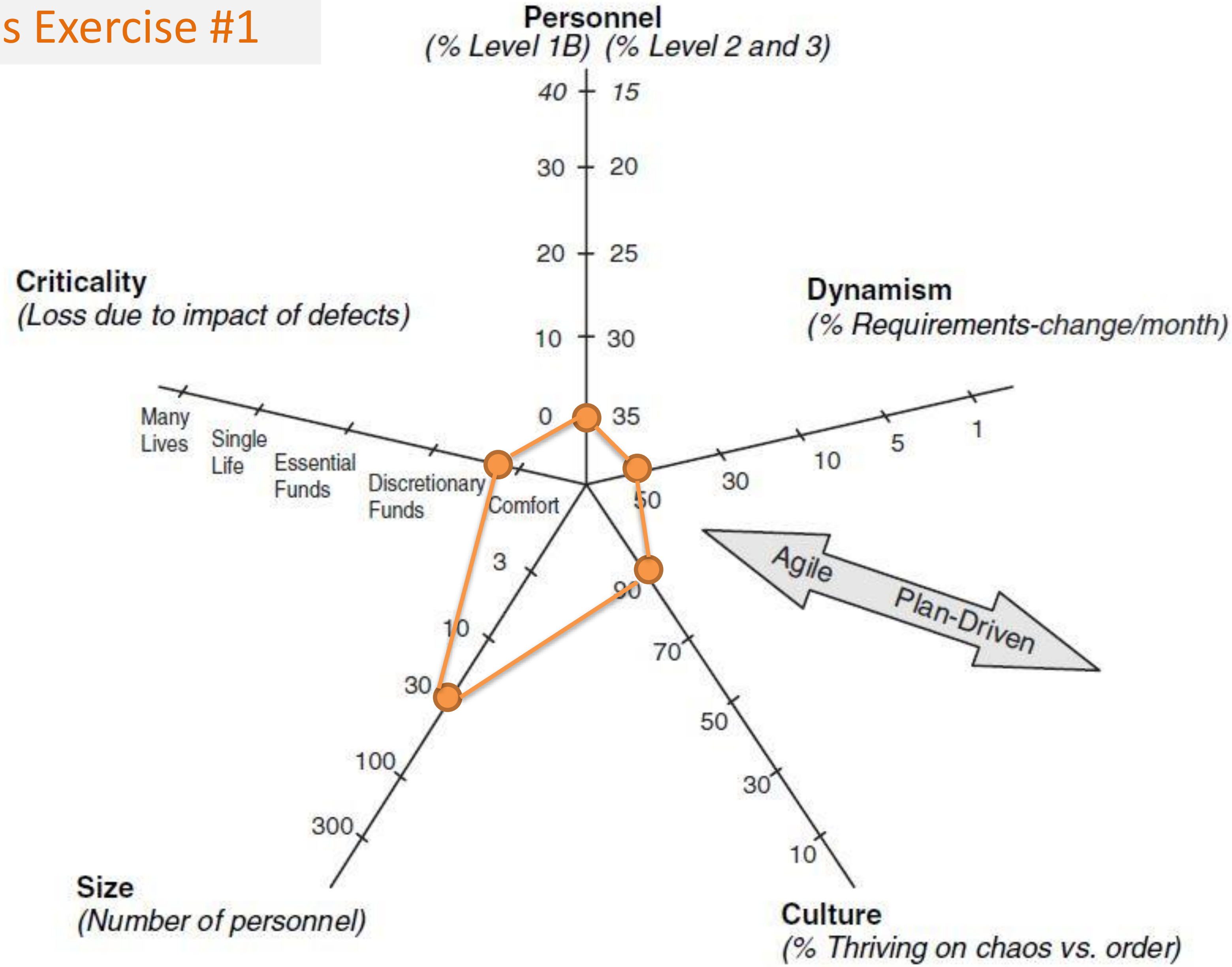


Figure 2-2 Dimensions Affecting Method Selection

Class Exercise #2

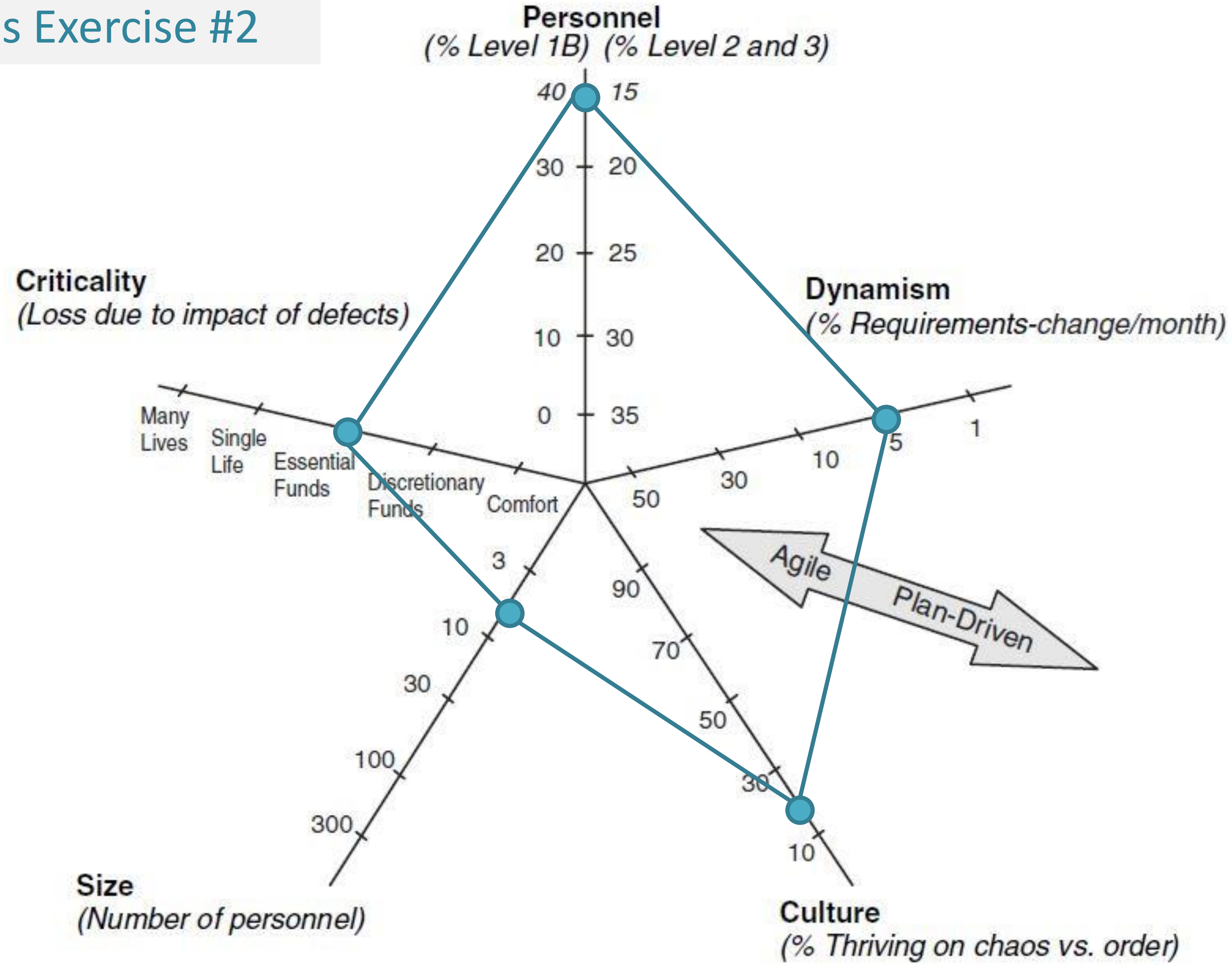


Figure 2-2 Dimensions Affecting Method Selection

BEST PRACTICES IN SOFTWARE ENGINEERING PROCESSES

Best Practices

- Incremental Development
  - ✓Software Component Incremental
  - ✓Iterative (features incremental)
  - ✓Scenario-driven (use cases incremental)
- Separation of concerns

Addressing the Range of Separable Concerns – Requirements

1. Requirements
   - Problem (why – from a functional point of view)
   - Business case (why – from the business point of view)
   - User perspectives (as the user sees the system)
   - Visible function (what the system is supposed to do)
   - Non-functional requirements (how the system is supposed to do it)
   - Prioritized requirements (when)
   - Acceptance criteria (validation by the customer)

Addressing the Range of Separable Concerns – Analysis

Analysis:

- Static structure
- Dynamic Behavior
- Done at multiple levels – at least: Domain, Problem and Solution

Addressing the Range of Separable Concerns – Architecture and Design

- System architecture
- Target environment
- Subsystem / Component model
- The static structure of the system components
- The dynamic behavior of the system components

Addressing the Range of Separable Concerns – Project Management,

1. Project management:
   - Process design
   - Resources
   - Roles
   - Schedule
   - Release and iteration design
   - Risk management

Addressing the Range of Separable Concerns –Implementation and Testing

Implementation and Testing:

- Code management
- Unit validation
- Component validation
- System validation

Other Key Considerations

- Verification – doing the thing right
- Validation – doing the right thing
- Traceability – forward and backward

Principles for teamwork

- Communication and visibility
- Feedback
- Keeping “inventory” small
- Courage and respect
- Sustainable work

Principles: Risk-driven management

- Expectations
    - ✓Handled by software development process
- Unexpected risks
    - ✓Risk planning

<Switch>

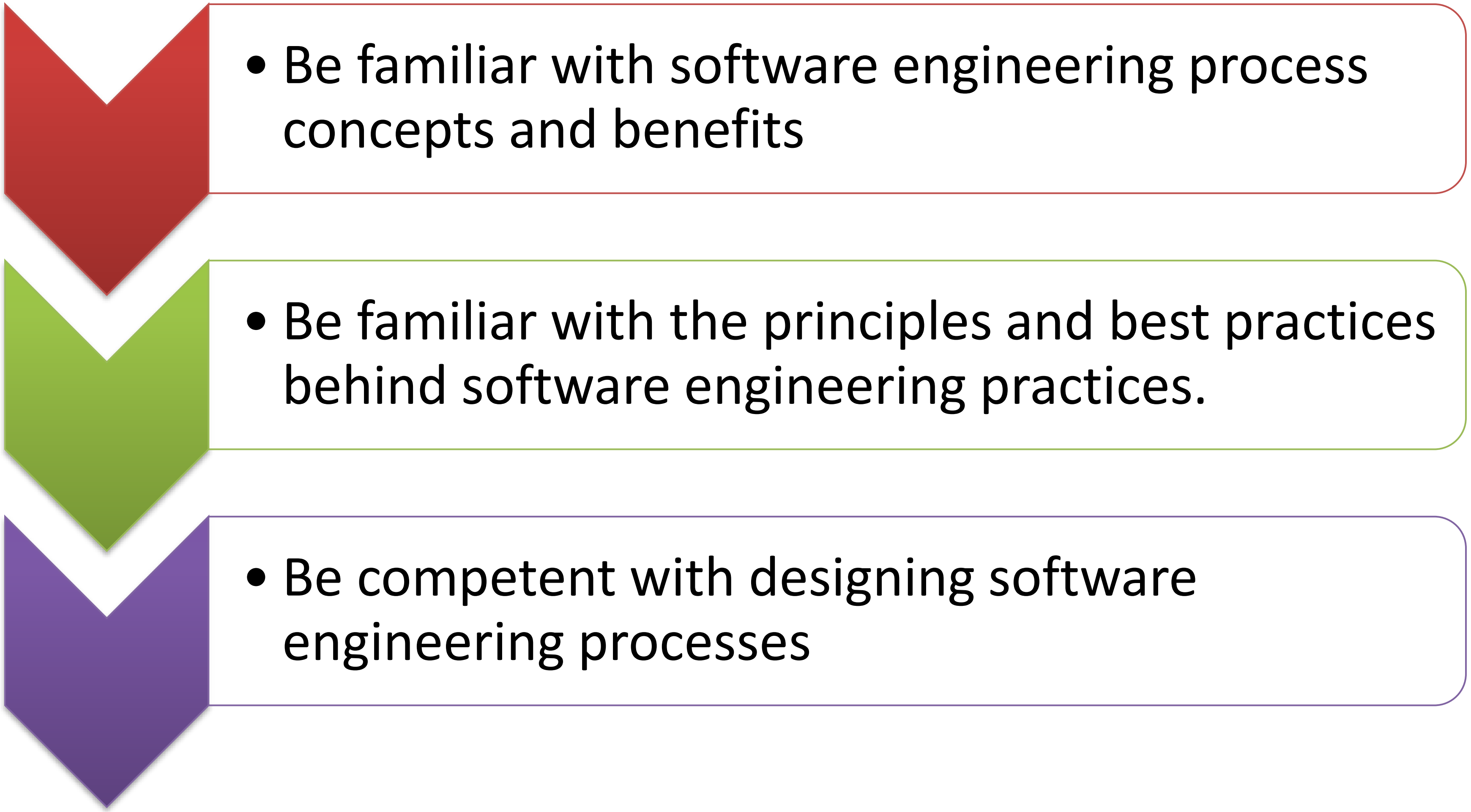
SDLC APPROACHES & SCRUM

CUSTOMIZING SOFTWARE ENGINEERING PROCESSES

Risks in Agile vs. Plan-Driven Processes

- Environmental risk (risks that result from the project's general environment) that apply to both kinds of processes:
  - The technology to be used has many uncertainties
  - Many diverse stakeholders need to be coordinated
  - Complex system of systems
- Agile risks: risks that are specific to the use of agile methods
  - Scalability needs and criticality are high
  - Use of simple design might not work
  - There is too much personnel turnover or churn
  - There are not enough people skilled in agile methods
- Structured-driven risks: risks that are specific to the use of structured methods
  - There is rapid change, making long-term planning useless
  - Need for rapid results
  - There are emergent requirements
  - There are not enough people skilled in plan-driven methods

What have we learned



- Be familiar with software engineering process concepts and benefits
- Be familiar with the principles and best practices behind software engineering practices.
- Be competent with designing software engineering processes

References

- Developing Object-Oriented Software – An Experience-Based Approach (online):
  - Chapters 1, 2, 3, 4 – REQUIRED
  - Chapters 5, 6, 7 – needed for project
- Balancing Agility and Discipline (online)
  - Extracts - REQUIRED

Image Credits

Placeholder

“Title of picture,” by Author Name, cc licensed (BY NC SA): http://flic.kr/p/8VkRnd