[L8P1]

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 a 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 as to when the task will be considered “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 deciding on the features to be delivered at a certain milestone.

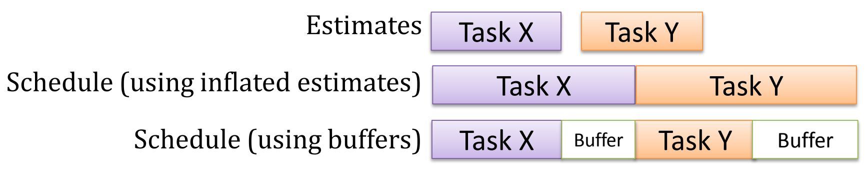
In some projects, it is not practical to have a very detailed plan for the whole project due to the uncertainty and unavailability of required information. In such cases, consider a high-level plan for the whole project and a detailed plan for the next few weeks.

Here is an example: [Note that the scope of the task is defined by the product version that is to be released 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		

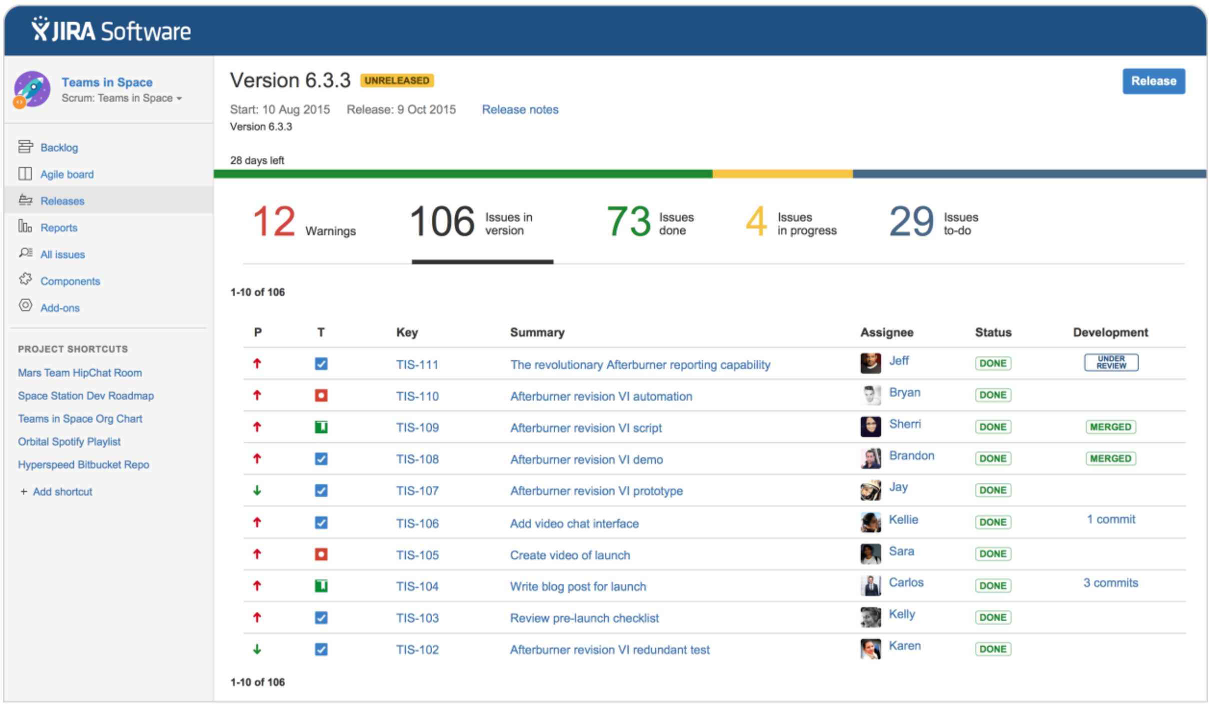
Buffers

It is important to include buffers (i.e. 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 commonly used to track task assignment and progress. E.g. Bugzilla. Most online project management software such as GoogleCode, GitHub, SourceForge and BitBucket come with an integrated issue tracker. The following is a screenshot from the Jira Issue tracker software.


JIRA Software

Teams in Space
Scrum: Teams in Space

Backlog
Agile board
Releases
Reports
All issues
Components
Add-ons

PROJECT SHORTCUTS
Mars Team HipChat Room
Space Station Dev Roadmap
Teams in Space Org Chart
Orbital Spotify Playlist
Hyperspeed Bitbucket Repo
+ Add shortcut

Version 6.3.3 UNRELEASED

Release

Start: 10 Aug 2015 Release: 9 Oct 2015 Release notes
Version 6.3.3

28 days left

12 Warnings 106 Issues in version 73 Issues done 4 Issues in progress 29 Issues to-do

1-10 of 106

P	T	Key	Summary	Assignee	Status	Development
↑	✓	TIS-111	The revolutionary Afterburner reporting capability	Jeff	DONE	UNDER REVIEW
↑	🟥	TIS-110	Afterburner revision VI automation	Bryan	DONE	
↑	🟩	TIS-109	Afterburner revision VI script	Sherri	DONE	MERGED
↑	✓	TIS-108	Afterburner revision VI demo	Brandon	DONE	MERGED
↓	✓	TIS-107	Afterburner revision VI prototype	Jay	DONE	
↑	✓	TIS-106	Add video chat interface	Kellie	DONE	1 commit
↑	🟥	TIS-105	Create video of launch	Sara	DONE	
↑	🟩	TIS-104	Write blog post for launch	Carlos	DONE	3 commits
↑	✓	TIS-103	Review pre-launch checklist	Kelly	DONE	
↓	✓	TIS-102	Afterburner revision VI redundant test	Karen	DONE	

1-10 of 106

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

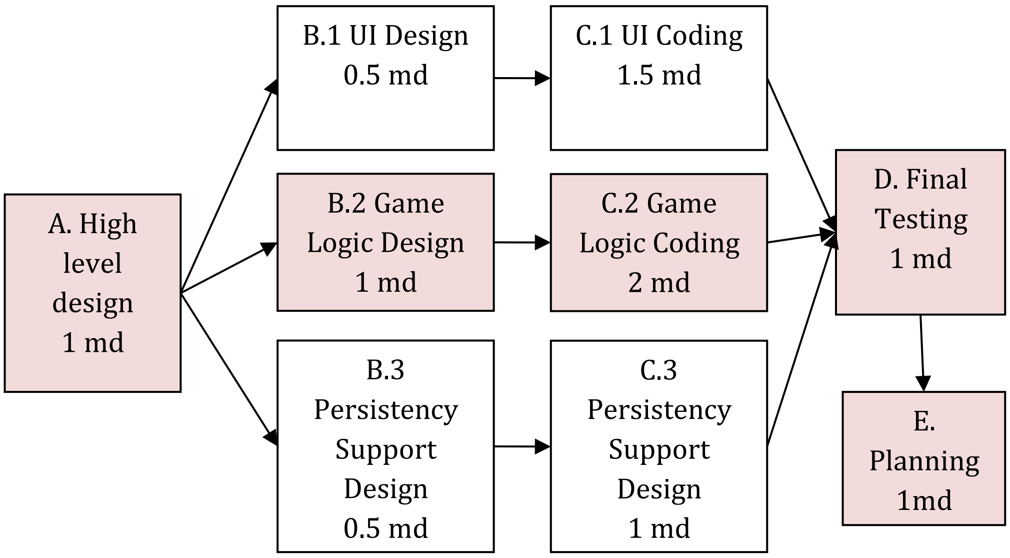
More elaborate tools for scheduling and tracking projects include PERT chars and Gantt charts.

PERT (Program Evaluation Review Technique) charts

PERT chart uses a graphical technique 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:

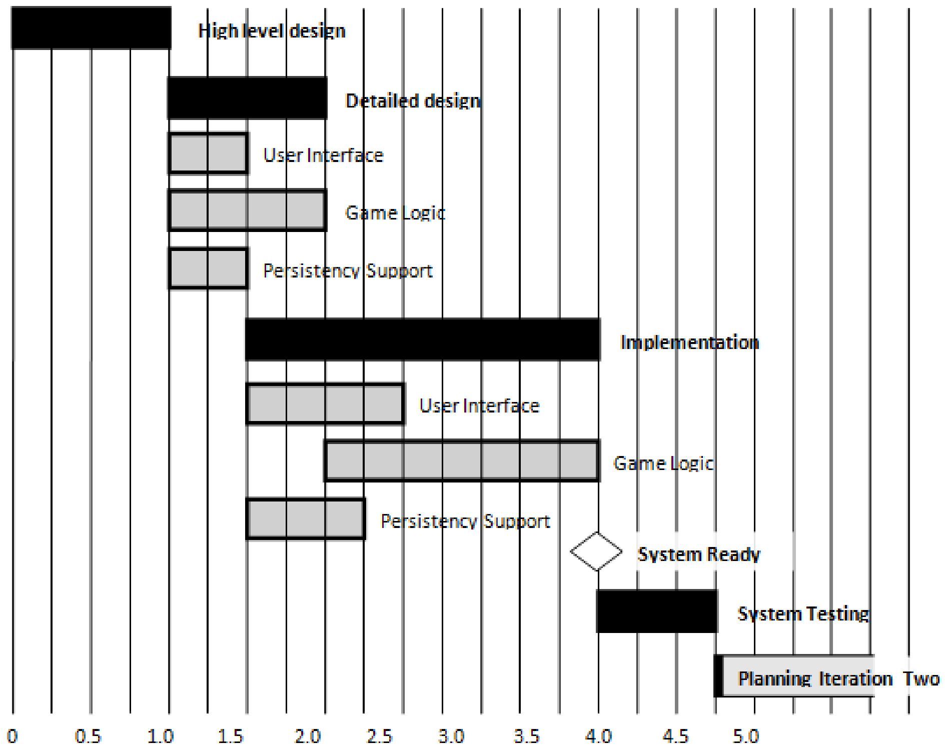

A. High level design 1 md
B.1 UI Design 0.5 md
B.2 Game Logic Design 1 md
B.3 Persistency Support Design 0.5 md
C.1 UI Coding 1.5 md
C.2 Game Logic Coding 2 md
C.3 Persistency Support Design 1 md
D. Final Testing 1 md
E. Planning 1md

With the above chart, it is much easier to determine the order of a particular task. For example, observe that the *Final Testing* phase cannot begin until all coding of individual subsystems have been completed. Also, is it easy to determine the concurrent tasks that can proceed upon completion of the prerequisite task(s). For example, the various subsystem designs can start independently once the *High level design* is completed. The depiction of parallel tasks facilitates the optimal utilization of manpower.

A closely related issue is estimating the *shortest possible* completion time. From the above PERT chart, there is a path (indicated by the shaded boxes) from start to end that determines the longest possible completion time. 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

Despite the usefulness of PERT charts, 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. A *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 to be performed (represented by horizontal bars). Using minesweeper as an example:



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

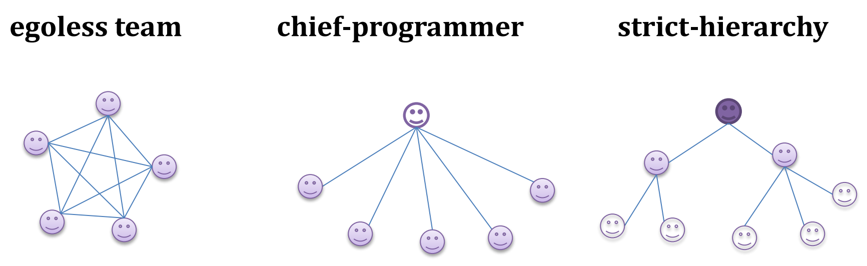
Here are some of points to note about scheduling and tracking:

- **Manpower allocation is not just arithmetic:** Only a simplistic view has been presented so far, which might lead one to think 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 typical task might not be easily split into totally independent subunits. As such, frequent synchronization between team members can take a significant amount of time. Take this into consideration when assigning more than one person to a task, and adjust the estimated time accordingly.

- **Progress chart is not progress itself:** Having a complex chart does not equate to accomplishing a lot of tasks. The various charts are just *tools* to monitor project progress, and real work must still be done on the tasks to push the project forward. In the same vein, do not spend too much time on chart drawing.
- **Use reasonable estimates:** Assigning the minimum possible time to each of the tasks may make the project seem well under control, but it will do the team more harm than good if the schedule has to be adjusted frequently due to 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, the ‘everybody is responsible for everything’ approach can result in more chaos and hence slower progress.



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 a *democratic team structure*. This team structure usually finds a good solution to a relatively hard problem as all team members contribute ideas.

However, the democratic nature of the team structure bears a higher risk of falling apart due to the absence of an authority figure to manage the team and resolve conflicts.

Chief programmer team

Frederick Brooks proposed that software engineers 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 authoritative 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 *solely* 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 a 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.