

[Handout for L4P2] Herding Cats: Making Teams Work

Work Breakdown Structure (WBS)

Before starting project work, it is useful to divide the total work into smaller, well-defined units. Relatively complex tasks can be further split into subtasks. Commonly, the high-level tasks that represent the overall software development stages are first defined, followed by a detailed version containing information about subtasks. A structure that depicts the above information about tasks and their details in terms of subtasks is known as a *work breakdown structure*. In complex projects a WBS can also include prerequisite tasks and effort estimates for each task.

For the Minesweeper example, the high level tasks for a single iteration could look like the following:

Task Id	Task	Estimated Effort	Prerequisite Task
A	Analysis	1 man day	-
B	Design	2 man day	A
C	Implementation	4.5 man day	B
D	Testing	1 man day	C
E	Planning for next version	1 man day	D

The effort is traditionally measured in *man hour/day/month* i.e. work that can be done by one person in one hour/day/month. The *Task Id* is a label for easy reference to a task. Simple labeling is suitable for small project, while a more informative labeling system can be adopted for bigger projects. The high level tasks structure can be further refined into subtasks. For example, the above table can be further refined to:

Task Id	Task	Estimated Effort	Prerequisite Task
A	High level design	1 man day	-
B	Detail design 1. User Interface 2. Game Logic 3. Persistency Support	2 man day • 0.5 man day • 1 man day • 0.5 man day	A
C	Implementation 1. User Interface 2. Game Logic 3. Persistency Support	4.5 man day • 1.5 man day • 2 man day • 1 man day	• B.1 • B.2 • B.3
D	System Testing	1 man day	C
E	Planning for next version	1 man day	D

All tasks should be well-defined. In particular, it should be clear when the task will be “done”. E.g.

not good: more coding | better: implement component X

not good: do research on UI testing | better: find a suitable tool for testing the UI

Scheduling and tracking

Milestones

In projects, a milestone is the end of a stage which indicates a significant progress. For example, each intermediate product release is a milestone. Take into account dependencies and priorities when you decide which features will be delivered at which milestone.

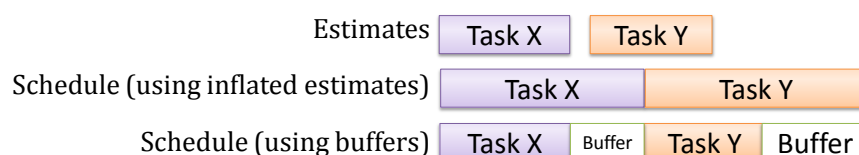
In some projects, it is not practical to have a very detailed plan for the whole project due to uncertainty and unavailability of required information. In such cases we should have a high-level plan for the whole project and a detailed plan for the next few weeks.

Here is an example: [Note that the scope of the task is defined by what we release at the next milestone. i.e. TextUI means “Text UI for V0.1”]

	Day 1	Day 2-5	Day 6	Day 7
Jean	Write MSLogic skeleton	MSLogic	Testing before release	Fine-tuning and/or planning the next version
James	Specify test case format for automated tester	Automated tester		
Kumar	Circulate meeting minutes	TextUI		
Chunling	Update project manual (in point form)	test cases		

Buffers

It is important to have buffers (time set aside to absorb any unforeseen delays). Do not inflate tasks to create hidden buffers. Have them explicitly set aside.



Bug/Issue trackers

Issue trackers (sometimes called bug trackers) are a practical way to track task assignment and progress. E.g. Bugzilla. Most online project management software such as GoogleCode, SourceForge and BitBucket come with integrated issue tracker. Given below is a screenshot from Google Code Issue tracker.

Project Home	Downloads	Wiki	Issues	Source	Administer	
Select: All None Actions...						
	ID ▼	Type ▼	Status ▼	Priority ▼	Owner ▼	Summary + Labels ▼
<input type="checkbox"/> ★	81	----	New	Low	smkhaaant	Avoid repetitions in web pages
<input type="checkbox"/> ★	82	----	Fixed	Low	smkhaaant	Too much black space at ends of H
<input type="checkbox"/> ☆	41	Enhancement	Accepted	Low	jdwxjdw	Timezone and Minutes Support fo
<input type="checkbox"/> ☆	64	Enhancement	Fixed	High	jdwxjdw	Send reminder e-mails automatica
<input type="checkbox"/> ☆	65	Enhancement	Fixed	High	jdwxjdw	E-mails are sent to students on pu
<input type="checkbox"/> ☆	73	Defect	Accepted	Medium	jdwxjdw	Published Results editable by stud
<input type="checkbox"/> ★	77	Defect, Enhancement	Accepted	Medium, Medium	damith	› 'Fair share' may not be clear to so
<input type="checkbox"/> ★	78	Defect, Enhancement	New	Medium, Medium	damith	› Direct login links from home page

Issue trackers can also be used as *task trackers* i.e. to define tasks to be done, track who is doing what and the status of each task.

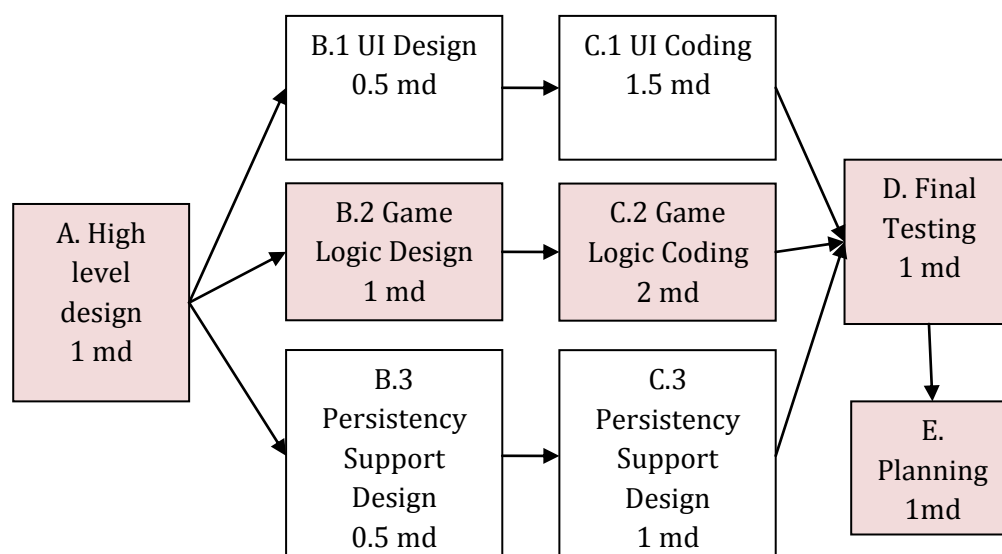
PERT charts and Gantt charts (explained next) are more elaborate tools for scheduling and tracking projects.

PERT (Program Evaluation Review Technique) charts

PERT chart is a graphical technique that is frequently used to show the order (sequence) of tasks. It is based on a simple idea of drawing a directed graph in which:

- Node or vertex captures the effort estimation of a task, and
- Arrow depicts the precedence between tasks

Here is a PERT chart for the Minesweeper example:

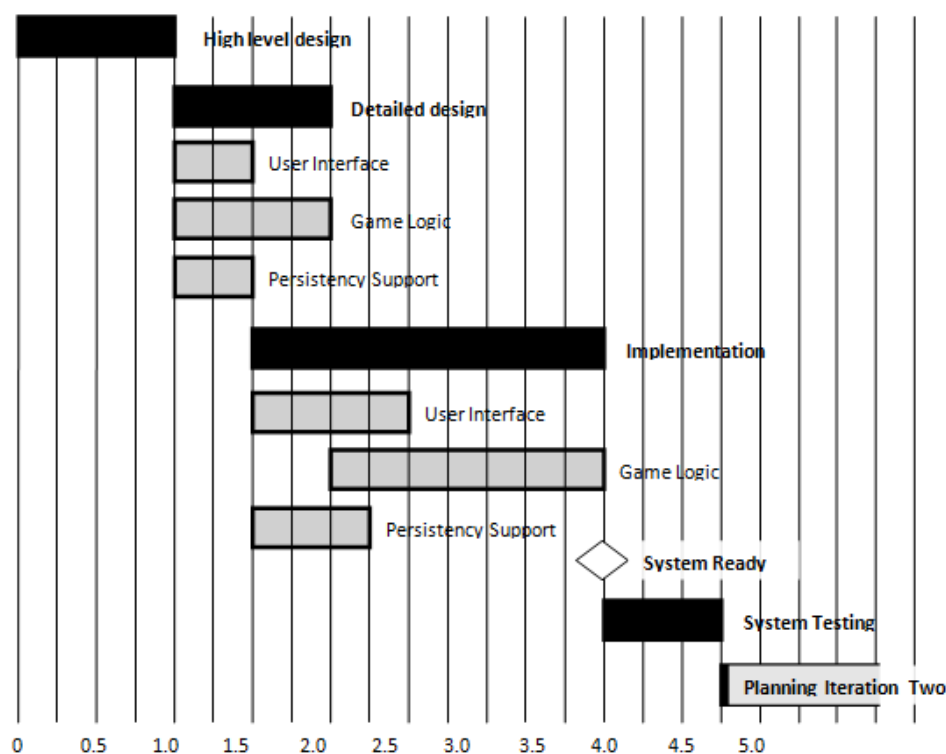


With the above chart, it is much easier to determine the order of a particular task. For example, we can see that the *Final Testing* cannot begin until we have finished coding the individual subsystems. Also, we can see tasks that can proceed concurrently once the prerequisite task(s) has been completed. For example, the various subsystem designs can start independently once we have finished the *High level design*. These parallel tasks allow us to estimate how manpower can be best utilized.

A closely related issue is estimating the *shortest possible* completion time. Let us look at the information in the PERT chart again. We can find a path starting from one of the starting tasks to the final task that determines the shortest possible completion time (indicated by the shaded boxes). This path is given the name *Critical Path* to emphasize that it is *critical* to ensure tasks on this path are completed on time. Any delay would directly affect the delivery time for the project.

Gantt charts

We have seen the usefulness of PERT chart in the previous section. However, there is a shortfall. As each node only shows the elapsed time of a task, it is hard to see the actual start and end time of the node. *Gantt chart* is a graphical tool designed specifically for this purpose. The basic idea is to have a 2-D bar-chart, drawn as time vs tasks (represented by bars) to be performed. For the minesweeper example:



In the chart, a solid bar represents the main task, which is generally composed of a number of subtasks, shown as grey bar. The diamond shape indicates an important deadline or deliverable or a *milestone*.

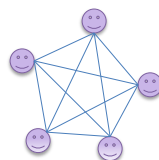
Here some of points to note about scheduling and tracking:

- **Manpower allocation is not just arithmetic:** We have presented a simplistic view so far, which might give you a perception that adding manpower to a task would reduce the time needed by a proportional amount. However, *Brook's law*, by Frederick Brooks, states that: "*Adding more manpower to an already late project makes it even later*". It may sound counter-intuitive, but the reasons are simple: a newly assigned team member requires time to understand the existing system before he/she can be productive. Secondly, adding manpower also increases the communication overhead. Do not forget that a task usually cannot be split into totally independent subunits. Synchronization between team members is frequently needed and can take a significant amount of time. Take this into consideration when you assign more than one person to a task, and adjust the estimated time accordingly.
- **Progress chart is not progress itself:** Do not let chart drawing lures you into thinking you have accomplished a lot. The various charts are just *tools* to monitor project progress, and real work must still be done on the tasks themselves to push the project forward. In the same vein, do not spend too much time on chart drawing.
- **Use reasonable estimate:** Assigning the minimum possible time to each of the tasks may make the project seem well under control, but it will do your team no good if you are forced to adjust the schedule frequently because of overly optimistic estimations.

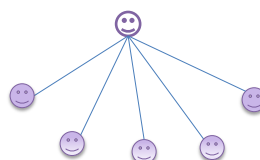
Team structures

Given below are three commonly used team structures in software development. Irrespective of the team structure, it is a good practice to assign roles and responsibilities to different team members so that someone is clearly in charge of each aspect of the project. In comparison, 'everybody is responsible for everything' approach can result in more chaos and slower progress.

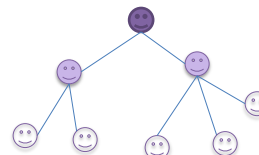
egoless team



chief-programmer



strict-hierarchy



Egoless team

In this structure, every team member is equal in terms of responsibility and accountability. When any decision is required, consensus must be reached. This team structure is also known as *democratic team structure*. This team structure usually finds a good solution to a relatively hard problem as all team members contribute ideas.

However, this team structure also carries relatively higher risk of falling apart due to absence of an authority figure to manage the team and to resolve conflicts.

Chief programmer team

Frederick Brooks proposed software engineers to learn from the medical surgical team in an operating room. In such a team, there is always a chief surgeon, assisted by experts in other areas. Similarly, in a chief programmer team structure, there is a single

authority figure, the chief programmer. Major decisions, e.g. system architecture, are made solely by him/her and obeyed by all other team members. The chief programmer directs and coordinates the effort of other team members. When necessary, the chief will be assisted by domain specialists e.g. business specialists, database expert, network technology expert, etc. This allows individual group members to concentrate *only* on the areas where they have sound knowledge and expertise.

The success of such a team structure relies heavily on the chief programmer. Not only must he be a superb technical hand, he also needs good managerial skills. Under a suitably qualified leader, such team structure is known to produce successful work. .

Strict hierarchy team

In the opposite extreme of an egoless team, a strict hierarchy team has a strictly defined organization among the team members, reminiscent of the military or bureaucratic government. Each team member only works on his assigned tasks and reports to a single “boss”.

In a large, resource-intensive, complex project, this could be a good team structure to reduce communication overhead.

---End of Document---