Project Planning

JD Kilgallin

CPSC:480

09/19/22

Pressman Ch 6, 25

"I have not failed. I've just found 10,000 ways that won't work." –Thomas A. Edison

Project 1

- Submissions for project 1 are now closed, but I will still review anyone's resume any time, on request, outside of the assignment.
  - Per syllabus, late work is not accepted without a **documented,** excusable reason
  - For project 1, "late" means past the final cutoff today, half an hour ago.
  - This is a 400-level class; customers/coworkers in the real world won't be satisfied if work is late because you didn't start till the last day and then felt sick.
- Portfolios should make it easy for the reader to understand what they're looking at. A description in the README about what the software does, how to build/install/run it, and how to use it is helpful.
- Linking your GitHub portfolio in your resume is strongly encouraged.
- Resumes should order work/school experience with most recent first.
- You may want to remove phone numbers/email addresses and give alternate instructions to contact you, such as LinkedIn.

Notes

- Great job on quiz 3, ex 1, and project 1; average was ~95% on each!!
- You may delete or hide the exercise 1 repos if you are done with them. **Do not delete the project 1 repo!**
- See Brightspace for top issues. #1 is merge conflicts in Ex 1.
- Project 2 & exercise 2 (Wed) draft posted. Exercise 2 has two options.
- Group requests for initial team project assignments due tomorrow. Keep in mind teams may shift between (or during) projects at my discretion. So far I have 1 full team and 2 partial team requests.
- I've drafted tentative teams from survey responses regarding areas of interest, languages known, and balance in experience level.

Learning objectives

- Project planning concepts
- Estimating effort required
- Measuring progress according to project plan

Planning

- Aims to optimize cost and schedule of a project to meet requirements and make best use of resources. Primary concern is estimating effort.
- Happens to some degree in conjunction with requirements engineering and product design.
  - Requirements must be known in order to build a plan, but a plan is needed to assess cost/timeline and whether the project is feasible at all.
  - Some design is needed to build a plan, but design work should be planned.
- Iterative process with rough task outline and estimates for the whole project, gradually refined and broken down to sprint plans.
- Some aspects unique to software vs other engineering disciplines (lack of reference, difficulty capturing requirements, scope/complexity).

Types of plan

- Plans occur at many levels: overall product roadmap (indefinite), version release (months-year), individual sprint (weeks-month)
- Initial high-level project plan may only have a rough outline with "t-shirt size" effort estimates (classify requirements as XS/S/M/L/XL)
- Plans for a particular sprint must specify much greater detail on work to be done in that sprint and more precise, granular estimates.
- Amount of detail in a plan at any level varies significantly.
  - More detailed plans take longer and thus risk delaying the release (and may still need to be rewritten due to changing requirements)
  - less detailed plans risk failing to identify challenges ahead of time and make progress harder to measure.

Elements of a plan

- Tasks to be done – Decomposing requirements into work tasks
  - Effort – Estimating amount of work required to complete each task.
  - Dependencies between tasks – "Task network" model.
- Risks & Unknowns – Anticipated technical or management challenges.
- Work products – Defined artifacts (eg designs, prototypes, tests, docs)
- Schedule & milestones – Measurable intermediate goals & timeline.
- Responsibilities – Who will participate in tasks to be done.
- Not all plans (especially early plans) include all elements. "A good plan today is better than a perfect plan tomorrow."

Estimation challenges

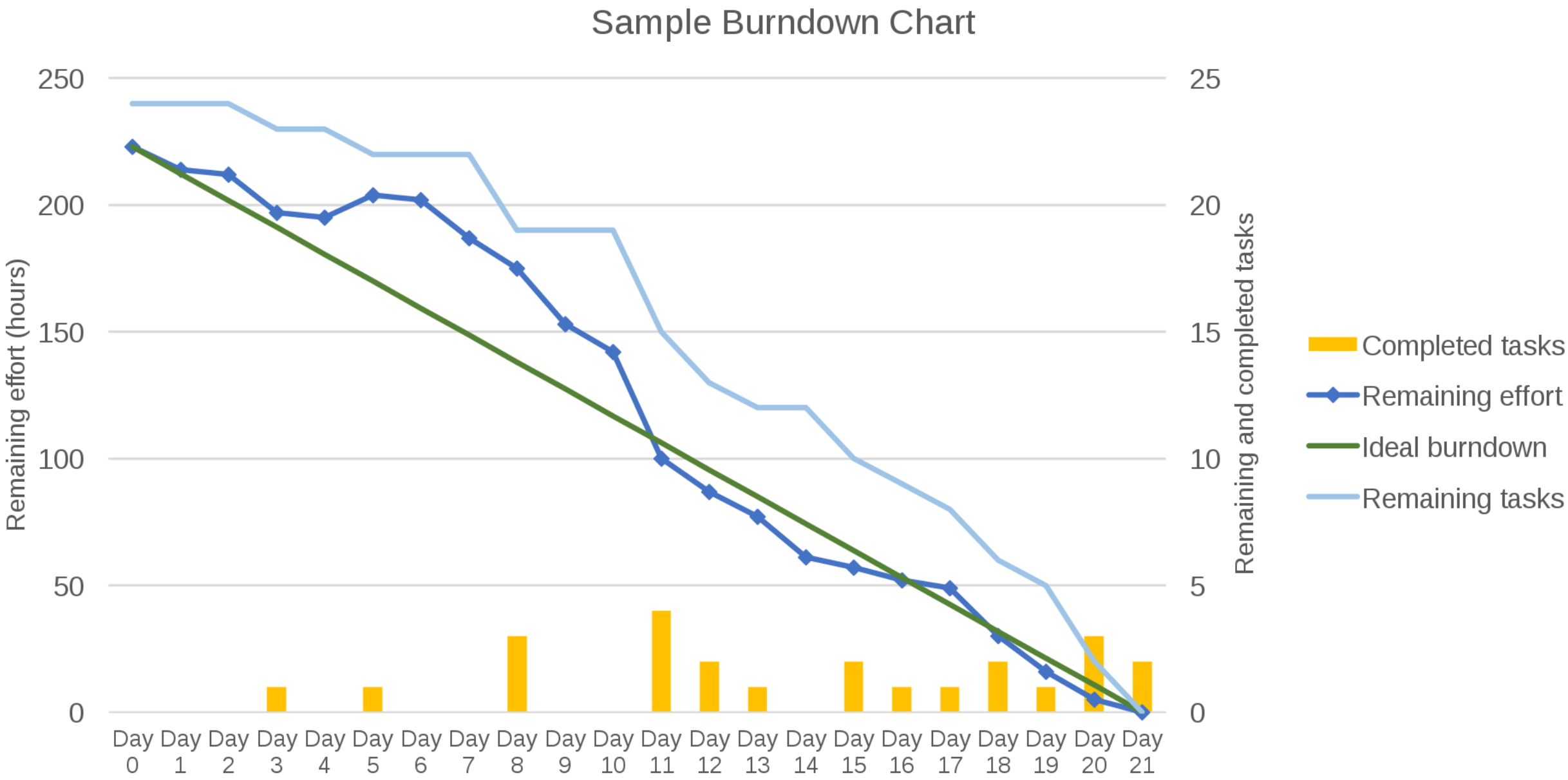
- Earlier estimates risk being inaccurate, later estimates not actionable.
- Requirement changes – Alter scope, invalidate old designs/code
- Refactoring, untracked tech debt, disagreement about implemented code
- Failure to account for architecture/setup, tests, bug fixes, documentation
- Dependency scheduling – Unidentified or delayed task prerequisites
- Difficult integrating changes across a large team
- Technology availability – Hardware/software needed to complete work
- Developer availability – Rarely long-term, but high impact short-term
- Unforeseen technical challenges – "The first 90% of a project takes 90% of the scheduled time. The last 10% takes another 90% of scheduled time."

Time vs effort

- Attempts to accelerate product release schedule requires exponential increase in effort and the number of personnel needed to complete that effort, so available resources must be considered carefully.
  - Additional communication/management overhead grows quadratically.
  - Additional complexity integrating changes; more incompatibilities introduced.
  - More code rewrites needed to maintain project integrity ("too many cooks")
  - Harder to have all developers productive all the time (training, blocked work)
  - Requirement changes and dependency delays affect more people.
- Effort is frequently measured in person-months, but more person-months are required for faster projects.
- *Putnam-Norden-Rayleigh (PNR) curve* suggests effort is proportional to (Lines of Code)^3/(Time)^4. Delivering 10% faster requires 50% more staff; 25% faster => ~4x the staff. Impossible to go much beyond this.

Velocity

- Tracking the amount of estimated work done in each sprint or development cycle across the team.
- Gives insight into estimate accuracy and sources of misestimation.
- Shows team *capacity* - Amount of **estimated** work a team can do in a set time. This is crucial information for future project planning.
- May start slowly in a given cycle as developers ramp up on tasks but will approach a relatively constant value once completed items start rolling in.
- Individual contribution rates can be analyzed and can factor into managers' review of your performance and value to the organization.
- Can compute in lines of code as well as development hours.
- A *Burndown chart v*isualizes velocity, ideal velocity, work remaining over time, projected completion date, and status (ahead/behind schedule).


Sample Burndown Chart
Remaining effort (hours)
250
200
150
100
50
0
25
20
15
10
5
0
Remaining and completed tasks
Completed tasks
Remaining effort
Ideal burndown
Remaining tasks
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Day 11
Day 12
Day 13
Day 14
Day 15
Day 16
Day 17
Day 18
Day 19
Day 20
Day 21

References

- The mythical man-month : essays on software engineering. Frederick Brooks. 1975; 1995. Addison-Wesley Publishing.
  - This is the classic textbook on software engineering project planning and project management. Widely referenced. Highly recommended.
- Software Project Planning. JavaTpoint. 2021.
  - Subpage: Putnam Resource Allocation Model.
- Burndown Chart. Wikipedia.
- Agile Burndown Charts. Erica Chapell. Feb 2020. Clickup.

- *Reading for next lecture: Project 2 and exercise 2a/2b on GitHub*