Week 11 - Monday

COMP 3100

Last time

- What did we talk about last time?
- Exam 2!
- Before that: review
- Before that:
  - Deployment and maintenance

Questions?

Project 3

Task Identification and Effort Estimation

Task identification and effort estimation

- As you probably can tell from working on Project 3, it's important for software managers to:
  - Divide the development into tasks
  - Estimate how long those tasks will take
- Otherwise, it's impossible to plan:
  - How long a development project will take
  - How much it will cost
- This information is central to traditional, waterfall processes
- Even in agile, we need to decide on sprintable stories for a sprint

Task identification and organization

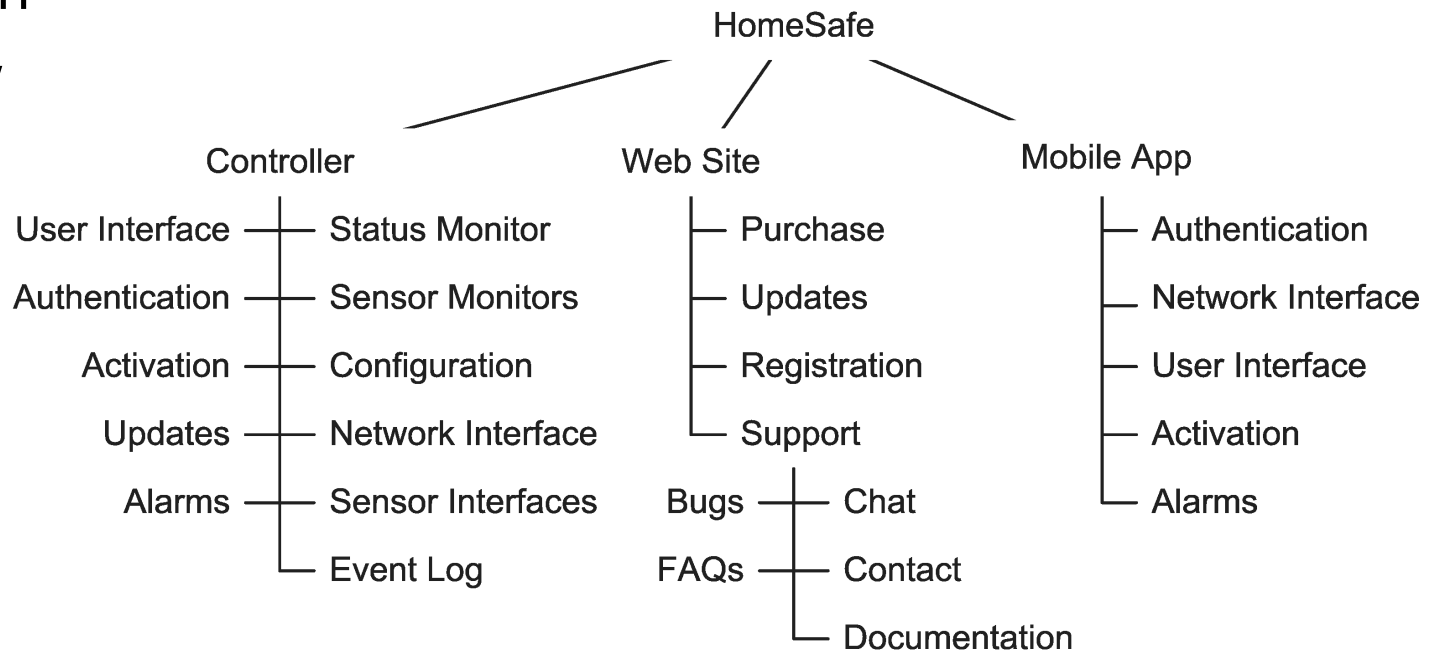
- High level tasks are pretty easy to identify
  - "Add networking support"
- But that level of detail isn't very useful
- Tasks are either:
  - Non-decomposable, also called **actions**
  - Decomposable, also called **activities** or **processes**
- The right level of detail is called a **work package**
  - A work package is a task that is small enough and detailed enough to estimate

Work breakdown structure

- A **work breakdown structure (WBS)** can be used to map out tasks at the right level of abstraction
  - The book prefers hierarchy diagrams to represent a WBS, since they balance the readability of trees with the space efficiency of hierarchical lists
- Nodes in a WBS are work to be done
- The root of a WBS is the project name
- The first level is all the deliverables for a project
- Each level below represents more and more detailed work
- Leaf nodes are work packages

WBS example

- The hierarchy diagram to the right shows a WBS for a home security product
- Note that different strategies can be used to decompose the work, especially at different levels:
  - Project deliverables
  - Product features or services
  - Project phases
  - Organizational units
  - Physical product decomposition
  - Logical product decomposition
  - Geographical location of team members



WBS heuristics

- But how do you know if you've done a good job breaking things down?
- **One hundred percent rule:** Nodes descended from a parent represent 100% of the work of the parent
  - Nothing's left out
  - No work is from outside the project
- **Mutually exclusive siblings:** No sibling nodes have overlapping work
- **8 / 80 rule:** Work packages (the leaves) take between 8 and 80 person-hours of effort
  - Work in that range (one day to two weeks) can be estimated reasonably well
- Get your project team and stakeholders together and make your WBS on a whiteboard

Effort estimation in traditional processes

- In traditional processes, effort estimation can be done in a few ways:
  - **Analogy:** Is your project like another project? It should take about the same effort
    - Problem: Only works if your project is very similar to another project
  - **WBS to effort:** Estimate the effort for each work package in a WBS and add them up
    - Problem: It's really hard to estimate effort accurately
  - **Size to effort:** Estimate the size of the final software product and use some math to predict how much work it will take to make the product
    - Problem: Oh, so many problems, which we'll discuss

Measuring size

- **Functional measures of size** have to do with how much functionality the program provides
  - Number of pages on a website
  - Number of reports in a database
  - Number of windows in a GUI
- **Non-functional measures of size** are based on the program's structure
  - Lines of code
  - Number of classes
- Non-functional measures are easy to measure after development but hard to predict ahead of time

Lines of code

- **Lines of code (LOC)** is a count of the lines of code needed for a project
  - LOC is the most popular non-functional measure of size
- Some people prefer **source lines of code (SLOC)**, ignoring whitespace (and perhaps comments)
  - It's even possible to weight some lines
  - LOC is only meaningful in context, since some programming languages tend to take more LOC to get the same job done
- Estimating LOC is done by breaking the product design into smaller and smaller components until the size of each component can be estimated
- Accuracy is hard to achieve early on, since there isn't even a design yet

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."

-Bill Gates

Function points

- Alternatively, a functional measure of size is possible called **function points**
- Function points are calculated by looking at five different types of components, organized into two categories:
- **Processes or Transactions**
  - **External Inputs (EI)**: Processes that provide data that will be used or stored by the product
  - **External Queries (EQ)**: Processes that retrieve stored data
  - **External Outputs (EO)**: Processes that provide derived information to a user (performing calculations)
- **Data Storage**
  - **Internal Logical Files (ILF)**: Groupings of data maintained by the product
  - **External Interface Files (EIF)**: Groupings of data external to the product but used by the product

More on function points

- Components of each kind contribute different amounts of effort
- Likewise, there are simple, average, or complex cases
- To account for these differences, we give a weight to each component based on this table

Measure	Simple	Average	Complex
External Inputs	3	4	6
External Queries	3	4	6
External Outputs	4	5	7
Internal Logical Files	7	10	15
External Interface Files	5	7	10

Final weights

- It gets worse!
- The whole number of function points is further weighted by answering each of the following 14 questions with a number between 0 (meaning not important) to 5 (meaning essential)
  1. Does the system require reliable backup and recovery?
  2. Are specialized data communications required?
  3. Are there distributed processing functions?
  4. Is performance critical?
  5. Will the system run in an existing, heavily utilized operational environment?
  6. Does the system require on-line data entry?
  7. Does the on-line data entry require input transactions over multiple screens or operations?
  8. Are the ILFs updated on-line?
  9. Are the inputs, outputs, files or inquiries complex?
  10. Is the internal processing complex?
  11. Is the code to be designed to be reusable?
  12. Are conversion and installation included in the design?
  13. Is the system designed for multiple installations in different organizations?
  14. Is the application designed to facilitate change and ease of use by the user?
- The final number of function points is

$$F = \sum_{m=1}^{5} \sum_{d=1}^{3} M_{md} \cdot W_{md} \cdot \left[0.65 + 0.01 \cdot \sum_{q=1}^{14} V_q \right]$$

Other methods

- The Common Software Measurement International Consortium (COSMIC) proposed counting data movements
  - Moving data from or to users or from or to storage
- Roetzheim tweaked function points for web apps
  - EI corresponds to input screens or forms
  - EQ corresponds to externally published interfaces
  - EO corresponds to HTML pages
  - ILF corresponds to internal database tables or XML files
  - ELF corresponds to external database tables or XML files
- Boehm suggests **object points** instead of function points
  - Three measures: screens in the interface, reports, and modules
  - Each measure is simple, medium, or difficult (weighted appropriately)
  - Object points are the sum of weighted measures multiplied by how much reuse there is (between 0 and 1)

Effort estimation

- All these estimates of size give us some arbitrary number, but how much effort is needed?
- **Algorithmic cost models** try to turn size estimates into a measure of effort called the person-month
  - The amount of effort a normal developer does in one month
  - Each person month has about 22 person-days
  - Effort covers all work from requirements, design, coding, testing, documentation, collecting data, management, and so on

Simple models

- Maybe work grows linearly with function points
- Two different studies tried to model this to estimate effort $E = \alpha + \beta F$
- They found the following:

Study	α	β
Albrecht and Gafney	-91.4	0.255
Kemerer	-37.0	0.960

- These results are frustrating
  - The first one suggests that each function point adds ¼ person-month of work
  - The second suggests each function point adds about 1 person-month of work
- They were looking at different organizations and different accounting of function points, so estimates might work well only within an organization that is consistent about such things

Exponential models

- Alternatively, some researchers have looked at exponential models relating thousands of lines of source code (KLOC) to total effort using the following equation, where *L* is KLOC:
  - $E = \alpha \cdot L^{\beta}$
- Results found the following values of α and β:

Study	α	β
Watson and Felix	5.20	0.91
Basili and Freburger	1.38	0.93
Boehm	3.20	1.05

- Note here that $\beta < 1$ means economies of scale (time per line of code decreases at the project grows) while $\beta > 1$ means the opposite

State of the art

- The book goes into the Constructive Cost Model (COCOMO) and its successor COCOMO II
  - It uses some measure (either KLOC or function points)
  - It tweaks an economy of scale parameter based on factors like how similar the project is to previous results and team cohesion
  - It tweaks effort modifiers based on characteristics of the product, platform, team, and language
- If it's not clear to you, we as an industry have no idea how to estimate effort
- Your effort estimates are probably only meaningful if you can compare the product to a similar product made by a similar team

Effort estimation in Scrum

- Everything we said before was about waterfall estimates
- Scrum skips size estimates and goes straight for effort estimates
- As you know, units of effort in Scrum are called **story points** (or sometimes task points)
  - Story points are relative units
  - They're based on some of the smallest tasks, using them as a baseline of 1 story point
  - Everything is estimated relative to those
- Story points aren't used for epics since they're too big and abstract
- As PBIs get refined, their effort estimate gets refined too
- By the time they're sprintable, they need a relatively accurate story point estimate
- This means that there are good estimates for sprintable stories but no estimates for how much work the whole project will take

Detailed estimation in Scrum

- What if members of the team disagree on the story points needed for several stories?
- Agreement is needed for the sake of fairness and to plan how much work can actually get done in a sprint
- **Planning poker** is a way to bring the team to consensus about the relative difficulty of user stories
- Its goal is accuracy (ranking the stories by true difficulty) rather than precision (getting true estimates of how long things will take)
  - It's really hard to get true estimates, but it's good to know which stories take more work

Planning poker

- First, the team decides what numbers to use as estimates
- The numbers are usually sequences that grow exponentially, written on cards
  - Modified Fibonacci: 1, 2, 3, 5, 8 ,13, 20, 40, 100
  - Powers of two: 1, 2, 4, 8, 16, 32, 64
  - This means that large stories won't be estimated precisely, but that's okay
- Planning poker has rounds
  - Each round estimates the effort for one PBI
  - Each team member throws in one card to show her effort estimation
  - If all cards match, the value is the estimate
  - If they don't match, the team discusses their estimates, focusing on the highest and lowest estimators
  - Repeat the round until consensus is reached
- It usually only takes a couple of rounds to reach consensus
- Estimates are usually pretty good because of discussion

Upcoming

Next time…

- Wednesday will be financial planning
- Work day on Friday
  - Please keep working during the week!
  - Deadlines on your Gantt charts are flying by
  - Please come to office hours for help!

Reminders

- Read Chapter 13: Financial Planning for Wednesday
- Keep working on Project 3