Today

- Estimation (cont)
- Planning (intro)
- Testing (intro)
- Project Plan

Homework

- HW6 out

Some notes on the project plan document:

- **No floating figures!** Always refer to figures in text, e.g.:
  - “As shown in Figure 1, our system relies on ...” or “Figure 2 shows ...”

- **Text describes figures!** Mostly text descriptions w/ supporting figures.

- **No giant figures!** Figures should be “normal size”
  - e.g., font same or slightly smaller in figures than text

- **Quality figures.** E.g., UI Sketches should be “cleaned up”
  - neatly drawn or digital ... if scanned, high-quality

Major features checklist

- together should define a coherent and largely complete product

- see the template
**Estimation Uncertainty**

We want estimates that we are confident are accurate

- some uncertainty will exist ...
- “This process is called estimation, not exactimation” (Armour)

Estimates often “single-point” numbers (e.g., dev days)

- suggests both high accuracy and confidence

Estimates have inherent uncertainty

- better to think of as ranges and/or confidence levels
- rule of thumb: aim for “90% confidence” in estimates
- e.g., to get 9/10 stories within the estimate on average
Exercise 3: Redo your user story estimates (use planning poker)

(a). Make a “best case” estimate (min possible time/points)

(b). Make a “worst case” estimate (max possible time/points)

(c). Make a “most likely case” estimate (under “normal” conditions)

Steps to find 90% Confidence Level over Estimates (all stories)
... from Program Evaluation & Review Technique (PERT)

(d). Your “expected case” estimate (avg over long run ... 50% confidence)

\[
\text{Expected} = \frac{\text{Best} + (4 \times \text{Most Likely}) + \text{Worst}}{6}
\]

(e). To find 90% confidence, find standard deviation

\[
\text{StdDev} = \sqrt{\frac{\sum (\text{Expected} - \text{ExpectedMean})^2}{n}}
\]

if \( n < 10 \), can approximate as 1/6 range:

\[
\text{StdDev} = \frac{\sum \text{Worst} - \sum \text{Best}}{6}
\]

(f). 90% confidence: Expected + (1.28 \times \text{StdDev})

... watch for precision (e.g., 120 days vs 118.64)

• Note: assumes estimates follow a normal distribution
How well does this really work?

- Estimation only improves from experience
- Being aware of best, worst, most likely can help ...

Caution:

- Most developers “worst case” more like “best case”
- We tend to underestimate!!!

“You never have to fear that estimates created by developers will be too pessimistic, because developers will always generate a too-optimistic schedule.” – Chris Peters, Microsoft
Better Estimation through Task Decomposition

Easier to estimate “piecewise”

1. Break each story into a set of tasks

2. Estimate each task, e.g., using planning poker

3. Sum the task estimates for the story estimate

Exercise 4: Task-based Estimates
Estimation and Planning

Overview of basic approach to planning:

1. Finalize initial backlog for MVP
2. Prioritize backlog items
3. Estimate backlog items (in story points)
4. **Estimate velocity** (e.g., story points per sprint)
5. Revise scope as needed (based on estimates & velocity)
6. Develop milestones (way points) & sprint release plan
How long will your project take?

First approach
- say you add up stories/tasks and get 364 dev days
- and you have 4 developers
  - \(364 \div 4 = 91\) calendar days \(\approx 3\) months

Q: Is this a realistic estimate?
- think of the days on a calendar
- don’t want to sign up for working weekends!

Second approach
- say your target is in 90 calendar days
- there are only 60 dev days in 90 calendar days!

<table>
<thead>
<tr>
<th>mo</th>
<th>tu</th>
<th>we</th>
<th>th</th>
<th>fr</th>
<th>sa</th>
<th>su</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \(364\) dev days \(- (4\) devs \(\times 60\) dev days\) \(= 124\) dev days
- need an additional 124 dev days!

Q: What should we do?
- reduce scope (stories), and/or
- increase cost (devs), and/or
- increase time (commitment)
Q: we still have a problem ... what is it?

- we are assuming everyone is 100% productive every day
- vacation, illness, software/hardware updates, paperwork, ...

**Third approach**

- We need to account for our team’s “velocity”
- measure of how “productive” team is
- alternatively, how off our estimates tend to be
- relies on past estimates and actuals

**Velocity**

- measured as % of productive work

\[
\text{Velocity} = \frac{\text{EstimatedDaysOfWork}}{\text{ActualDaysRequired}}
\]

- Q: How many days needed to do \( n \) estimated days of work?

\[
\text{DaysRequired} = \frac{\text{EstimatedDaysOfWork}}{\text{Velocity}}
\]

- have to keep track of totals through lifetime of (and across) projects
- notice the number of days keep shrinking!
  - 30 calendar days = 20 days of work
  - 20 work days = 14 productive days (if 70% velocity)
Velocity ... An alternative definition ... see textbook

The average number of story points per sprint

• given a backlog of \( n \) story points,
• team velocity \( v \) (in story points / sprint),
• and 2 week sprints
• need \( \frac{n}{v} \) sprints ... or \( 2 \times \left(\frac{n}{v}\right) \) total weeks for 2-week sprints

Exercise: What do you think your team’s velocity is?

• Consider both “definitions” of velocity ...
• How accurate have your estimates been?
• How many story points per sprint (on average)?
Scheduling: Release Planning

**Scheduling**: mapping what you need to accomplish in your project to a timeline and then monitoring your progress to ensure you deliver your product on time

- Our goal is to come up with an initial/rough “schedule”
- Which we are calling the **sprint release plan**
- This is in contrast to a detailed schedule (Gantt Chart)
  - Word Breakdown Structure (WBS) + Dependencies + Schedule

![Example Gantt Chart](image-url)
Milestones

A “milestone” is an important project event or accomplishment

- e.g., when a feature area is completed
- others: components integrated, alpha/beta release, deployment, testing begins/ends, data obtained, etc.

Milestones are an important part of scheduling

- determining when milestones need to occur (planning)
- “checkpoints” to see if project is on track (monitoring)

Defining project milestones

What are the major accomplishments & events along the way?

- consider milestones for all the project deliverables
- consider major feature areas
- consider both technical and non-technical areas
- consider dependencies
- consider testing
- consider deployment (e.g., app submission)
Defining project milestones

Example milestones for your projects ...

- final version of UI design for XYZ features
- initial version of DB schema
- version 1, 2, 3, ... of XYZ features
- final version of authenticated login
- first submission to app store, final submission
- migration to deployment server
- initial set of usability tests complete for v1 features
- system tests for XYZ features
- web service API feature freeze

Exercise: Come up with a list of milestones for your project

- Consider all deliverables (and dependencies) ...
- E.g., documentation, deployment, system testing, usability testing, deployment
Using milestones in scheduling

Goal is to come up with a schedule for your project ...

- we will be using “two-week sprints/iterations”
- after project plan, there are ≈ 22 weeks (7 + 15)
- each sprint will be a “mini” milestone (visible features)
- these should lead to the “major” milestones

What are the goals for each sprint?

- First: when do the milestones need to be completed?
- Then: what will you do each sprint to ensure this?
- While: keeping users engaged ... focus on visible progress