1 Reading 2: HFSD Chapter 2, Page 29-64, 66

Gathering requirements, knowing what the customer wants. Although you cannot always get what you want, the customer should. Great software development delivers what the customer wants.

- Gathering requirements
  - When gathering requirements, write them down.
  - A requirement is a single thing that the software has to do.
  - There's no specific format for writing these requirements out.
  - Requirements are similar to user stories, but not exactly like them.
  - Descriptions are probably going to be vague, talk to customer to clarify.
- Talk to your customer to get more information.
  - There will always be gaps in understanding of what your software is supposed to do, esp. early in project.
  - When you have questions, or start to make assumptions, go back and talk with the customer to get answers to your questions.
  - Try to gather additional requirements – talking to the customer doesn't just give you a chance to get more details about existing requirements. You can also find out about additional requirements the customer didn't think to tell you about earlier.
- Bluesky with your customer.
  - When you iterate with the customer on their requirements, think big.
  - Brainstorm with other people (developers, other users, customers, etc.); don't rule out any ideas in the beginning, just try to capture everything.
  - It's okay if you come up with some wild ideas as long as you're all focusing on the core needs that the software is trying to meet.
  - This is called **blueskying** for requirements.

    > Avoid office politics! Try as much as possible to leave job descriptions and other baggage at the door when blueskying requirements. Everyone should get an equal say to ensure you get the most out of each brainstorming session.

  - Sometimes, your bluesky session can be as muffled as a foggy day in winter. That is, some people will give you a lot of information and others will say nothing.

    > The key of capturing good requirements is to get as many of the stakeholders involved as possible. If getting everyone in the same room is just not working, have people brainstorm individually and then come together and put all their ideas on the board and brainstorm a bit more.

- Find out what people really do.
  - Everything that's ethical and legal is pretty much fair game when you're trying to get into your customer's head to understand their requirements.
  - Role playing and observation might be helpful.
    - Role playing: if customer is finding it hard to visualize how they need their software to work, act it out. You pretend to be the software and your customer attempts to instruct you in what they would like you to do. Write down each thing the software needs to do on one of your requirement cards.

    - Observation: sometimes, the best way to understand how people will work with your software is to watch them, and figure out where your software will fit in. Try to observe the same interactions more than once with multiple observers so you don't just have one person's impression of an event.

- Your requirements must be **customer**-oriented.
  - A great requirement is written from your customer's perspective, describing what the software is going to do for the customer. A requirement should be written in the customer's language and read like a **user story**: a story about how their users interact with the software you're building.
  - When deciding if you have good requirements or not, judge each one against the following criteria; user stories should
    - describe one thing that the software needs to do for the customer.
    - be written using language that the customer understands.
    - be written by the customer.
    - be short (no more than three sentences).

    User stories should not
    - be a long essay.
    - use technical terms that are unfamiliar to the customer.
    - mention specific technologies.
  - If you're still unclear after creating more user stories, ask the customer. You're only ready to move on to the next stage when you have no more questions and your customer is also happy that the user stories capture everything they need the software to do for now.
  - User stories usually have a title and description; the title is just a good way of referring to the story.

    > User stories are written from your customer's perspective, and includes information like who the user is, what they want, and why (what the software should do).
    >
    > Requirements are usually more detailed and more technical (how the software should work).

- Develop your requirements with customer feedback.
  - Process for getting customer ideas and refining those ideas into user stories (usually done at the beginning of each iteration):
    1. Capture basic ideas (customer ideas).
    2. Bluesky brainstorming.
    3. Constructing user stories.
    4. Finding holes and clarity on details using customer's feedback (refining your initial set of user stories).
    5. Write clear, customer-focused user stories

- User stories define the **what** of your project, estimates define the **when**.
  - After your initial requirement-capture stage, you will have a set of clear, customer-focused user stories that you and the customer believe capture **what** it is you want to try to build (at least for the first iteration or so).
  - What about **when**? The customer wants to know when those stories will be built. In other words, *how long will it all take?*
    - Your project estimate is the sum of the estimates for your user stories.
    - To figure out how long it will take to complete all of the requirements captured in your user stories, you need to

- add an estimate to each user story for how long you think it will take to develop (design, code, test, deliver) that functionality.
- add up all the estimates to get a total estimate for how long your project will take to deliver the required software.

Hardest part is figuring out the estimates to each user story.

> Getting rid of assumptions is the most important activity for coming up with estimates you believe in.

- Playing planning poker.
  - To come up with accurate estimates, you need to get rid of all the assumptions that put your estimates at risk of being wrong.
  - Want a set of estimates that **everyone** (just the people directly involved in the development, so not customers) believes in and are confident that they can deliver, or at the very least you want a set of estimates that let you know what assumptions everyone is making before you sign on the dotted line.
  - Planning poker is one way to estimate user stories. How to play?
    1. Place a user story in the middle of the table. Remember, we want a solid estimate for how long it will take to develop the story. Don't forget that development should include designing, coding, testing, and delivering the user story.
    2. Everyone is given a deck of 13 cards. Each card has an estimate written on one side. Decks may include
       - 0 days (i.e., already done.)
       - 1/2 day
       - 1 day
       - 2 days
       - 3 days
       - 5 days
       - 8 days
       - 13 days
       - 20 days
       - 40 days
       - 100 days
       - ? (use this card if you don't have enough information to estimate)
       - Coffee (use this card if you think everyone should take a break from estimating for a bit.)
    3. Everyone picks an estimate for the user story and places the corresponding card face down on the table. You pick the card that you think is a reasonable estimate for the whole user story. Don't discuss the estimate with anyone else, though.
    4. Everyone then turns over their cards at the exact same time. It's okay if the cards never match up.
    5. The dealer marks down the spread across each of the estimates. **The larger the difference between estimates, the less confidence you are in the estimate, and the more assumptions you need to root out.**
  - Large spreads can be a misunderstanding.
    - Several reasons why, including: some of your team misunderstood the user story; or, some members of the team are unsure of something that another part of your team is happy with.
    - Look at assumptions that your team is making and decide if you need to go back and speak to the customer to get more feedback and clarification on your user stories, along with any assumptions you're making about them.

    - Even if everyone's estimates is within the same narrow range, it's worth asking for everyone's assumptions to make sure that everyone is not making the same wrong assumption.
  - Put assumptions on trial for their lives.
    - No assumption is a good assumption when it comes to requirements.
    - Try to aim for as few assumptions as possible when making your estimates. When an assumption shows up in planning poker, even if your entire team shares the same assumption, expect the assumption to be wrong until it's clarified by the customer.
    - At least you know what you don't know.
      - Sometimes, your assumptions will survive clarification with the customer. Write this down as a risk for the user story (back of your user story card, for example). You may even choose to delay the development of a user story that has a number of surviving assumptions until they can be clarified.

      > While you can't always get rid of assumptions, the goal during estimation is to eliminate as many assumptions as possible by clarifying those assumptions with the customer. Any surviving assumptions then become risks.

    - Value your customer's time. Some customers may not have a lot of time to talk to you, so you may need to be efficient.
      - Try gathering a collection of assumptions together and then clarifying them all at once with the customer.
      - Schedule an assumption-busting session when you take in the collected assumptions and try to blast as many of them away as possible.

      When you have your answers, head back for a final round of planning poker.
  - Generally, do not think about who implements a user story when coming up with estimates.

    > Don't make assumptions about your assumptions! Talk about everything.

- A big user story estimate is a bad user story estimate.
  - An entire iteration should ideally be around 1 calendar month in duration (approx. 20 working days).
  - Generally, estimates taht are longer tha n15 days are much less likely to be accurate than estimates below 15 days. Estimates greater than 15 days per user story allow too much room for error.
  - When an estimate is too long...
    - Break your stories into smaller, more easily estimated stories: apply the AND rule to break the user story into smaller pieces. Any story that has an "and" in its title or descriptio ncan probably be split into two or more smaller user stories.
    - Talk to your customer again to see if there are any assumptions that are pushing your estimates out.
- The goal is convergence.
  - After a solid round of planning poker, the goal is to not only have estimates for each user story, but be confident in those estimates.
  - Goal is to get rid of as many assumptions as possible, and to converge all of the points on each user story's spread of estimates.
  - e.g. if you have similar estimates like 8, 10, 15 days, a reasonable estimate that a team could agree on is 13 days.
  - Run through this cycle of steps until you reach of consensus.
    1. Talk to the customer: get as much info as possible, remove as many assumptions and misunderstandings as possible, all by talking to the customer.

2. Play planning poker.
3. Clarify your assumptions.
4. Come to a consensues: note the low, converged, and high estimates to get an idea of the best/worst case scenarios.

> Your estimates are your promise to your customer about how long it will take you and your team to deliver.

- Requirements to estimate iteration cycle (usually done at beginning of iteration):
  1. Capture basic ideas (customer ideas).
  2. Bluesky brainstorming.
  3. Constructing user stories.
  4. Finding holes and clarity on details using customer's feedback (refining your initial set of user stories).
  5. Write clear, customer-focused user stories
  6. Play planning poker
  7. Get any missing inforamtion from customer, try to break up large user stories.
  8. Estimate how long all of customer's requirements will take.
- Estimating the whole project.
  - Now that we have an estimate for each story, we can ge ta total estimate for how long your project will take to deliver the required software.
    - Just add each of the converged estimates for your user stories. The total duration for your project is the sum of those converged estimates.