1 Reading 3: HFSD Chapter 3 Page 69-99, 103

Every great piece of software starts with a great plan.

- Customers want their software now.
  - Customers want their software when they need it, and not a moment later.
  - What if the problem is that developing everything the customer said they needed will take too long?
    * You can pick the stories you think are important, but **the customer sets the priorities**.
    * When user stories are being prioritized, you need to stay customer-focused. Only the customer knows that is really needed. So, when it comes to deciding what's in and what's out, you might be able to provide some expert help, **but it's a choice that the customer has to make**.
- Prioritize with the customer.
  - It's your customer's call as to what user stories take priority.
  - To help the customer make the decision, shuffle and lay out all your user story cards on the table.
  - Ask the customer to order the user stories by priority (the story most important to them first) and then to select the set of featurs that need to be delivered in Milestone 1.0 of their software.
- What is "Milestone 1.0?"
  - Your first major release of the software to the customer.
  - Unlike smaller iterations where you'll show the customer your software for feedback, this will be the first time you actually **deliver your software** (and expect to get paid for the delivery).
  - Some Do's and Don'ts when planning Milestone 1.0:
    * Do
      · balance functionality with customer impatience: help customer understand what can be done in the time available. Any user stories that don't make it into Milestone 1.0 are not ignored, just postponed until Milestone 2 or 3 or ...
    * Don't
      · Get caught planning nice-to-haves: Milestone 1.0 is about delivering what's needed, and that means a set of functionality that meets the most important needs of the customer.
      · Worry about length (yet). At this point, you're just asking your customer which are the most important user stories. Don't get caught up on how long those user stories will take to develop. You're just trying to understand the customer's priorities.
  - We know what's in Milestone 1.0
    * Collect together all the features of your software that your customer needs developed for Milestone 1.0.
  - Sanity-check your Milestone 1.0 estimate
    * Now that you know what features the customer wants in Milestone 1.0, time to find out if you now have a reasonable length of project if you develop and deliver all of the most important features.
  - If the features don't fit, re-prioritize
    * Suppose you have significantly more days of work for Milestone 1.0 than what the customer wants. This is pretty common since customers usually want more than what you can deliver. It's your job to go back to them and reprioritize until you come up with a workable feature set.
    * To reprioritize your user stories for Milestone 1.0 with the customer...

1. Cut out more functionality: the very first thing you can look at doing to shorten the time to delivering Milestone 1.0 is to cut out some features by removing user stories that are not **absolutely crucial** to the software working. Once you explain the schedule, most customers will admit they don't really need everything they originally said they did.
2. Ship a milestone build as early as possible: aim to deliver a significant milestone build of your software as early as possible. Don't let your customers talk you into longer development cycles than you're comfortable with. The sooner your deadline, the more focused you and your team can be on it.
3. Focus on the baseline functionality: milestone 1 is all about delivering JUST the functionality that is needed for a working version of the software. Any features beyond that can be scheduled for later milestones.

- Difference between milestone and version: not much. Milestone marks point in which you deliver significant software and get paid by your customer, whereas version more of simple descriptive term that is used to identify particular release of your software.
  - Version = label, doesn't mean anything more.
  - Milestone = you deliver significant functionality and you get paid.
- Software's baseline functionality: smallest set of features that your software needs to have in order for it to be useful to your customer and their users.
- If, after no matter how much you cut the stories up, you can't deliver what your customer wants when they want you to, confess to your customer. If your customer refuses to budge, you may need to walk away. You can also try to beef up the team with new people, although this may not get you any significant advantages.

- It's about more than just development time.
  - When adding more people can look attractive at first, it's not as simple as "double the people, halve the estimate."
    - Every new team member needs to get up to speed on project.
    - They need to understand the software, technical decisions, and how everything fits together.
    - While they're doing that, they cannot be 100% productive.
    - After that, you still need to get the new person set up with the right tools and equipment to work with the team. This all takes time.
    - Finally, every new person you add to the team makes the job of keeping everyone focused and knowing what they are doing harder.
    - In fact, there's a maximum number of people that your team can contain and still be productive, but this depends on your project, team, and who you're adding.
      - Monitor the team. If you see your team is getting less productive, even though you have more people, time to re-evaluate the amount of work you have to do or the amount of time in which you have to do it.
- More people sometimes means diminishing returns!
- Work your way to a reasonable Milestone 1.0
  - Steps
    - First, try to add new people to your team.
    - Then, re-prioritize with customer.
  - Priorities are usually in multiples of 10s (10, 20, 30, 40, 50). This gets the brain thinking about groupings of features, instead of ordering each and every feature separately with numbers like 8 or 26 or 42.
  - Customer should be making the priorities, not you.

> Keep your software continuously building and your software always runnable so you can always get feedback fro mthe customer at the end of an iteration.

- Iterations should be short and sweet.
  - Keep iterations short: the shorter your iterations are, the more chances you get to find and deal with changes and unexpected details as they arise.
  - Keep iterations balanced: each iteration should be a balance between dealing with change, adding new features, beating out bugs, and accounting for real people working. 30-day iterations are basically 30 calendar days, which you can assume turn into about 20 working days of productive development.
  - Short iterations help you deal with change and keep you and your team motivated and focused.
- Velocity accounts for overhead in your estimates.
  - **Velocity** is a percentage: given X number of days, how much of that time is productive work?
  - Start with a velocity of 0.7.
    - On first iteration with a new team, it's fair to assume that your team's working time will be about 70% of their available time.
    - This means taht your team has a velocity value of 0.7.
    - That is, for every 10 days of work time, about 3 of those days will be taken up by holidays, software installation, paperwork, phone calls, and other nondevelopment tasks.
  - This is a conservative estimate and you may find that, over time, your team's actual velocity is higher.
  - If that's the case, then at the end of ucrrent iteration, adjust velocity and use that new figure to determine how many days of work can go into the next iteration.
  - You can also apply velocity to your amount of work and get a realistic estimate of how much that work will actually take.

    $$\frac{\text{Days of work}}{\text{Velocity}} = \text{Days required to get work done.}$$

    Here,
    - Days of work = days of work it takes you to develop user story, or iteration, or even entire milestone.
    - Velocity = between 0 and 1. Start with 0.7 on a new project as a good conservative estimate.
    - Days required to get work done = should always be bigger than the origianl days of work, to account for days of administration, holidays, etc.

    Alternatively,

    $$\text{Workable Days} * \text{Velocity} = \text{Actual Working Days}$$

- Programmers think in utopian days. Programmers will always give you a better-than-best-case estimate. Developers think of real world days.
  - To be a software developer, you need to deal with reality. You probably got a team of programmers and you got a customer who won't pay you if you're late. Your estimates should always be more conservative and take into account real life.

    $$1 \text{ Calendar Month} \xrightarrow{\text{Remove weekends/holidays}} 20 \text{ Workable Days} \xrightarrow{\text{Apply velocity}} 14 \text{ days of real work.}$$

- Deal with your velocity before you break into iterations.
  - By applying velocity up front, you can calculate how many days of work you and your team can produce in each iteration.

  - Then, you know exactly what you can really deliver in MS 1.
  - Steps
    * First, apply team velocity to each iteration. You can calculate how many days of actual work your team can produce in one iteration.

$$\text{Num. Ppl.} \cdot \text{Working Days/Iteration} \cdot \text{Team's First-Pass Velocity} = \text{Work Days}$$

$$\text{Work Days} \cdot \text{Num. Iterations in MS 1} = \text{Amt. Work Days in MS 1}$$

- Making an evaluation
- Deliver bad news to customer.
- Managing pissed off customers.
  - What do you do when this happens?
    1. Add an iteration to milestone 1.0. Extra work can be done if an additonal iteration is added to a plan. Longer development time will be required but at least they get what they want.
    2. Explain that the overflow work is not lost, just postponed.
    3. Be transparent about how you came up with your figures.