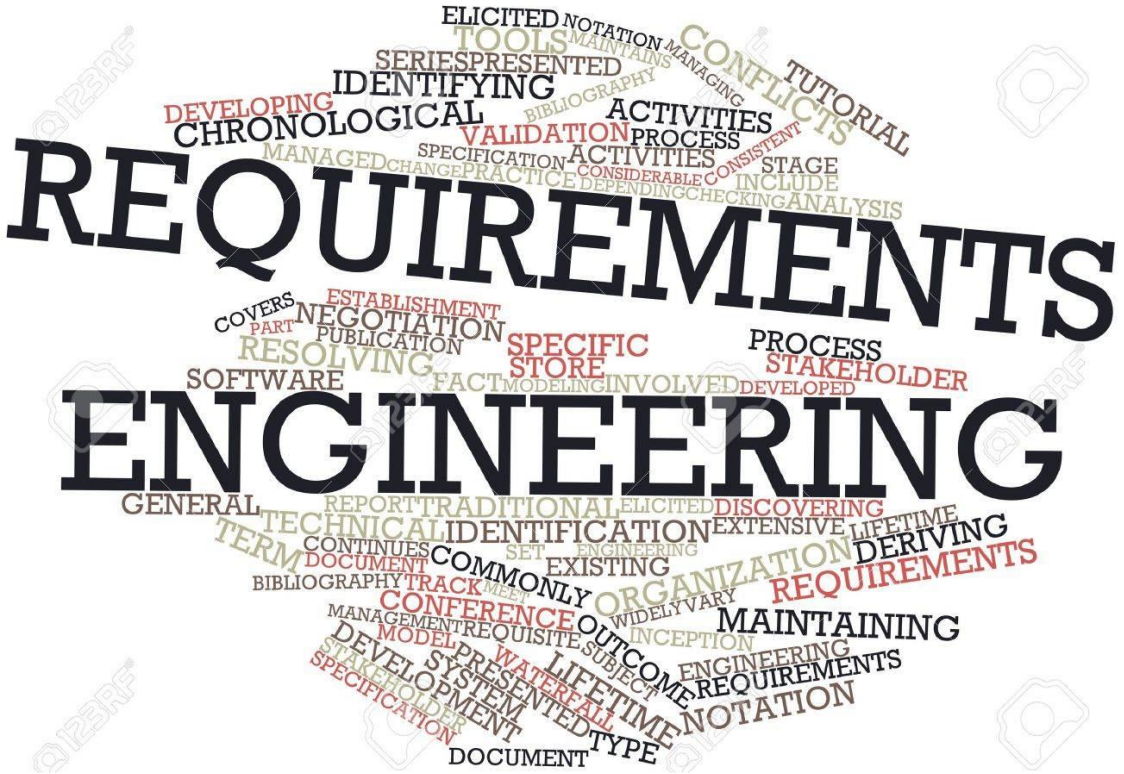


SWE 215: Software Requirements Engineering

Lecture 8

User Stories and Agile Estimation

Course Topics

- ~~Why Requirements Engineering?~~
- ~~Introduction to Requirements~~
- ~~RE in Software Development Life Cycles~~
- ~~System Vision, Context, and RE Framework~~
- ~~Fundamentals of Goal Orientation~~
- ~~Fundamentals of Scenarios~~
- ~~Requirements Discovery~~
- User Stories and Agile Estimation
- Features Prioritization
- Requirements Negotiation
- Requirements Validation
- Fundamentals of Requirements Management

Lecture Objectives

- Learn how to model user stories
- Learn user story estimation techniques

Lecture Outline

- User Story
- User Story Acceptance Criteria
- Spikes
- Relative estimation
- Estimation using story points
- Estimation using planning poker
- Estimation using tabletop
- Team velocity
- Estimation using Ideal Developer Days (IDDs)

User Story Overview

Basic definition of a **user story**:

A user story is a brief statement of intent that describes something the system needs to do for the user.

- In **XP**, **user stories are often written by the customer**, thus integrating the customer directly in the development process
- In **Scrum**, **the product owner often writes the user stories**, with input from the customer, the stakeholders, and the team.
- In **practice**, **any team member with sufficient domain knowledge** can write user stories, but it is up to the product owner to accept and prioritize these potential stories into the product backlog.

User Story Overview

- A user story captures a short statement of function on an **index card** or perhaps with an **online tool**.
- In **simple backlog form**, stories can just be a list of things the system needs to do for the user:

Log in to my web energy-monitoring portal
See my daily energy usage
Check my current electricity billing rate

- The user story provides the common language to build understanding between the user and the technical team (**help bridge the developer – customer communication gap**)

User Stories are not Requirements

They are different from use cases and SRS in many ways:

- They are **not detailed requirements specifications** but are rather negotiable expressions of interest.
- They are **short**, **easy to read**, and **understandable** to developers, stakeholders, and users.
- They represent **small increments of valued functionality** that can be developed in a period of days to weeks and **can be safely discarded after implementation** (no need for maintenance).
- They are relatively **easy to estimate**, so effort to implement the functionality can be rapidly determined.

User Story Form

Three elements of a user story:

- **Card** represents two to three sentences used to describe the intent of the story (details remain to be determined).
- **Conversation** represents a discussion between the team, customer, product owner, and other stakeholders, which is necessary to determine the more detailed behavior required to implement the intent.
- **Confirmation** represents the acceptance test, which is how the customer or product owner will confirm that the story has been implemented to their satisfaction.

User Story Voice

It has the following form:
***As a* <role>, *I can* <activity> *so that* <business value>**

- **<Role>** represents who is performing the action or perhaps who is receiving the value from the activity.
- **<activity>** represents the action to be performed by the system
- **<business value>** represents the value achieved by the activity

Example:
As a **consumer** (<role>), I want to be **able to see my daily energy usage** (<what I do with the system>) so that I **can lower my energy costs and usage** (<business value I receive)"

User Story Detail

- The details for user stories are conveyed primarily through **conversation between the product owner and the team**.
- In case more **details** are needed about the story, they can be provided in the form of an **attachment** (**mock-up, spreadsheet, algorithm**, etc.).
- Additional notes, assumptions, and acceptance criteria can be kept with the user story.

User Story Acceptance Criteria

- User Story:

As a consumer, I want to be able to see my daily energy usage so that I can lower my energy costs and usage"

Examples of acceptance criteria:

- *Read DecaWatt meter data every 10 seconds and display on portal in 15-minute increments and display on in-home display every read.*
- *Read KiloWatt meters for new data as available and display on the portal every hour and on the in-home display after every read.*

- **Acceptance criteria are not functional or unit tests; rather, they are the conditions of satisfaction being placed on the system.**

User Stories

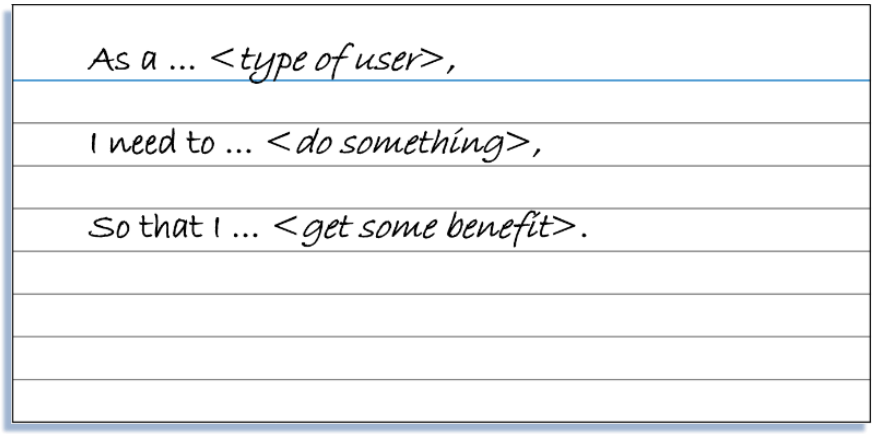
Epics

- A Scrum epic is a **large user story**.

Theme:

- A "theme" is a **collection of user stories**.
- Sometimes it's helpful to think about a group of stories so we have a term for that.

Story Modeling with Index Cards

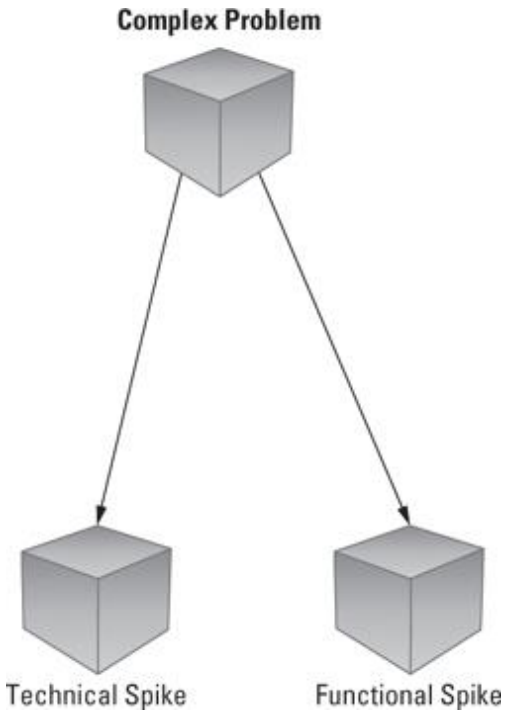


- Provides a powerful visual mean
- The physical size of index cards **forces a text length limit**, requiring the writer to articulate their ideas in just a sentence or two. This helps keep user stories **small and focused**.
- Arrange them by **feature** (or theme), **by time** or **iteration to help evaluate dependencies, understand logical sequencing.**

Spikes

- Spikes are a special type of story used to drive out risk and uncertainty.
- Spikes may be used for basic research to familiarize the team with a new technology or domain.
- The story may contain significant technical risk, and the team may have to do some research to gain confidence in a technological approach.

Technical and Functional Spikes



- **Technical Spikes:**

Used to research various technical approaches in the solution domain. For example, a technical spike may be used to determine a build-versus-buy decision, to evaluate potential performance of a new user story, to evaluate specific implementation technologies.

- **Functional spikes:**

Used whenever there is significant uncertainty as to how a user might interact with the system. Functional spikes are often best evaluated through some level of prototyping, whether it be **user interface mock-ups**, **page flows**, etc.

Spikes

Some user stories may require both types of spikes.

Example:

As a consumer, I want to see my daily energy use in a histogram so that I can quickly understand my past, current, and projected energy consumption.

In this case, a team might create two spikes:

1. *Technical spike: Research how long it takes to update a customer display to current usage, determining communication requirements, bandwidth, and whether to push or pull the data.*

2. *Functional spike: Prototype a histogram in the web portal and get some user feedback on presentation size, style, and charting attributes.*

Guidelines for Spikes

- Spikes should be the exception not the rule
- A spike story should be reserved for the more critical and larger unknowns.
- Like other stories, spikes are put in the backlog
- Spike results are different from a story, because they generally produce information, rather than working code.
- A spike may result in a decision, prototype, proof of concept, or some other partial solution to help drive the final results.
- The output of a spike is **demonstrable**, both to the team and to any other stakeholders.

Project Estimation

- Estimating provides substantial value added for several reasons:

Determining cost
Establishing prioritization
Scheduling and commitment

- In traditional project estimating:

Use a **work breakdown structure** to identify every task, estimate each task, add the tasks up, build a Gantt chart, and predict the cost and schedule.

Estimating Scope with Story Points

- A **story point** is an **integer number** that represents an aggregation of a number of aspects, each of which contributes to the potential "**bigness**" of a story:

- *Knowledge*: Do we understand what the story does?
- *Complexity*: How hard is it to implement?
- *Volume*: How much of it is there? How long is it likely to take?
- *Uncertainty*: What isn't known, and how might that affect our estimate?

- Story points are **unit-less** but numerically relevant (that is, a **two-point story** should expect to take **twice as long** as a one-point story).

Exercise Part 1: Relative Estimating

Think about the relative "bigness of things."

St. Bernard

Exercise: Assign "dog points" to each of the following types of dog to compare their "bigness"

6	Labrador Retriever
~~2~~	Dachshund
7	Great Dane
~~4~~	Terrier
7	German Shepherd
1	Poodle
8	St. Bernard
~~3~~	Bulldog

German Shepherd

Labrador Retriever

Bulldog

Terrier

Dachshund

Poodle

Great Dane

Relative Estimating

In this simple exercise, teams immediately struggle with ambiguity:

- What does the instructor mean by bigness? Height, weight, mass, muscle, bite, attitude?
- What kind of poodle is it? Standard poodle? Toy poodle?



- What scale should we use?

When in doubt, ask the product owner for clarification.

Estimating Real Work with Planning Poker

- Need for some amount of preliminary design discussion is appropriate.
- Estimate the items within a short timebox (maybe 30 minutes).

Rules for Planning Poker

Participants include all agile team members.

The product owner participates but does not estimate (the product owner is briefed on role and content prior to exercise).

Each estimator is given a deck of cards with 0, 1, 2, 3, 5, 8, 13, 20, 40, and 100 as their "value."

(Note: some instructors coach an optional calibrations step: calibrate *size*, such as pick one story that is agreed to be small and agree that the story would be say of size 2 and perhaps pick one more, such as a larger one that would be of size 8.)

For each story, the product owner reads the description.

Questions are asked and answered.

Each estimator privately selects a card representing his or her estimate.

All cards are simultaneously turned over so that all participants can see each estimate.

High and low estimators explain their estimates.

After discussion, each estimator reestimates, and the cards are turned over for a second time.

The estimates will likely converge. If not, the process for that story is repeated until it does.

Repeat until all stories are estimated.

Estimating Real Work with Planning Poker

Subtle aspects built into this estimating technique:

- The estimate comes from the *team as a whole* including **developers and testers** (e.g., fairly easy to code but really hard to test, and the reverse can also be true).

- The range of numbers (Cohn's modified Fibonacci series, that is, 0, 1, 2, 3, 5, 8, 13, 20, 40, 100) is cleverly designed. The lower range (0, 1, 2, 3, 5, 8) is designed to help teams more precisely estimate small things they understand well. However, the gaps in the sequence become larger as the size of the estimate increases, reflecting greater uncertainty.

- The expanded range (20, 40, 100): If the estimates reach this range, the story is too big for an iteration anyway and probably represents a feature or epic.

Estimating Real Work with Planning Poker

- Zero gives the teams a way to ignore small stories that can be implemented in just a few hours.
- A consensus must be achieved before a final estimate is reached. By discussing only the high and low estimates, teams discover assumptions behind the estimates.
- Since the cards are turned over all at once, this prevents individual estimators from being biased by the opinions of others prior to "showing their card."
- It happens pretty fast. Guidance is to allow at most two to five minutes of discussion per item, so a team should be able to estimate ten to twenty stories in an hour or so, which is about the maximum amount of time a team should spend estimating.

Exercise Part 2: Estimating Real Work with Planning Poker

Table 8-1. Sample Class Estimating Backlog

Example Sprint Backlog	
1.	Estimate the pages in the student workbook.
2.	Accurately count the pages in the workbook.
3.	Calculate the square root of 54289 without a computer or calculator.
4.	Add the following ten numbers with a calculator and be certain the answer is correct: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.
5.	Add the following ten numbers without a calculator and be certain the answer is correct: 1, 2, 3, 5, 8, 13, 20, 50, 100.
6.	Introduce yourself to every person in your team, and write down their children's names.
7.	Write a program, without Excel, that accepts 10 numbers from a user and displays the total as each number is entered.
8.	Estimate the cubic volume of the room to within approximately 30%.
9.	Estimate the cubic volume of the room to within approximately 5%.
10.	Estimate the snowfall in Oulu this winter in centimeters, inches, meters, and feet.
11.	Estimate the snowfall in Oulu this winter in centimeters.
12.	Estimate the number of words in the workbook.
13.	Estimate the cubic meters of snowfall in Oulu this winter.
14.	Obtain an accurate count of the number of words in the workbook.

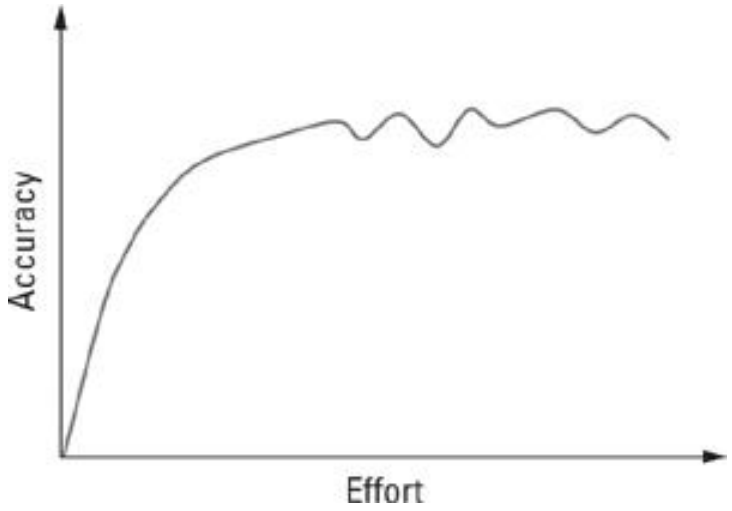
Example: Estimating Real Work with Planning Poker

S. No.	Sprint Exercise Team 1 Estimates	Estimates
1.	Estimate workbook page count.	1
2.	Accurately count workbook pages.	2
3.	Calculate square root of 54289.	5
4.	Add ten numbers with a calculator.	1
5.	Add ten numbers without a calculator.	3
6.	Introduce yourself.	1
7.	Write a program.	5
8.	Estimate cubic volume (within 30%).	5
9.	Estimate cubic volume (within 5%).	8
10.	Estimate snowfall in centimeters, inches, meters, and feet.	3
11.	Estimate snowfall in centimeters.	3
12.	Estimate workbook word count.	3
13.	Estimate the cubic meters of snowfall.	5
14.	Accurately count workbook words.	2
	Totals	**47**

S. No.	Sprint Exercise Team 2 Estimates	Estimates
1.	Estimate workbook page count.	1
2.	Accurately count workbook pages.	3
3.	Calculate square root of 54289.	5
4.	Add ten numbers with a calculator.	1
5.	Add ten numbers without a calculator.	2
6.	Introduce yourself.	1
7.	Write a program.	5
8.	Estimate cubic volume (within 30%).	4
9.	Estimate cubic volume (within 5%).	9
10.	Estimate snowfall in centimeters, inches, meters, and feet.	3
11.	Estimate snowfall in centimeters.	3
12.	Estimate workbook word count.	8
13.	Estimate the cubic meters of snowfall.	3
14.	Accurately count workbook words.	2
	Totals	**50**

How Much Time Should We Spend Estimating?

- More investment in estimating time rarely has a material effect on the actual estimates.
- The results indicate that all three estimates were within a few percentage points of each other.



Team 3

Example Sprint Worklist After 3 Rounds of Estimating	Round 1 Estimates	Round 2 Estimates	Round 3 Estimates
1. Estimate workbook page count.	1	1	1
2. Accurately count workbook pages.	2	2.5	2
3. Calculate square root of 54289.	9	10	9
4. Add ten numbers with a calculator.	2	2	1
5. Add ten numbers without a calculator.	3	2	2
6. Introduce yourself.	1	1	1
7. Write a program.	5	5	6
8. Estimate cubic volume (within 30%).	7	6	6
9. Estimate cubic volume (within 5%).	40	40	40
10. Estimate snowfall in centimeters, inches, meters, and feet.	4	5	3
11. Estimate snowfall in centimeters.	3	4	3
12. Estimate workbook word count.	3.5	3.5	4
13. Estimate the cubic meters of snowfall.	4	3	3
14. Accurately count workbook words.	2	2	2
Totals	86.5	87	83

Estimating Caution: A Story within a Story

- The estimates for "estimate the cubic volume of the room":

- The first two team's estimates for measuring the cubic volume of the room were quite similar:

  - **Team 1**: 5 (within 30%) and 8 (within 5%)
  - **Team 2**: 4 (within 30%) and 9 (within 5%)

- However, Team 3 was **40 (within 5%)**.

Why the big difference?

Estimating Caution: A Story within a Story

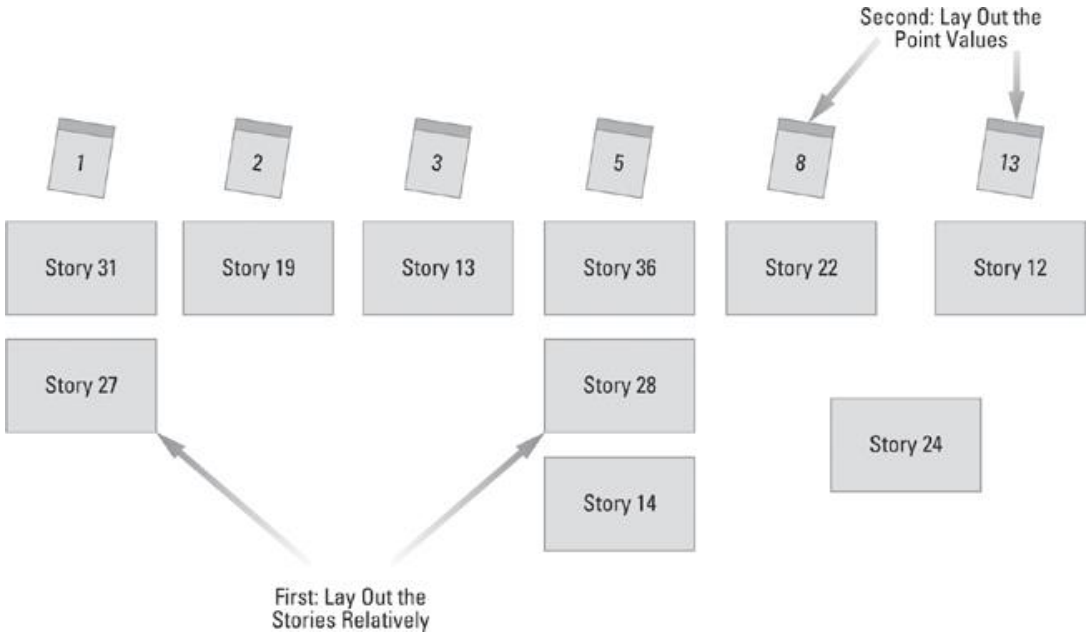
- Simply, the three teams were in **two different rooms**. Teams 1 and 2 were in a **modest-sized, cubic conference room with low ceilings**. Team 3 was in a much larger space with high vaulted ceilings and **a very complex geometry**.

The moral is as follows:

- Before you compare team estimates for theoretically comparable user stories, you must first understand what kind of room (software platform, programming languages, new team versus experienced team, computing resources, legacy versus green-field development, and so on) each team is in.

Tabletop Relative Estimation

- Requires face-to-face communication.
- The team discusses each story in the backlog and places the story on the table in a size position relative to other stories—small stories to the left, bigger stories to the right.
- **Stories of about the same size are stacked in columns.**
- It is expected that stories are shuffled after being discussed.



Tabletop Relative Estimation

- Each story can be seen with respect to all the other stories.
- The stories aren't really estimated yet; they are just placed in relative sizes.
- To create the actual estimates, points can be assigned to columns.
- Visualization of the entire iteration enhances the team's understanding of the work ahead.

From Scope Estimates to Team Velocity: Establishing Velocity

- A team's velocity is simply **how many points** the team can complete in a **standard iteration**.
- The shaded areas represent stories that the team was unable to complete in the timebox. Team 1 completed 28 story points in their iteration, and team 2 completed 32. In other words, team 1's velocity is 28 points/iteration, and team 2's velocity is 32 points per iteration.

S. No.	Sprint Exercise Team 1 Actuals	Actuals
1.	Estimate workbook page count.	1
2.	Accurately count workbook pages.	2
3.	Calculate square root of 54289.	-
4.	Add ten numbers with a calculator.	1
5.	Add ten numbers without a calculator.	3
6.	Introduce yourself.	1
7.	Write a program.	5
8.	Estimate cubic volume (within 30%).	5
9.	Estimate cubic volume (within 5%).	-
10.	Estimate snowfall in centimeters, inches, meters, and feet.	-
11.	Estimate snowfall in centimeters.	-
12.	Estimate workbook word count.	3
13.	Estimate the cubic meters of snowfall.	5
14.	Accurately count workbook words.	2
	Totals	**28**

S. No.	Sprint Exercise Team 2 Actuals	Actuals
1.	Estimate workbook page count.	1
2.	Accurately count workbook pages.	3
3.	Calculate square root of 54289.	5
4.	Add ten numbers with a calculator.	1
5.	Add ten numbers without a calculator.	2
6.	Introduce yourself.	1
7.	Write a program.	-
8.	Estimate cubic volume (within 30%).	-
9.	Estimate cubic volume (within 5%).	-
10.	Estimate snowfall in centimeters, inches, meters, and feet.	3
11.	Estimate snowfall in centimeters.	3
12.	Estimate workbook word count.	8
13.	Estimate the cubic meters of snowfall.	3
14.	Accurately count workbook words.	2
	Totals	**32**

Caveats on the Relative Estimating Model

- **Simple and reliable process** that works quite well, subject to some caveats:
- It is **based on historical data** and is predictive only to the extent that the future (new stories) looks like the past (stories already completed).
- It is valid only to the extent that the **team continues to have the same individuals**. If you change the team members (for example, if we doubled the size of team), velocity will change dramatically, but it should stabilize after few iterations.
- A team's **velocity cannot be compared to any other team**. (Imagine if team 1 had used 2 as the smallest story and compared everything to that. Their apparent velocity would be twice as large, but the actual productivity would be the same.)

Increasing Velocity, Be Careful What You Ask For

- The goal is to **continuously increase velocity while improving quality at the same time.**

- The ability to achieve a certain amount of functionality in a time period, **is not a complete measure of productivity**.

- **Velocity** is only a tool by which teams manage and measure themselves. If management attempts to use velocity as a measure of team performance, the team will respond in one of three ways:

1. **Continuously improve the team's true productivity and agility in all aspects**
2. **Cut back on quality**
3. **Simply increase the size of the estimates**.

From Velocity to Schedule and Cost: Estimating Schedule

If we know **size** and **velocity**, we can calculate **how long it will take to complete a story**.

Estimating cost: simply take the average burdened cost for a team and divide it by their velocity. That provides the cost per story point *for that team*. Then when the team estimates an arbitrary backlog, just multiply the cost per story point for that team by the total estimate for the backlog.

Problems with Estimating using story points

1. It isn't so easy to understand by the team, and it's even less easy to understand by their **outside stakeholders.**
2. It's hard to get started. Until teams have done a few iterations, they have no idea how to predict what they can accomplish.
3. Getting to schedule and cost estimates is very indirect. You have to work through relative estimates, establish velocity, and so on, and you have to understand the burden cost of each individual team, before you can translate a story point into a cost.
4. Teams occasionally struggle to adjust their velocity based on the availability of team members. For example, if a team member is only part-time for a sprint or a key resource is not available for a period, what is the anticipated velocity then?
5. Team velocities are not normalized. It's not unusual for one small team to have a velocity of 40 points per iteration, while a team twice that size has a velocity of half that. That makes for some pretty uncomfortable discussions.

Estimating with Ideal Developer Days (IDDs)

A unit for estimating the size of product backlog items based on how long an item would take to complete if:

- **It were the only work being performed,**
- **there were no interruptions,**
- **and all resources necessary to complete the work were immediately available.**

The reason the estimates are called "ideal" developer days is that the team typically deprecates their capacity for planning, demos, management meetings, and other team and company overhead items.

Estimating with Ideal Developer Days (IDDs)

- With IDDs, the team returns to a **more traditional way** to estimate their work.

- The team looks at each story, discusses it with respect to the same complexity factors and then estimates **how many IDDs** it will take to do the story.

There are many advantages to this method:

- Teams have always done it that way.
- It's far easier to understand and explain.
- It's easy to adjust velocity for sick leave, vacations, training, and so on.

Estimating with Ideal Developer Days

However, it has **a disadvantages** as well:

- Teams tend to get caught up when estimating in times. It's too tangible and too meaningful. They feel they have to get it right.
- It's far more personal and can be politically loaded. One developer might say a story takes two days, another four. Either could be correct—for them—but again, more interesting discussions result. And these discussions are not likely to be supportive of the team spirit.
- **Given these disadvantages, in balance, we prefer the relative estimating model.**

A Hybrid Model

Teams can proceed in large part with the relative estimating model. But we add **two simple rules**:

1. Estimate the smallest story, that can be **done by one person in about a day, as a 1.**

2. Only **8 IDDS** per team member per two-week iteration. This leaves about 20% for planning, demoing, company functions, training, and other overhead.