User Stories to Tasks

Taking an Iteration Down to Code
User Stories to Tasks

Breaking up the work and keeping track
User Stories are hard to take to code

- In customer language
- Features
- “Big”, multi-skilled

Answer is to subdivide Stories into Tasks. Task is:

- Written in technical language
- Small unit of developer work
  - Divided by skill, subsystem, etc.
- Often a module of executable functionality (class or two)
Goals of Creating and Executing Tasks

1. Break iteration’s user stories into units of work to be performed (tasks)

2. **Estimate** work-days of the tasks
   - Apt to be more accurate than user story estimates

3. **Prioritize tasks** by user story priorities and task dependence
   - Divvy up work according to skills (or in our case, to learn skills!)
   - Try to order assignment of tasks so that system always runs

4. Daily start-of-day **stand-ups** to assess progress and identify problems

5. Track work on **big board**, with **burn-down chart**
New Concept: **Task**

- **Customer-oriented feature**
  - Your task descriptions should have just enough information to describe what the actual development work is.
  - Example: Task 1: Create a date class that contains events.
  - Task 2: Create user interface to create, view, and edit a date.
  - Task 3: Create the schema for storing dates in a database.
  - Task 4: Create SQL scripts for adding, finding, and updating date records.

- **Code-oriented functionality**
  - It's OK if your tasks are a bit different, as long as they cover all the user story's functionality.

- **Estimates**
  - Example: Task 1: 2 days, Task 2: 5 days, Task 3: 2 days, Task 4: 2 days.
  - Your new task estimates:
    - Task 1: 2 days
    - Task 2: 5 days
    - Task 3: 2 days
    - Task 4: 2 days

- **Update** (was 9 days)

---

Tasks are units of computation to be developed by one person:

- Technically oriented; not end-user features

Often distinguished by skill set or subsystem

Estimates likely more accurate
Tasks for Friends on Map?

1. Display birds-eye map with user at the center (H)

2. Show GoogleTalk friends on the map (3 days)
   - (learn GT API)
   - user login to GT
   - retrieve the friend list from GT
   - retrieve locations of friends
   - display friend name on the map
The Hierarchical Structure of Agile Planning

All part of **making a large project act like a small one**

Different frequencies for different types of “ship” manages the tradeoff between **cost** of shipping and **value** of feedback.

ียว

Released to team

Demo’d to customer

Released to self/pair/US “team”
Expanded Concept: **Big Board**

Keeps track of what’s doing, who, and how fast

- Emerging requirements
- Bug fixing
- …

**Unplanned Tasks**
Scheduling Tasks on the Big Board

User stories

Title: Create a date
Description: User creates their custom date package

Task 4 MDE
Create SQL queries for adding, finding and updating date records

Task 1 BJD
Create a date class that contains events

Task 2 LUG
Create user interface to create, view and edit a date

Task 3 MDE
Create the schema for storing dates in a database

Schedule tasks in priority order (from user story priority)
Also task dependence – if A calls B, do B first!

Ideally system is always runnable
With every task completion, plot a point at (days left, work left)

This task takes 5 days to finish, where put next point?

Starting at (15, 34)
5 days past, 2 days of work

A. (13, 29)  B. (29, 13)  C. (10, 32)  D. (32, 10)

C. (10, 32)
New Concept: Stand-Up Meeting

- Key to Agile is knowing where you are, so you can make frequent, “agile” adjustments

- Big Board + Standups

- 5-15 minute daily AM meeting
  1. Progress, status
  2. Problems that are slowing things down
  3. Update big board
What **risks** do **stand up meetings** address?

A. falling behind (Burn Down?)

B. wrong assumptions about what others are doing (Big Board?)

C. unforeseen problems (identifying, start a fix) [ability to quickly iterate on prob. in meeting]

D. someone is stuck and needs a rescue [social aspects are key]

E. (Risk of coming apart as a team)
No Huddle Offense

Suzie says that the class project is too short to have time for daily stand-ups, plus the team’s individual schedules don’t align. She is proposing a **weekly meeting** on Sunday night and **daily check-ins via the mailing list**. What’s wrong with her plan?

A. E-mail doesn’t allow for fluid conversation, looking at documents together, eye contact, etc.

B. On a short, intense project, you may have shorter iterations, etc., requiring more meetings, not fewer

C. Team attitude, focus, and accountability may be lost – people may slack off, forget, or gloss over problems

D.
Your project has fallen behind on its iteration. What should your team do?

A. Look for causes – skill mismatches, bad software design, wrong velocity, underestimation, missed tasks, missed of risks

B. Assess possible consequences – late vs. drop features (e.g., hard to drop features from baseline)

C. Let the customer know, include in decision process

D. Early in the project, “watch and see” can be OK
Take-Aways from Class Today

- **User stories are too big and diverse** for one person to do – unnecessarily long to finish.

- **Tasks** are coherent **units of work**
  - Order them so that the **system is always runnable**

- **Big board** tells where you are at all times
  - Keeps team from getting in each other’s way

- **Burn down** chart lets you know when you’re falling behind
  - Talk to customer
  - Lets you make adjustments early

- It’s **all about identifying problems & risks early**