University of Hertfordshire UH

Project management – Lecture notes

7COM1085 – Research Methods

Dr. John Noll

University of Hertfordshire

Introduction

Learning Objectives

The aims of this lecture are to

1. understand what project *management* comprises;
2. know the elements of a useful (research) project *plan*;
3. distinguish *research* project management from other kinds of projects.

Definitions

Project A set of activities that ends with a specific achievement.

Characteristics:

- Non-routine.
- Distinct start.
- Distinct finish.

Definitions, cont.

Plan A set of tasks that lead to achievement of the project's objectives, comprising the W^5H^2m:

- Who.
- What.
- Why.
- Where.
- When.
- How.
- How much.

Plan: Definition, cont.

Specifically a plan comprises:

1. Deliverables (*What*).
2. Tasks (*How*).
3. Milestones (*When*).

Plan *execution* also determines:

- Responsible parties (*Who*).
- *Where* in the organization responsible parties exist (including location).
- Estimates (*How much*).

The plan's *objectives* should be justified by the *Why*: how the stakeholders will benefit from the project's objectives.

Definitions, cont

Project management what project managers do:

- Create plans.
- Assign tasks.
- Monitor progress.
- *Make decisions.*

Planning

Planning

"Plans are worthless but planning is everything." – General Dwight Eisenhower.

Planning is easy; execution is hard, because change happens and is inevitable.

Some Definitions

Deliverable	A tangible outcome. If a task doesn't have a deliverable, why do it?
Task	An activity that requires effort and produces one or more *Deliverables*.
Milestone	A date in the project schedule at which something *significant* happens, typically when a task's deliverable is ready.

Planning Process

1. Create a “work breakdown structure” of deliverables. (*What*.)
2. Specify tasks. (*How*.)* In research projects, *How* is the research method used to produce deliverables.
3. Estimate effort required. (*How much*.)
4. Derive milestones, based on effort. (*When*.)
5. Assign responsible parties and locations. (*Who* and *Where*.)

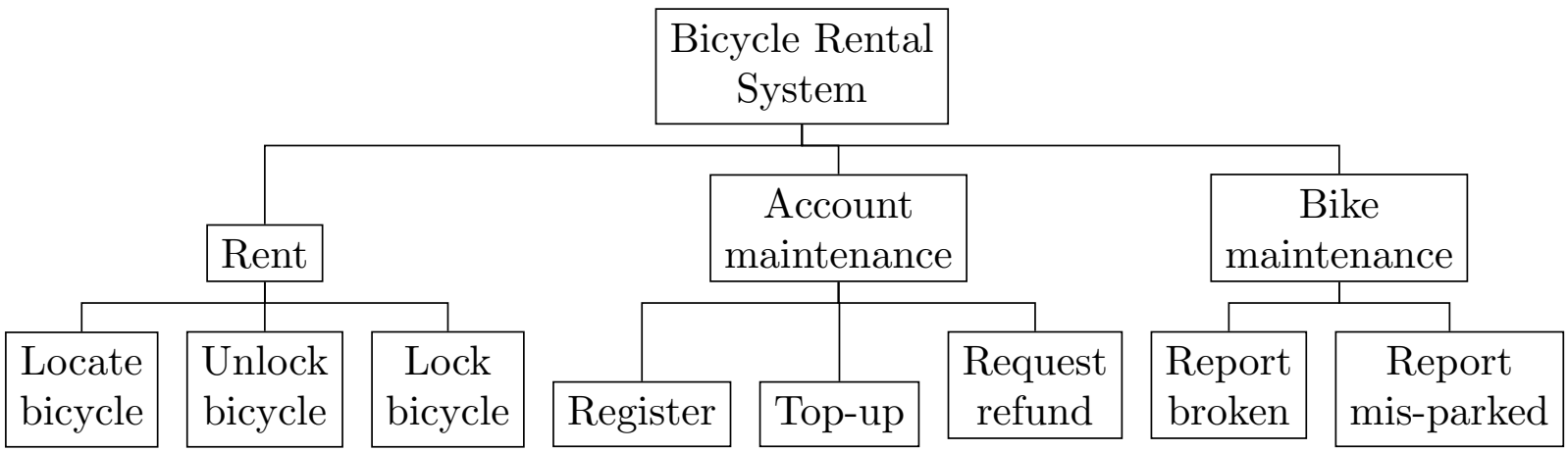
What about *Why*?

- *Why* is part of the motivation for the project: your knowledge of the literature should reveal gaps to be filled.

Work Breakdown Structure

Just what it says: the result of systematically breaking down the plan's objective into successively finer deliverables

Work Breakdown Structure (WBS): Example



Plan from WBS

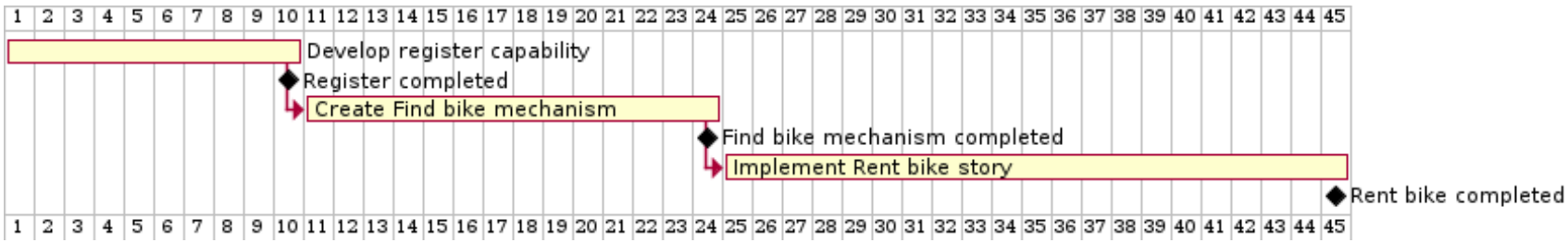
Task (verbs)	Deliverable (nouns)	Milestone
Develop Register capability	Register *capability*	?
Create Find bike mechanism	Find bike *feature*	?
Implement Rent bike story	Rent bike *story*	?

Why no Milestones?

- Milestones are dates.
- Dates depend on *estimates*; at this stage we haven't any.

Milestones: Gantt chart example

When we have estimates, we can forecast dates (milestones):



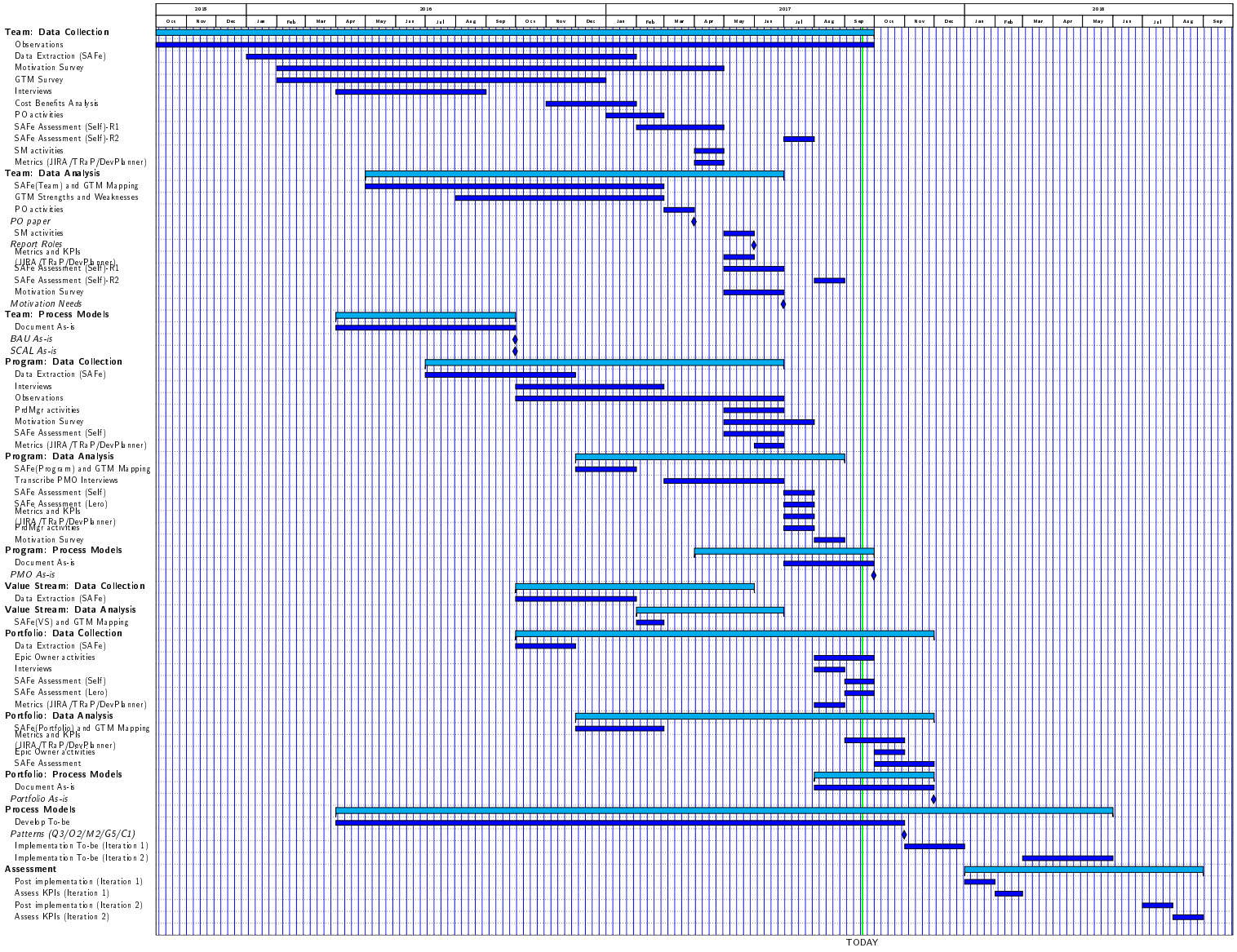
Notes:

This simple schedule assumes tasks are done sequentially. This is the case if the estimates are based on 100% staff allocation to each task.

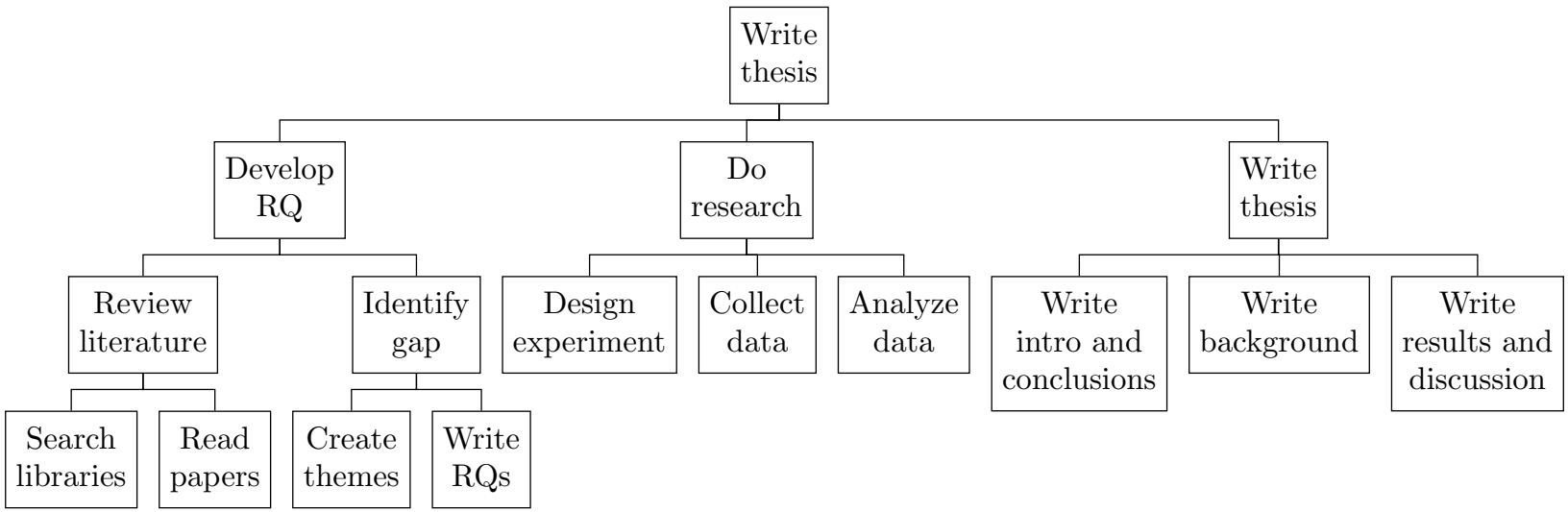
If the input of a task is not dependent on the output of another task, they *could* be performed concurrently if sufficient staff are available.

Complex plans involving concurrent tasks are not so useful in software engineering. Instead, we use an *Agile* approach based on “yesterday’s weather.”

Bigger Gantt chart example



Task-oriented WBS



Don't do this!

Notes:

Task-oriented work breakdown structures provide a general view of the things that need to be done for each deliverable, but they don't specify what needs to be done for a *specific* project.

A true work breakdown structure specifies *what* will needs to be produced.

The *process* or *method* determines *how* it will be produced.

Task-oriented WBS Features

- Task-oriented rather than outcome-oriented.
- All of these things are the same for every project $\Rightarrow$ *process*, not *plan.*

Notes:

The difference between a *process* and a *plan* is repeatability: processes are repeatable, and specify activities, sequence, roles, and outputs.

Given a repeatable process to do something, *plans* should define *what* the project will produce, approximately *when*. The process defines the rest.

The Work Breakdown Structure (WBS) concept was apparently invented by NASA for breaking down *deliverables* into their component parts. As such, the original concept was equivalent to our notion of WBS.

Unfortunately it often gets corrupted into a *task* breakdown structure, probably because of using the word "work" rather than "product." For example, this description of the PMBOK concept of a WBS has *tasks* ("excavate," "steel erection," "rough-in. . . ," "install. . . " "pour concrete"). The purpose of "excavate" is to dig a *hole*; the purpose of "steel erection" is to erect the *steel skeleton* of the building.

Stories, Sprints, Releases

Agile project planning centers on the four concepts of agile development:

1. Stories - semi-structured narrative descriptions of deliverables.
2. Releases - implementations of a collection of deliverables or *Epics* comprising useful end-to-end functionality, that are delivered to end users.
3. Sprints - fixed length (time-boxed) periods of effort ending in useful end-to-end functionality (a collection of *User Stories*), that may be delivered to end users.
4. Scrums - daily “stand up” meetings.

Epics, Stories

- A *release* is equivalent to six months or more of effort comprising several features or *epics*.
- An *epic* is a large chunk of end-to-end functionality.
- An epic must be decomposed into *stories*: semi-structured descriptions of product functionality that a developer can implement in one sprint.
- Implementation of an epic's stories happens in a series of *sprints* of one to four weeks.
- Stories are further decomposed into *scenarios* that can be individually tested. These should represent two days or less of effort.

This means a seven-person scrum team with four developers can implement an epic comprising 250+ stories or scenarios.

Notes:

In the agile methods world, a *story* is a short description of some functionality, written in a certain style: “as a , in order to , I want to .”

Stories are meant to be small enough to implement in on one sprint.

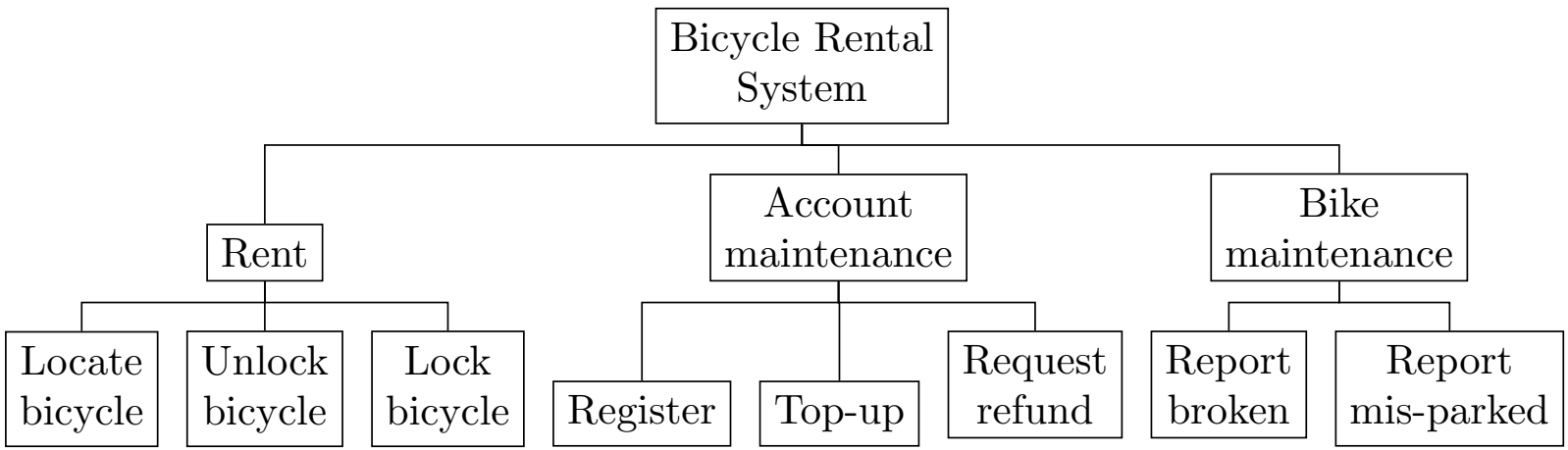
The Cambridge Dictionary defines *epic* as “a film, poem, or book that is long and contains a lot of action, usually dealing with a historical subject.” (https://dictionary.cambridge.org/dictionary/english/epic)

The connotation of epic is that it is longer than a story, hence the use of “epic” to mean “big story.”

In the context of a research project, a story would describe any deliveragle: a data collection instrument, literature review, data set, analysis chapter. An epic would be a publishable result, such as a journal paper or thesis.

Release and Epic Decomposition

Example of an *agile WBS*: the Bicycle Rental System epic, broken down into several stories:

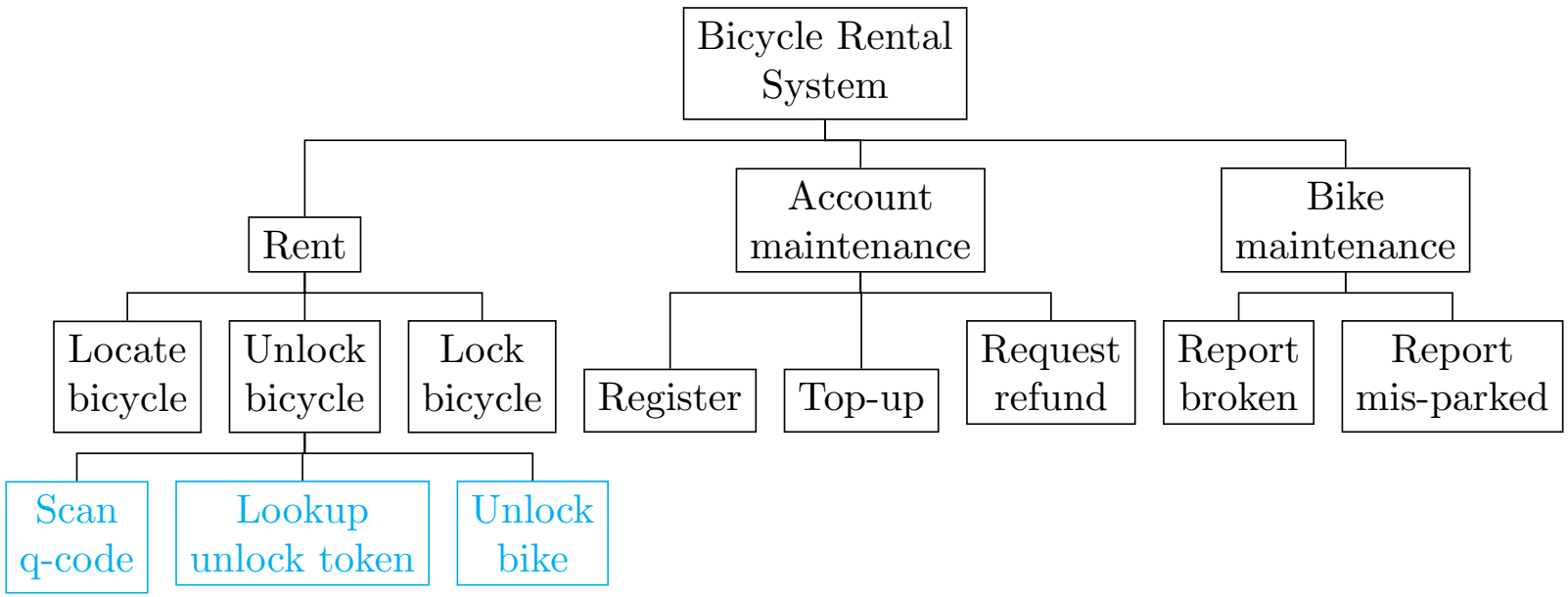


Notes:

- The breakdown into stories is from the user perspective.
- The agile *method* determines *how* the stories are implemented.

Tasks

Stories often require more than two days to implement. So they need to be decomposed further into smaller *tasks*:



Definition of Done

Every project team defines a *definition of done* that specifies when a deliverable has been completed and delivered.

Benefits:

- Improves estimation by making all required activities explicit.
- Aids progress tracking: done is *done*.
- Prevents “gold plating”: when it’s done it’s *done*, stop working on it and move to something else.

Notes:

For software, the definition of done captures the notion that the story is either delivered to the customer, or could be delivered, meaning it's been:

- implemented,
- tested,
- integrated into the main line of development,
- tested,
- demonstrated,
- accepted,
- deployed or delivered.

For research deliverables, the definition of done is more general:

- completed,
- reviewed,
- able to be used as input to subsequent tasks.

Plan Execution

Planning vs. Execution

For a project to be succesful,

1. The tasks in the plan need to be performed.
2. The deliverables need to be delivered.
3. Unnecessary or obsolete tasks need to be dropped in favor of new tasks.

Agile *plans* are executed using agile *methods*.

Kanban

Literally, “sign board:” a matrix of tasks and their status.

A minimal Kanban has three columns, which represent the *status* of a task:

Ready	Doing	Done
Register	Locate bicycle	Unlock bicycle
Top-up	Req. refund	Lock bicycle
Report broken		
Report mis-parked		

The “Ready” column is ordered by priority, with the most important tasks at the top.

The “Done” column shows progress toward completion.

Kanban, cont.

A more realistic Kanban has several intermediate columns:

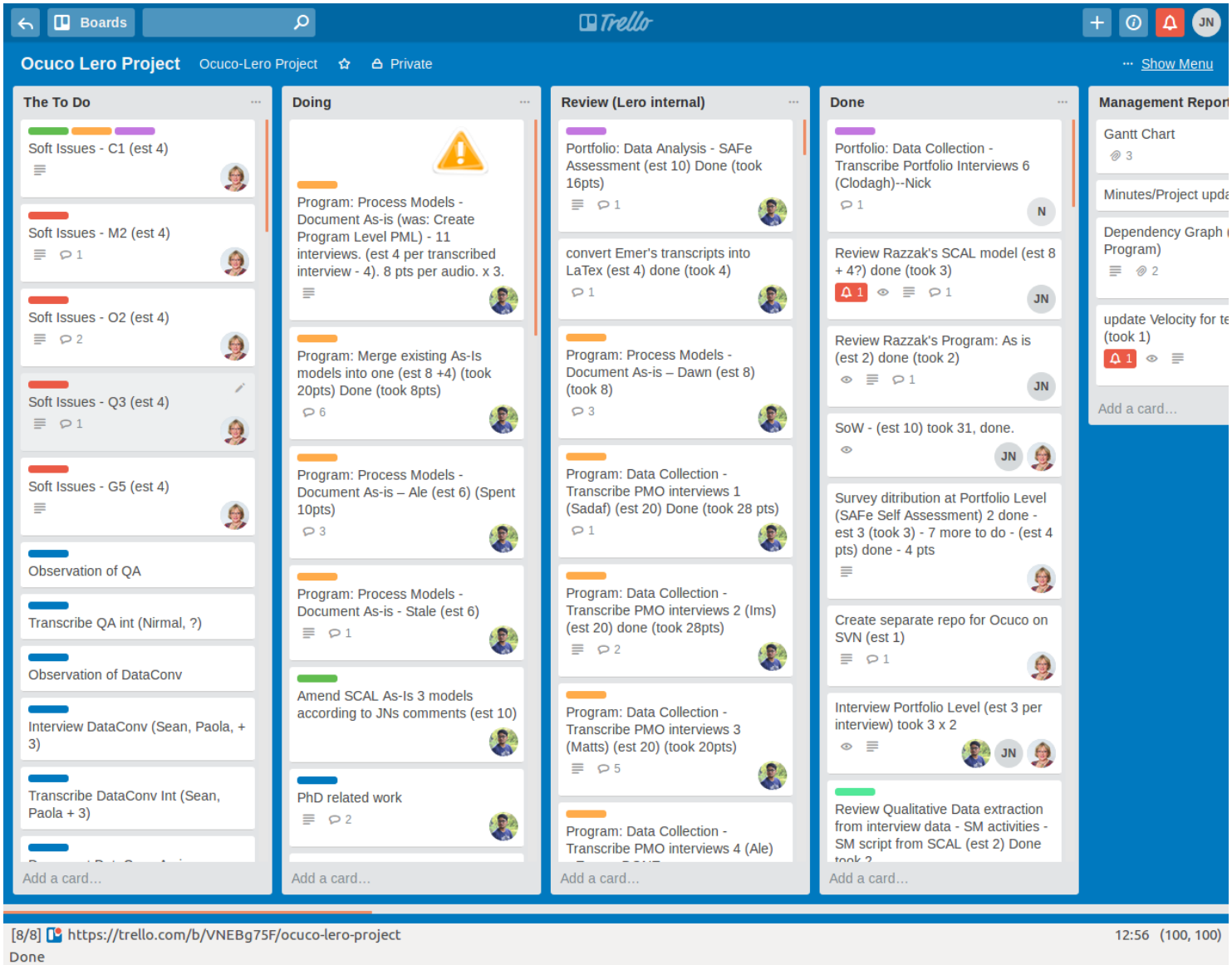
Ready	Doing (1)	Testing (1)	Demo (1)	Done
Top-up Report broken Report mis-parked	Register	Req. refund	Locate bicycle	Unlock bicycle Lock bicycle

Kanban, cont.

A research project needs at least two intermediate columns:

Ready	Doing (1)	In Review (1)	Done
Collect data Analyze data	Trial survey	Develop survey	Review Lit.

Kanban: real example



Kanban Process

1. Each column, except the Ready and Done columns, has a maximum capacity.
2. Tasks are *pulled* from the left column into next column when space is available.
3. Team members shift focus to columns that are blocking tasks from moving.

Notes:

Kanban is part of the “lean” method, which is based on the philosophy that waste is bad. Work-in-progress is a form of waste, because there is a chance it won’t be finished by the deadline, and so becomes wasted effort.

By strictly limiting the number of tasks in a column to an amount that represents the effort available, work-in-progress is reduced.

A Kanban board represents a pipeline that is tuned to achieve high *throughput*: as soon as a task in the rightmost column is completed, another slot opens that can be filled by pulling from the left. This ripples through the Kanban board to the “Ready” column.

Limiting work in progress prevents situations like the “QA backlog” that often develops near the end of a sprint: when the “Testing” column is full, no tasks can enter the “Doing” column until a space opens in the “Testing” column. Development is blocked, so developers should shift focus to testing, to help QA make space for new stories.

Also, if requirements change, some incomplete tasks may need to be cancelled; if they are in-progress, this is wasteful. On the other hand, if they haven’t been started, the cost of canceling a task is only the effort expended in writing it down and making an estimate.

This is why Epics are left as a high-level description until the list of stories in the “Ready” column is reduced to a sprint-ful.

Example

A small project with a Product Owner, Developer, and Tester:

Ready	Doing (1)	Testing (1)	Demo (1)	Done
Req. refund	Locate bicycle	Lock bicycle	Unlock bicycle	
Register				
Top-up				
Report broken				
Report mis-parked				

Each column is limited to *one* task.

Notes:

In this example, the process is blocked until the Developer demonstrates the “Unlock bicycle” story to the Product Owner.

Why was “Unlock bicycle” implemented before “Locate bicycle?” Because we envision having enough bicycles that customers can find them easily just by looking around. We want to start collecting revenue as soon as possible, which can’t be done until the bike is unlocked and the customer rides it somewhere.

Example, cont.

When “Locate bicycle” is demonstrated, it moves to “Done” and makes space for Unlock bicycle:

Ready	Doing (1)	Testing (1)	Demo (1)	Done
Req. refund→	Locate bicycle→	Lock bicycle→		Unlock bicycle
Register				
Top-up				
Report broken				
Report mis-parked				

Notes:

The space made available by moving “Locate bicycle” *pulls* tasks from the left.

Example, cont.

Tasks have been “pulled” from the left columns to fill the open space in “Demo.”

Ready	Doing (1)	Testing (1)	Demo (1)	Done
Top-up Report broken Report mis-parked	Register	Req. refund	Locate bicycle	Unlock bicycle

Notes:

The previous top task in the “Ready” column – “Register” – is now in “Doing.”

The board is now at capacity again, with each team member having something new to do.

Key points:

1. Every column (except “Ready” and “Done”) has a limit to the number of tasks that can be present in that column. When a column is full, no more tasks can enter that state until a space opens on the right.
2. The “Ready” column contains tasks that implement the *stories*, which are also the *leaves* in the agile work breakdown structure, ordered by priority.
3. The *Kanban process* specifies when tasks are started, according to the work-in-progress limits of each column.
4. The *definition of done* determines when a task moves to the “Done” column.
5. The Kanban board presents an immediate visual snapshot of the project’s status. At the beginning it will have more tasks on the left side; toward the end, more will be on the right.

Procrastination is opportunity’s natural assassin. – Victor Kiam

Questions for you:

1. What are the *outcomes* for your coursework?
2. What would its *Work Breakdown Structure* look like?