LECTURE 5: SOFTWARE PROJECT MANAGEMENT

Software Engineering

Mike Wooldridge

1 Introduction

- The "software crisis" of the 1960s and 1970s was so called because of a string of high profile software project failures: over budget, overdue, etc.
- The crisis arose in part because the greater power available in computers meant that larger software projects were tackled with techniques developed on much smaller projects.
- Techniques were needed for *software project management*.

  Good project management cannot guarantee success, but poor management on significant projects always leads to failure.

- Software projects have several properties that make them very different to other kinds of engineering project.
  - *The product is intangible*.
    Its hard to claim a bridge is 90% complete if there is not 90% of the bridge there.
    It is easy to claim that a software project is 90% complete, even if there are no visible outcomes.
  - *We don't have much experience*.
    Software engineering is a new discipline, and so we simply don't have much understanding of how to engineer large scale software projects.
  - *Large software projects are often "bespoke"*.
    Most large software systems are one-off, with experience gained in one project being of little help in another.
  - *The technology changes* very *quickly*.
    Most large software projects employ new technology; for many projects, this is the *raison d'etre*.

- Activities in software project management:
  - project planning;
  - project scheduling;
  - risk management;
  - managing people.

2 Project Planning

- The biggest single problem that afflicts software developing is that of *underestimating* resources required for a project.
- Developing a *realistic* project plan is essential to gain an understanding of the resources required, and how these should be applied.
- Types of plan:
  - *Software development plan*.
    The central plan, which describes how the system will be developed.
  - *Quality assurance plan*.
    Specifies the quality procedures & standards to be used.
  - *Validation plan*.
    Defines how a client will validate the system that has been developed.

  - *Configuration management plan*.
    Defines how the system will be configured and installed.
  - *Maintenance plan*.
    Defines how the system will be maintained.
  - *Staff development plan*.
    Describes how the skills of the participants will be developed.

- We will focus on software development & quality assurance plan.

2.1 The Software Development Plan

- This is usually what is meant by a project plan.
- Specifies the order of work to be carried out, resources, responsibilities, and so on.
- Varies from small and relatively informal to large and very formal.
- Developing a project plan is as important as properly designing code:

  On the basis of a project plan, contracts will be signed and careers made or broken…
- Important not to:
  - overestimate your team's ability;
  - simply tell clients what they want to hear;
  - be pressured by developers ("we can do that in an afternoon!")

2.2 Structure of Development Plan

1. *Introduction*

   brief intro to project — references to requirements spec

2. *Project organisation*

   intro to organisations, people, and their roles

3. *Risk Analysis*

   what are the key risks to the project?

4. *Hardware and software resources*

   what h/ware and s/ware resources will be required for the project and when?

5. *Work breakdown*

   the project divided into activities, milestones, deliverables; dependencies between tasks etc

6. *Project schedule*

   actual time required — allocation of dates

7. *Reporting and progress measurement*

   mechanisms to monitor progress.

2.3 Work Breakdown

- There are many ways of breaking down the activities in a project, but the most usual is into:
  - *work packages;*
  - *tasks;*
  - *deliverables;*
  - *milestones*.

- A *workpackage* is a large, logically distinct section of work:
  - typically at least 12 months duration;
  - may include multiple concurrent activities;
  - independent of other activities;
  - but may depend on, or feed into other activities;
  - typically allocated to a single team.
- A *task* is typically a much smaller piece of work:

  A part of a workpackage.
  - typically 3–6 person months effort;
  - may be dependent on other concurrent activities;
  - typically allocated to a single person.

- A *deliverable* is an output of the project that can meaningfully be assessed.

  Examples:

  – a report (e.g., requirements spec);
  – code (e.g., alpha tested product).

  Deliverables are indicators (but only indicators) of progress.

- A *milestone* is a point at which progress on the project may be assessed.

  Typically a major turning point in the project.

  EXAMPLES:

  – delivery of requirements spec;
  – delivery of alpha tested code.

- Usually…
  - work packages are numbered WP1, WP2, …;
  - tasks are numbered T1.1, T1.2, etc,
    the first number is the number of the workpackage;
    the second is a sequence number.
  - deliverables are numbered D1.1, D1.2, etc
  - milestones are numbered M1, M2 etc.

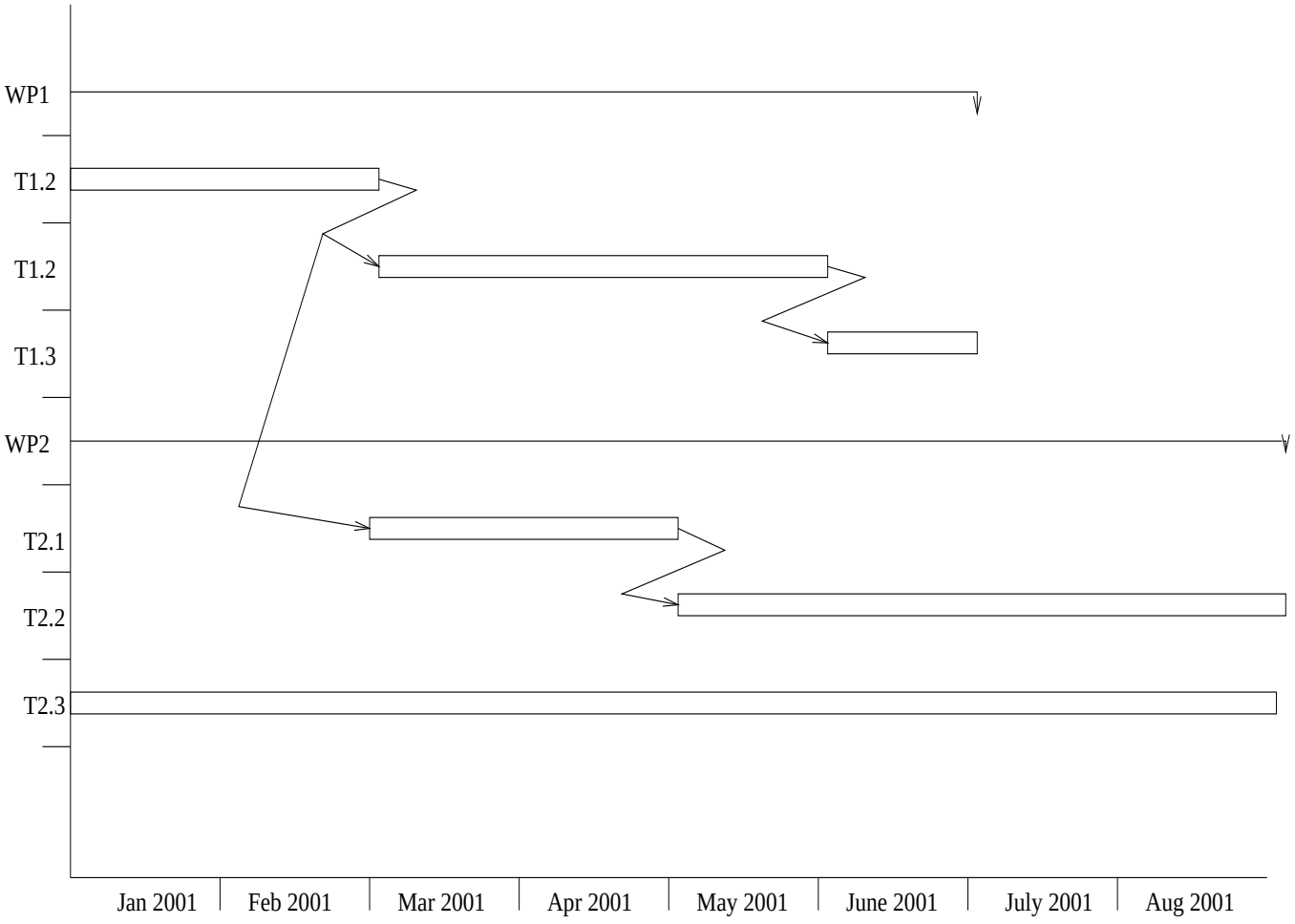
- For each workpackage & task, it is usual to document:
  - brief description;
  - earliest start date;
  - earliest end date;
  - total person months effort;
  - pre-requisite WPs or tasks;
  - dependent WPs or tasks;
  - who is responsible.

2.4 Critical Paths

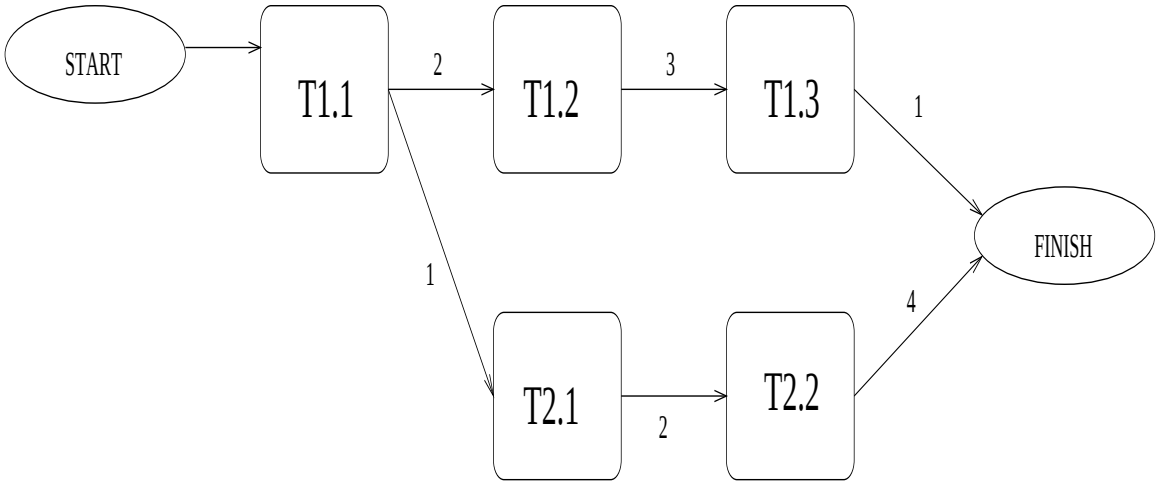
- The pre-requisites and dependencies of WPs and tasks determine a *critical path*: the sequence of dependencies in the project.
- The critical path is the sequence of activities that takes the longest time to complete.
- Any delay to an activity in the critical path *will* cause delays to the overall project.
- Delays to activities not on the critical path need not necessarily cause overall delays.

3 Gantt Charts & Activity Networks

- Gantt charts are a kind of bar chart:
  - time plotted on x axis
  - bars on y axis for each activity.



- An *activity network* is a labelled graph, with:
  - nodes corresponding to activities,
  - arcs labelled with estimated times;
  - activities are linked if there is a dependency between them.



4 Risks

When planning a project, it is critically important to know what the key risks are, and is possible plan for them:

- *staff turnover*;
- *management change*;
- *hardware unavailability*;
- *requirements change*;
- *specification delays*;
- *size underestimate*;
- *technology change*;
- *product competition*.

5 Quality Assurance

- Many organisations make use of a quality assurance plan, which sets out standards to be maintained during project development.
- – Documentation standards:
  - ∗ what documents;
  - ∗ format & content;
  - – Coding standards:
    - ∗ class/method/variable naming conventions;
    - ∗ comment standards (e.g., `javadoc`);
    - ∗ testing conventions;