Software Engineering and Architecture
Requirements & Planning with User Stories

Olivier Liechti
HEIG-VD
olivier.liechti@heig-vd.ch

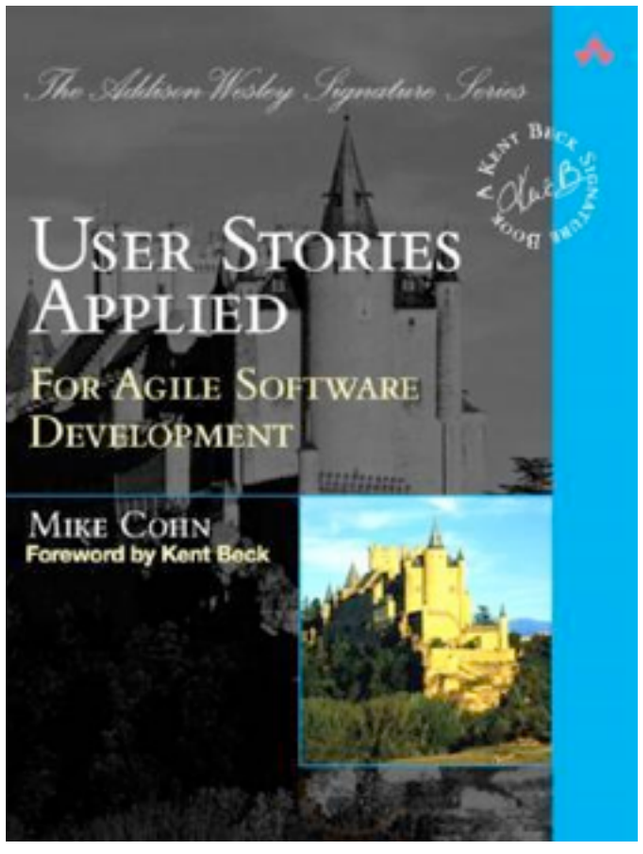
MSE | MASTER OF SCIENCE IN ENGINEERING

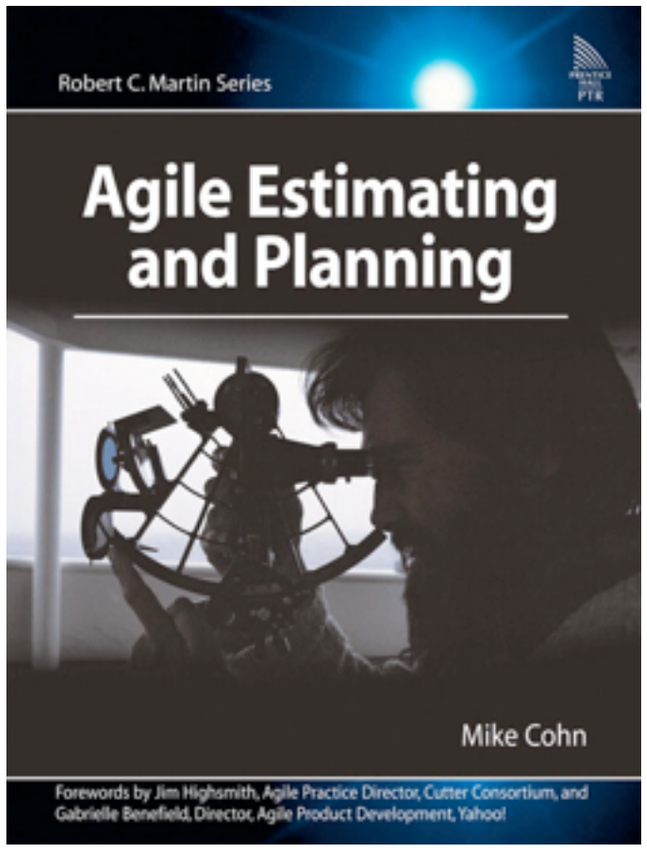
Planning

Date		Theory	Practice
20.09.12	agile	Introduction to agile & scrum	
27.09.12	agile	Agile development tools	Agile development tools
04.10.12	agile	**User Stories**	**Guest speaker (Lotaris)**
11.10.12	agile	The Product Owner	Presentations Agile Development Tools
18.10.12	agile	Lean Development	Guest speaker (Octo)
01.11.12	agile	Continuous Delivery	Lab
08.11.12	architecture	Development in and for the Cloud	Guest speaker (Cloudbees)
15.11.12	architecture	Behavior Driven Development (BDD)	BDD lab
22.11.12	architecture	BDD lab	BDD lab
29.11.12	evolution	Software Evolution	Guest speaker (SonarSource)
06.12.12	evolution	OO Reengineering	Presentations BDD
13.12.12	evolution	Intro to Repository Mining Lab	Repository Mining Lab
20.12.12	evolution	Repository Mining Lab	Repository Mining Lab
10.01.13	architecture	Domain Specific Languages	Lab
17.01.13	architecture	The design of APIs	Guest speaker (Evrythng)
24.01.13	evolution	Repository Mining Presentations	Repository Mining Presentations

User stories

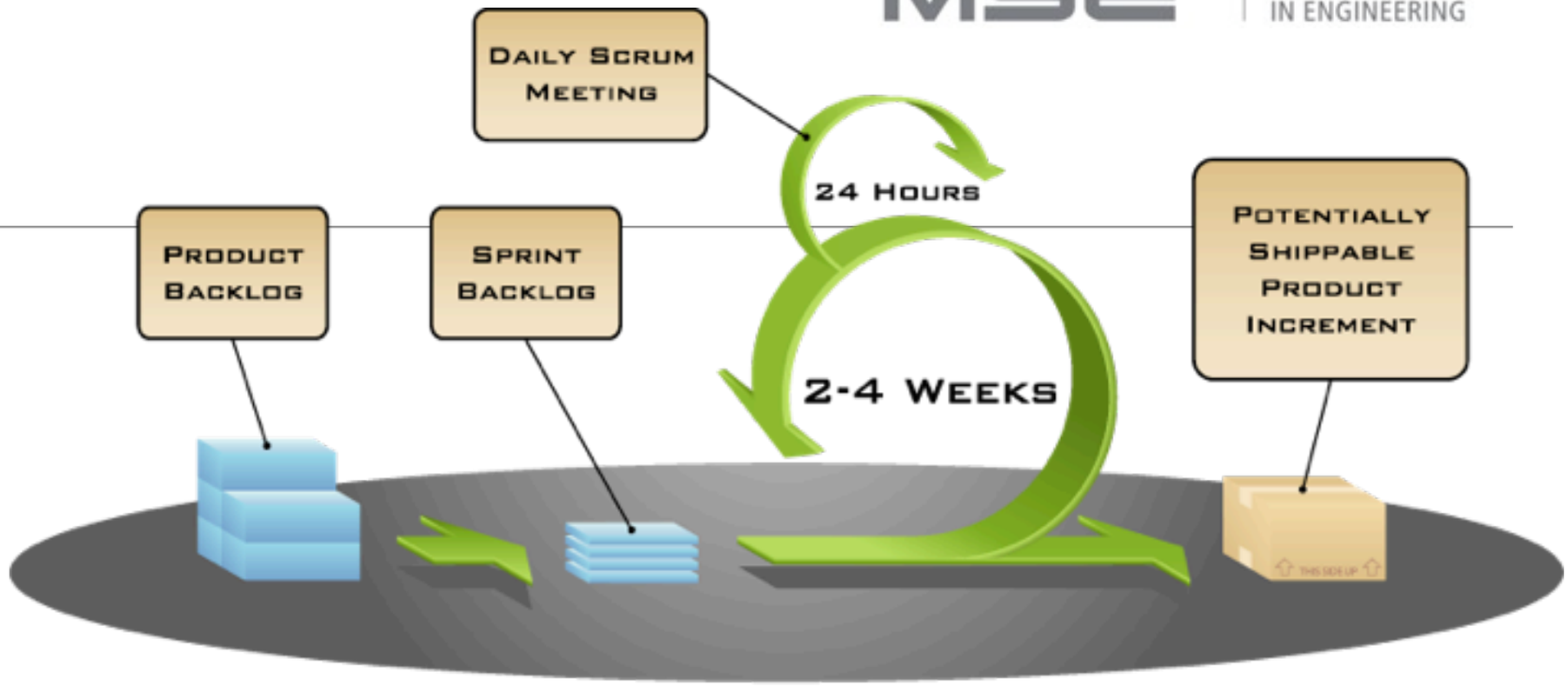
1. What are user stories, how do they support **requirements management**?
2. How can we use stories for **planning** work in an agile way?
3. How do we **estimate effort** with user stories?





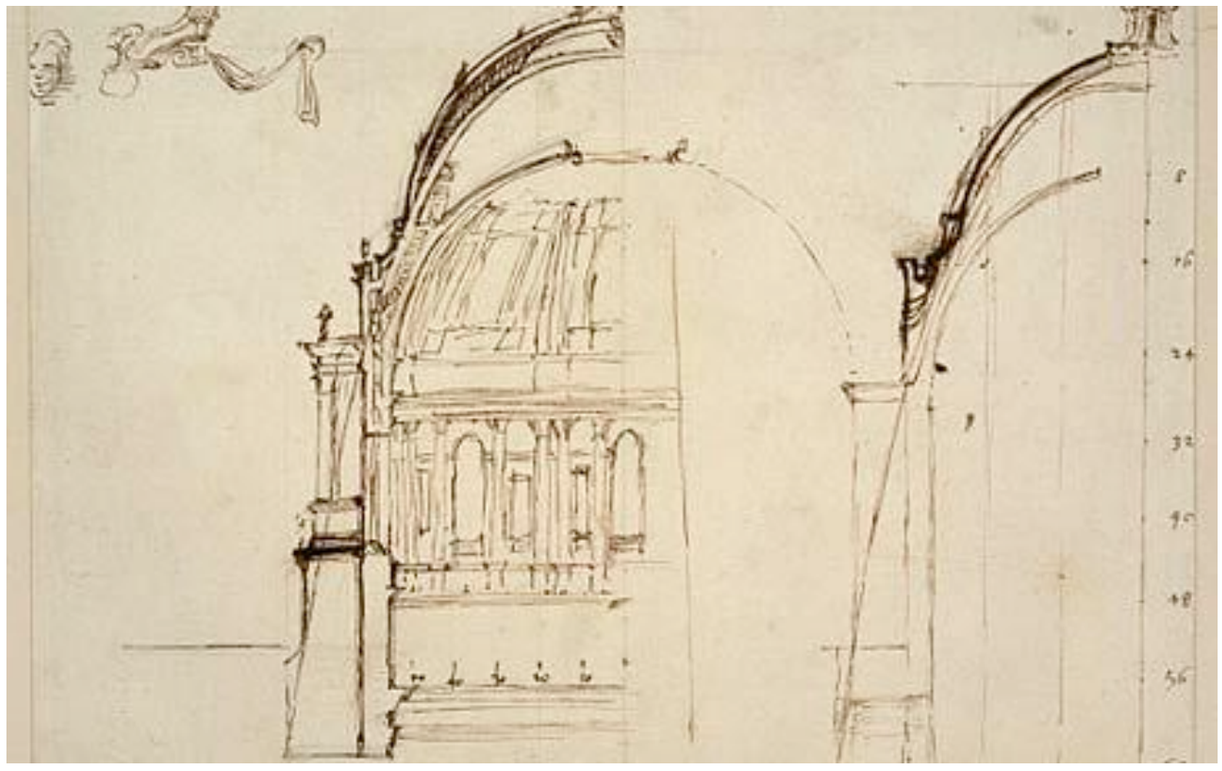
http://www.mountaingoatsoftware.com/

Scrum



Backlog item	Estimate
Allow a guest to make a reservation	3
As a guest, I want to cancel a reservation.	5
As a guest, I want to change the dates of a reservation.	3
As a hotel employee, I can run RevPAR reports (revenue-per-available-room)	8
Improve exception handling	8
…	30
…	50

1. Managing requirements with user stories



Requirements are a communication problem

- **Written requirements**
  - can be well **thought through**, reviewed and edited
  - provide a permanent **record**
  - are more **easily shared** with groups of people
  - **time consuming** to produce
  - may be less relevant or **superseded over time**
  - can be easily **misinterpreted**
- **Verbal requirements**
  - instantaneous **feedback** and clarification
  - information-packed **exchange**
  - easier to clarify and gain **common understanding**
  - more easily **adapted** to any new information known at the time
  - can **spark ideas** about problems and opportunities

Managing requirements in an “agile” way

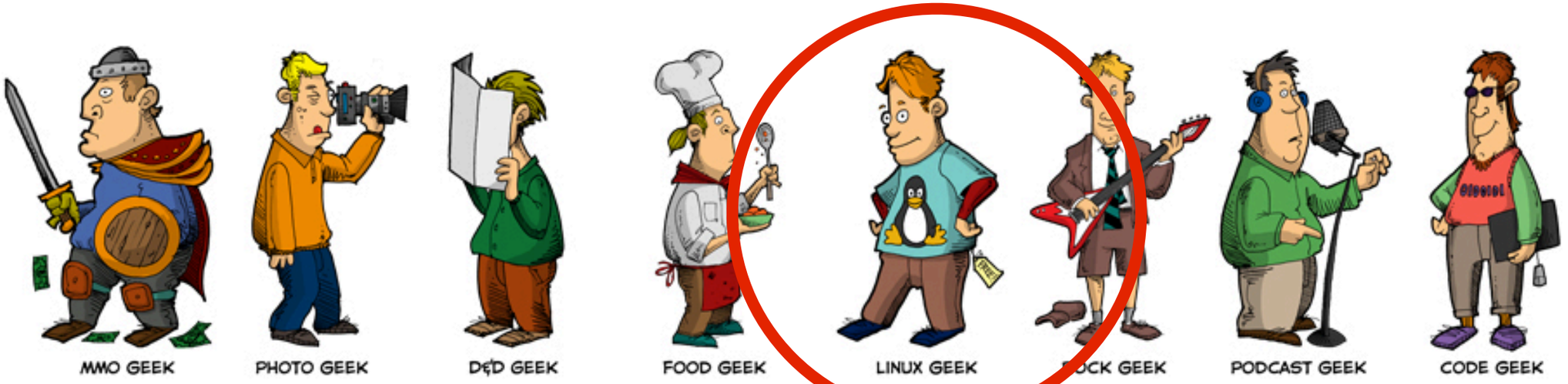
- User Stories seek to **combine the strengths** of written and verbal communication.
- Those who want the software must **communicate** with those who will build it.
- **Note**: it is not because you use user stories as the main artifact for requirements management that you cannot / should not write detailed specifications for some aspects of the system!

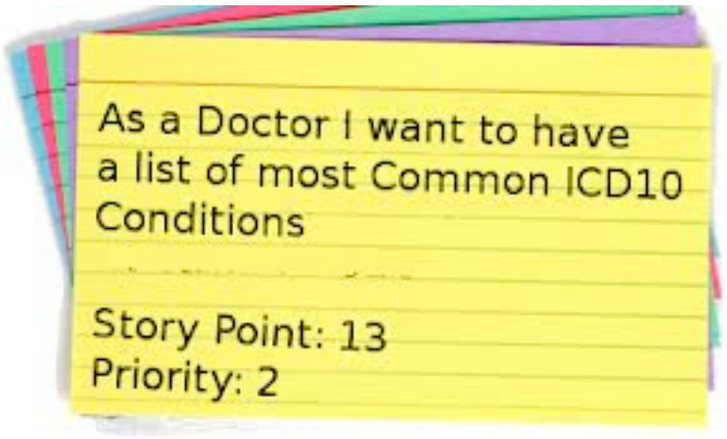
As a student, **I want to** check my grades, **so that** I can choose the lectures I have to repeat

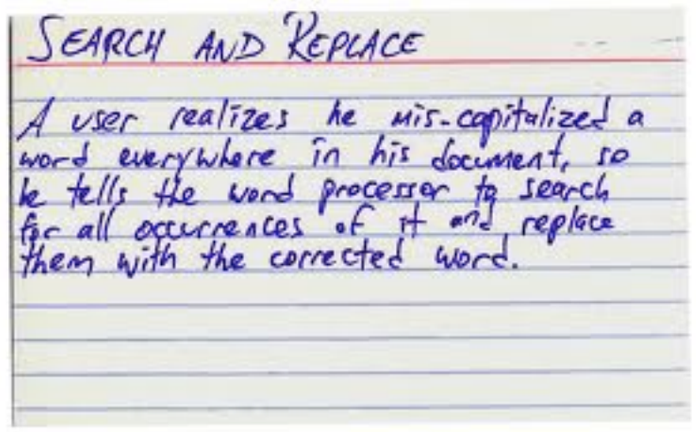
As a frequent flyer, **I want to** rebook a past trip, **so that** I save time booking trips I take

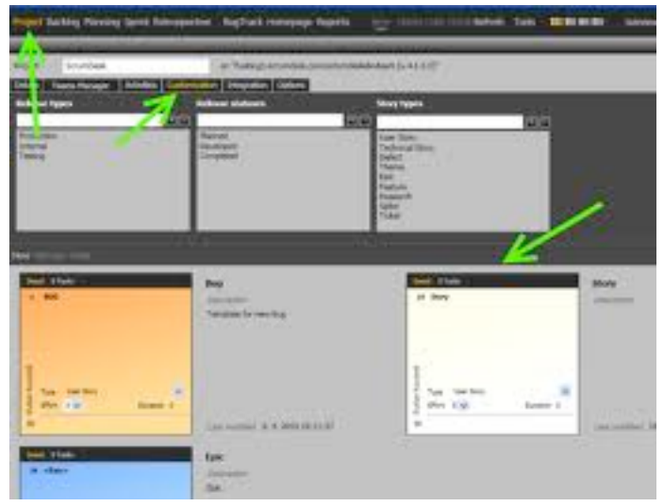
What is a user story?

- A **concise**, **written** description of a piece of functionality that will be **valuable** to a user (or customer) of the software.
- A story is defined with the basic template: “As a <**user role**>, I want to <**goal**> so that <**benefit**>”.
- User roles are important and need to be specific (avoid “As a *user*...”).
- **Estimates**, **notes** and **acceptance criteria** often complement the basic template. With physical cards, the two faces can be used.

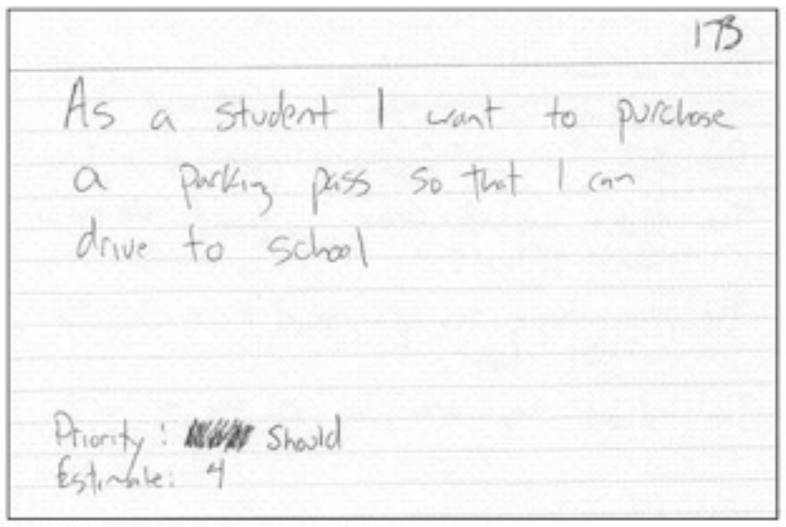




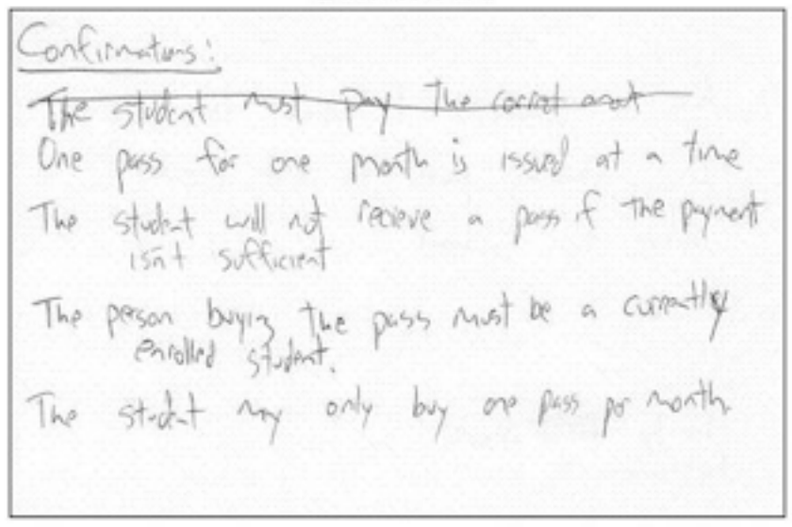


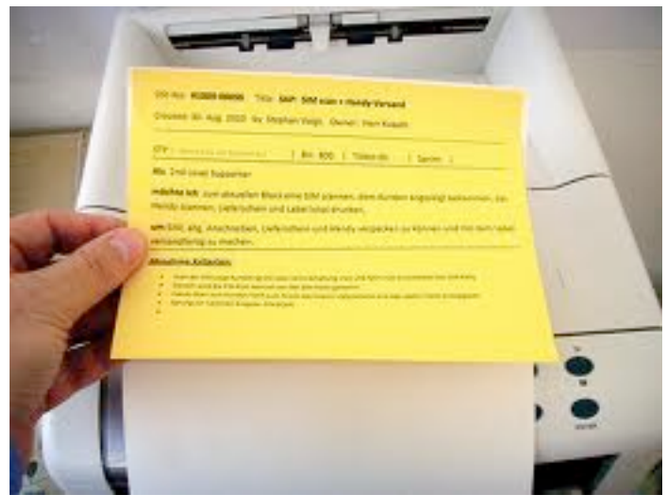


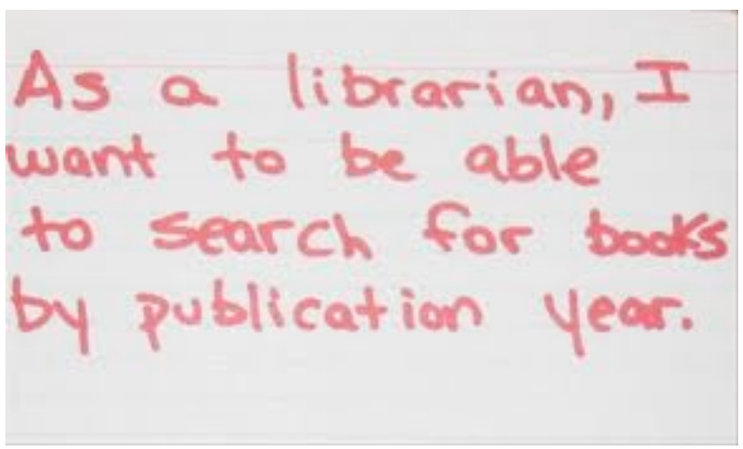
Front of Card



Back of Card







Some aspects of user stories

- User stories **combine written and verbal communication**.
- User stories should describe **features that are of value to the user**, written in a **user's language**.
- User Stories detail **just enough information** and no more.
- **Details are deferred** and captured through **collaboration** just in time for development.
- **Acceptance criteria and test cases** should be written before development, when the user story is written.
- User stories need to be the **right size** for planning. Stories can be defined at **different granularity levels**, depending on the maturity and priority.

User stories are more than paper cards...

- Ron Jeffries' Three Cs
  - **Card** - The paper card on which you write the story text and the minimal information that goes with it. You use the card as a **reminder**, as a **trigger** for discussions, as an **artifact** useful in planning sessions.
  - **Conversation** - What needs to be developed will **emerge** and be specified in **regular** and **frequent discussions** between the customer and the development team. What is produced during a conversation lives in the head of the people. Some of it might be **dumped** into text.
  - **Confirmation** - Both the acceptance criteria and the detailed requirements will be documented in a series of tests (at different levels and of different nature).

    Source: XP Magazine 8/30/01, Ron Jeffries.

Writing Good User Stories

- Bill Wake has come up with the **INVEST** acronym to capture **6 key attributes** of good user stories.
  - **I**ndependent
  - **N**egotiable
  - **V**aluable to users or customers
  - **E**stimatable
  - **S**mall
  - **T**estable

Independent

- From an **iterative planning** point of view, it should be possible to implement stories in **any order**.
- The size of a story (i.e. the estimated time required to deliver it) should not depend on which stories have already been implemented.
- Example:
  - Story 1: As a customer, I can pay with MasterCard.
  - Story 2: As a customer, I can pay with Visa.
  - Story 3: As a customer, I can pay with American Express.
- The problem is that the story that is implemented first will take longer. So, it’s better to write stories like that:
  - Story 1: As a customer, I can pay with one type of credit card.
  - Story 2: As a customer, I can pay with two additional types of credit cards.

Negotiable

- Stories are not written contracts or requirements that a software must implement. They are **short descriptions of functionality**, the details of which are to be negotiated in a **conversation** between the Product Owner and the Team.

- Story cards are **reminders** to have conversation rather than fully detailed requirements.

Valuable to Users or Customers

- Users are the “**end-users**” who interact with the software system. Customers are those who **pay** for the software system.
- In general, user stories should not be written from the point of view of **developers**. For instance, compare the following stories:
  - As a software component, I have to use a connection pool when accessing the database.
  - As user, I don’t see a performance degradation when I use a feature that accesses application data.
- The Product Owner should write the user stories.

Estimatable

- User stories are “units of planning”.
- At the beginning of an iteration, the product owner ranks the stories by priority and the team has to tell how many of these stories can be “done” within the iteration.
- Hence, the team must have an idea of the “size” of each story. As we will see later, the size can be specified in different ways. There are also different techniques for estimating the size of a story.

Small (but not too small!)

- If a story is too big, then it is very difficult to estimate - this makes planning very difficult.
- If stories are too small, then the backlog can become very crowded - this also makes planning very difficult (and people loose visibility in the process).

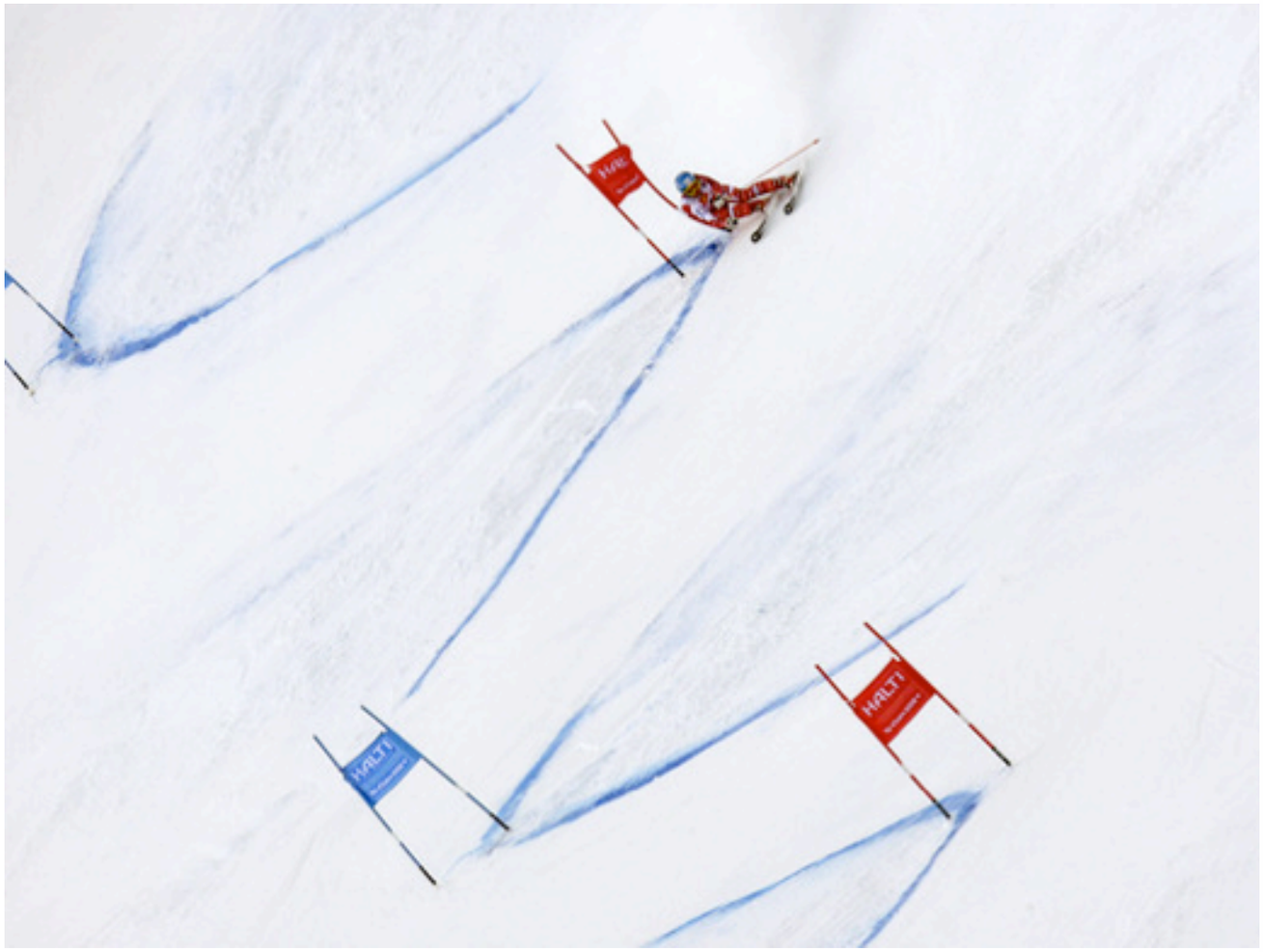
Testable

- Being “**done**” is an important notion when using an agile development methodology.
- A feature (i.e. a story) is only done when it has been **designed**, **implemented**, **tested** and **documented**.
- The **acceptance criteria** should be specified very early on, i.e. when writing the story.

User stories, epics, and themes

- A story is a **fine-grained requirement** (a feature). It corresponds to a feature and, as a rule of thumb, must be implementable during a single iteration.
- Sometimes, we encounter larger “chunks” of functionality that are still quite vague. We know that we may want to implement that functionality at some point, but not immediately. We want to keep track of the functionality, so that we don’t forget about it in the future planning activities.
- These types of stories are called “**epics**”. They are added at the bottom of the back-log. As they become priority items, they are then split in several stories and “bubble up” the product backlog.
- It is also possible to “tag” related stories with the notion of “**theme”**. For instance, several distinct features may be related to “performance” or “security”.

2. Agile Planning



Reminder : the Agile Manifesto

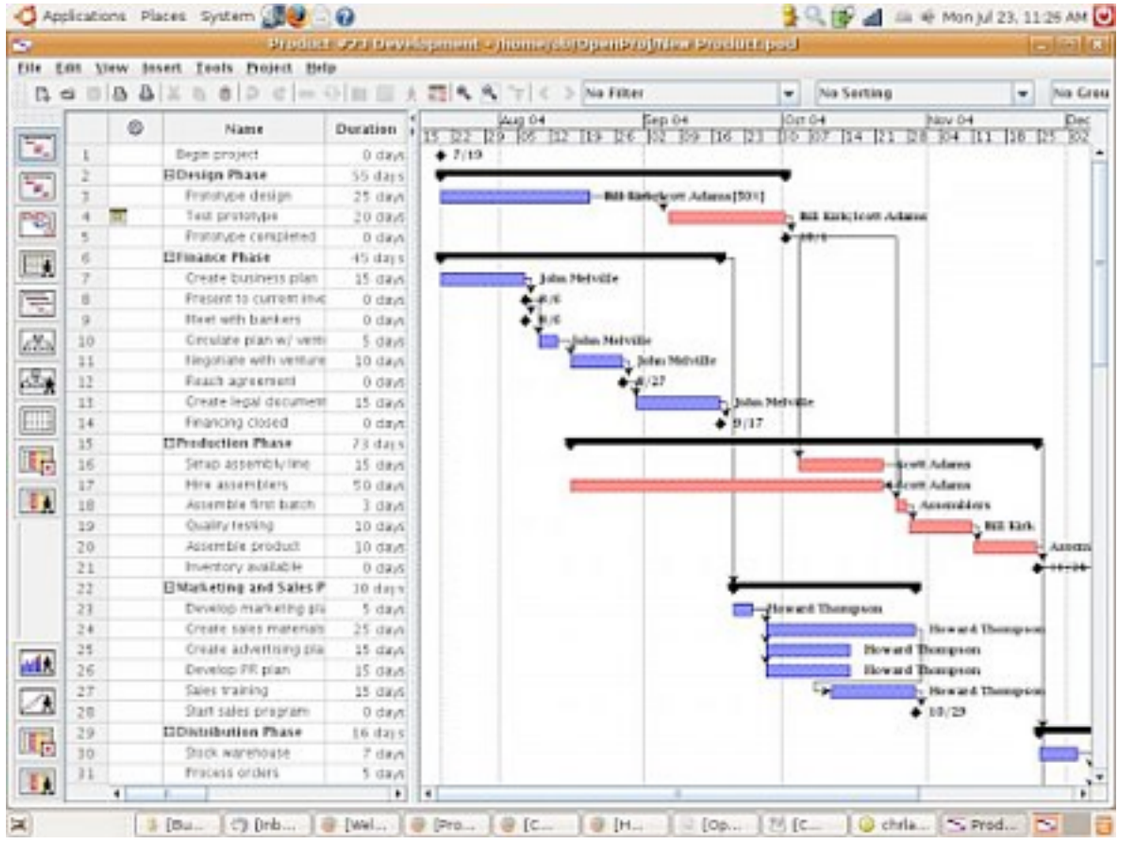
- The authors of the **Agile Manifesto** wrote that they value
  - Individuals and interaction over processes and tools… because great software is made by great individuals.
  - Working software over comprehensive documentation… because that makes it possible to get feedback early.
  - Customer collaboration over contract negotiation… because agile teams would like all parties to the project to be working toward the same set of goals.
  - ***Responding to change over following a plan… because their ultimate focus is on delivering as much value as possible to the project's customer***

What it does NOT mean

- “Agile teams don’t need plans.”
- “If we are sufficiently agile, we don’t need plans— we can always react quickly enough. If we need plans, we are just not agile enough.”

What it means

- **Planning** (the collaborative, ongoing activity) is more important than the **Plan** (the static artifact).

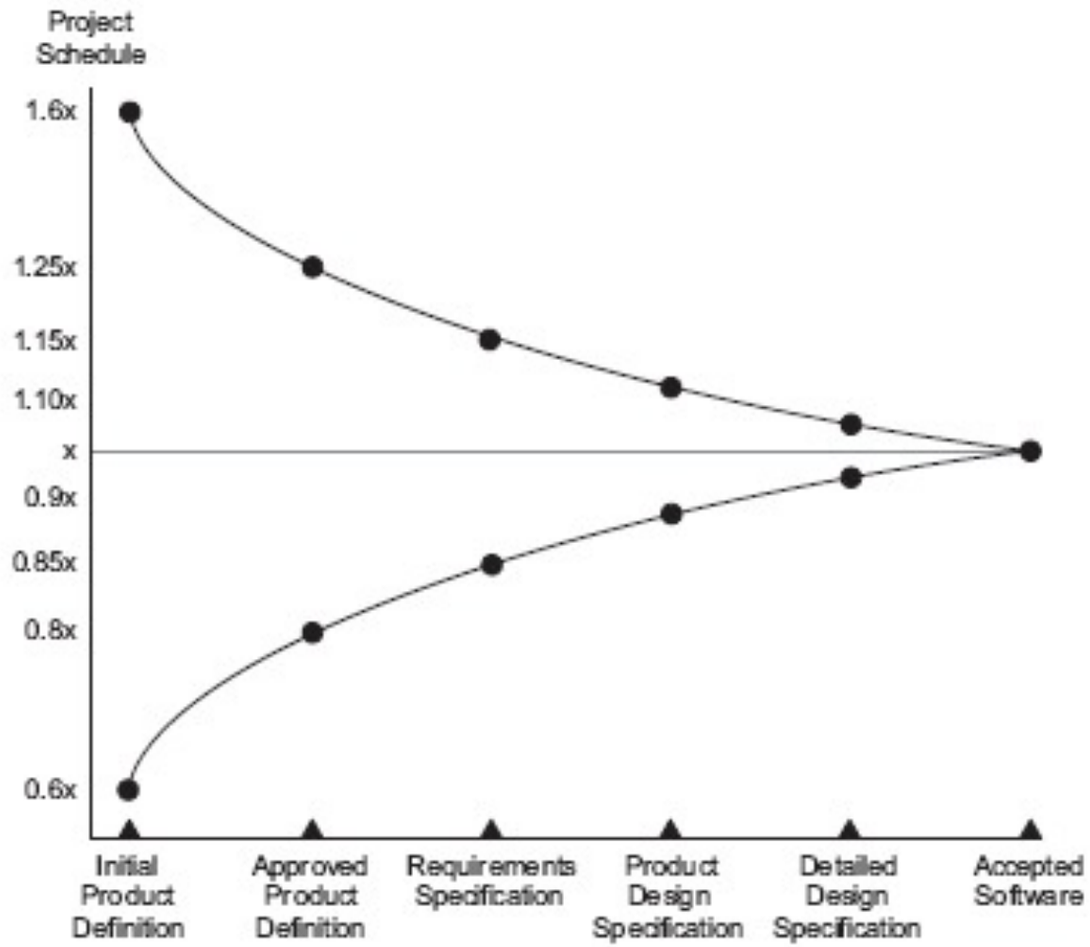


Planning vs plans

- **Plans** are documents or figures; they are **snapshots** of how we **believe** a project might unfold over an **uncertain future**.
- **Planning** is an **activity**, that is not done once and for all. It should happen on a continuous basis.
- **In other words, the initial plan needs to be regularly updated through iterative planning.**
- Agile plans are often (and gladly :o) changed: During a project we learn new things like
  - The customer wants more of this feature or less of that feature
  - The application server is more complex than expected

Planning is hard but important

- **Facts**:
  - Estimating and planning are difficult.
  - It is very hard to get an accurate estimate until late in a project.
- **Why is planning important?**
  - **Organizations** demand estimates (budget, marketing campaigns, product release date, training internal users, and so on)
  - Planning is an ongoing **Quest for Value**: it is an attempt to find an optimal answer to the overall product development question: what should we build?



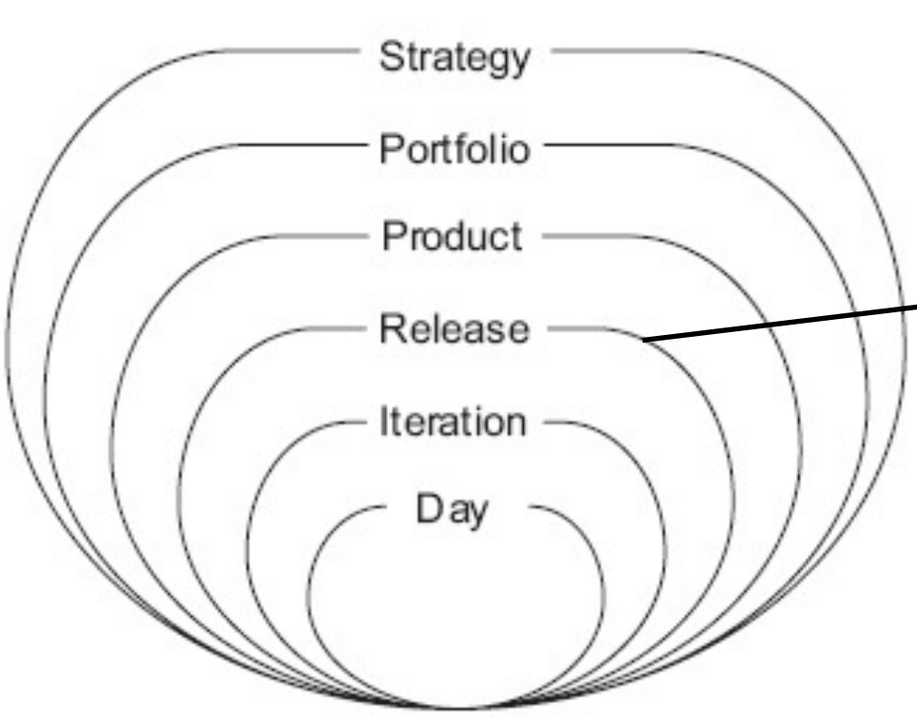
The cone of uncertainty, Boehm 1981

Planning process

- A good iterative planning process:
  - Reduces **risk**
  - Reduces **uncertainty**
  - Supports better **decision making**
  - Establishes **trust**
  - Conveys **information**
- **A good plan is one that is sufficiently reliable that it can be used as the basis for making decisions about the product and the project.**

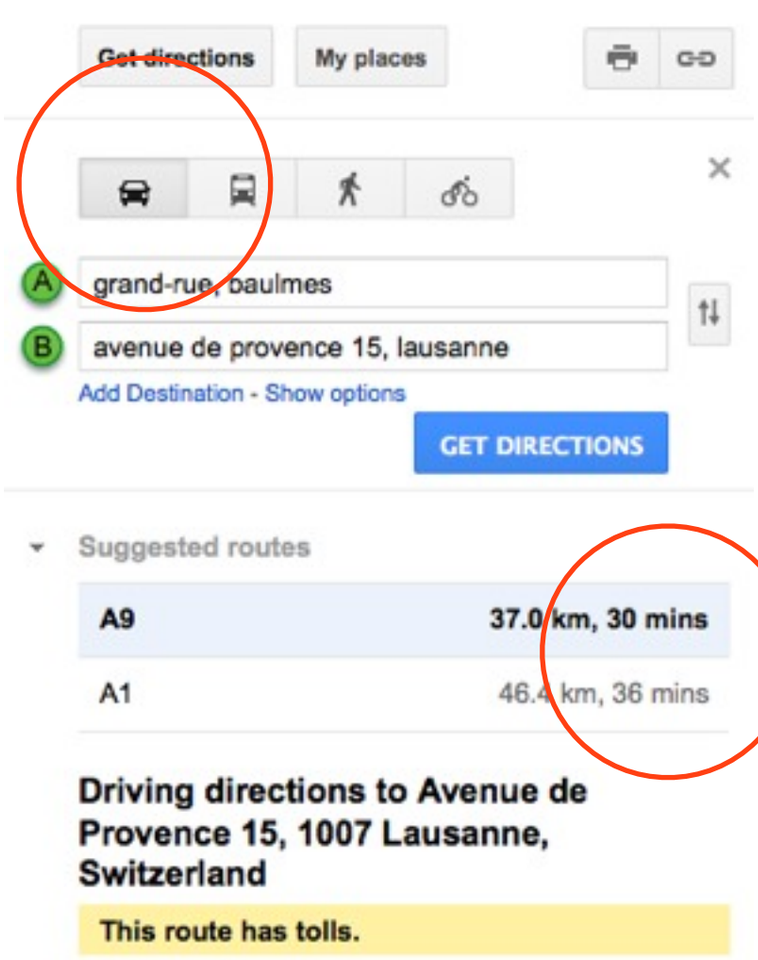
Multiple levels of planning

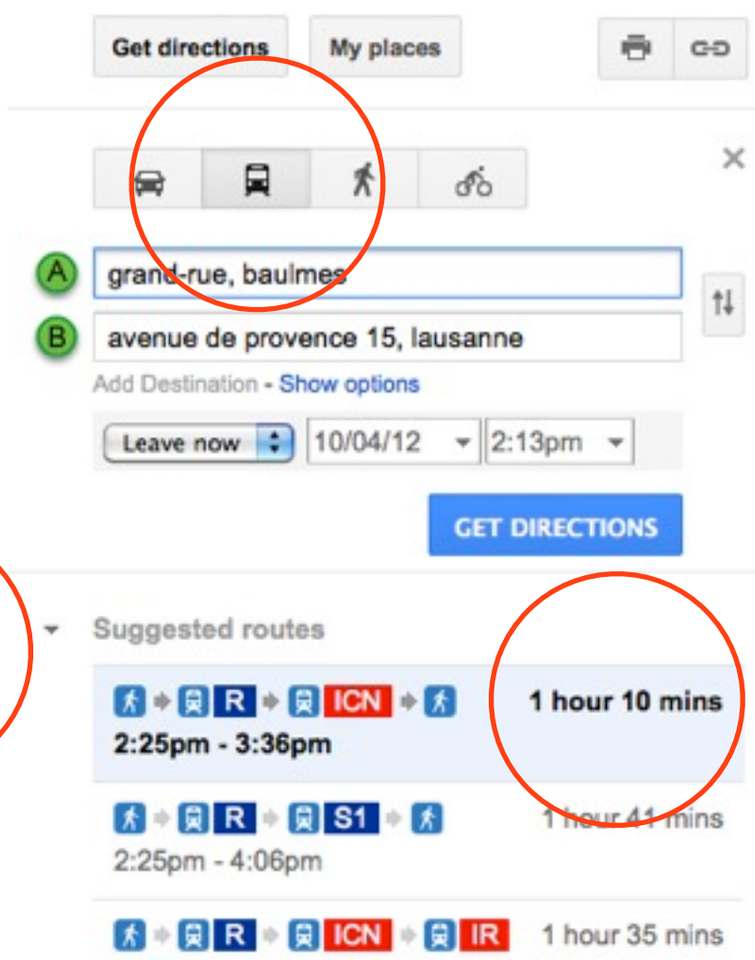
- Planning is a **process** of **setting** and **revising goals** that lead to a **longer-term objective**.
- The planning onion

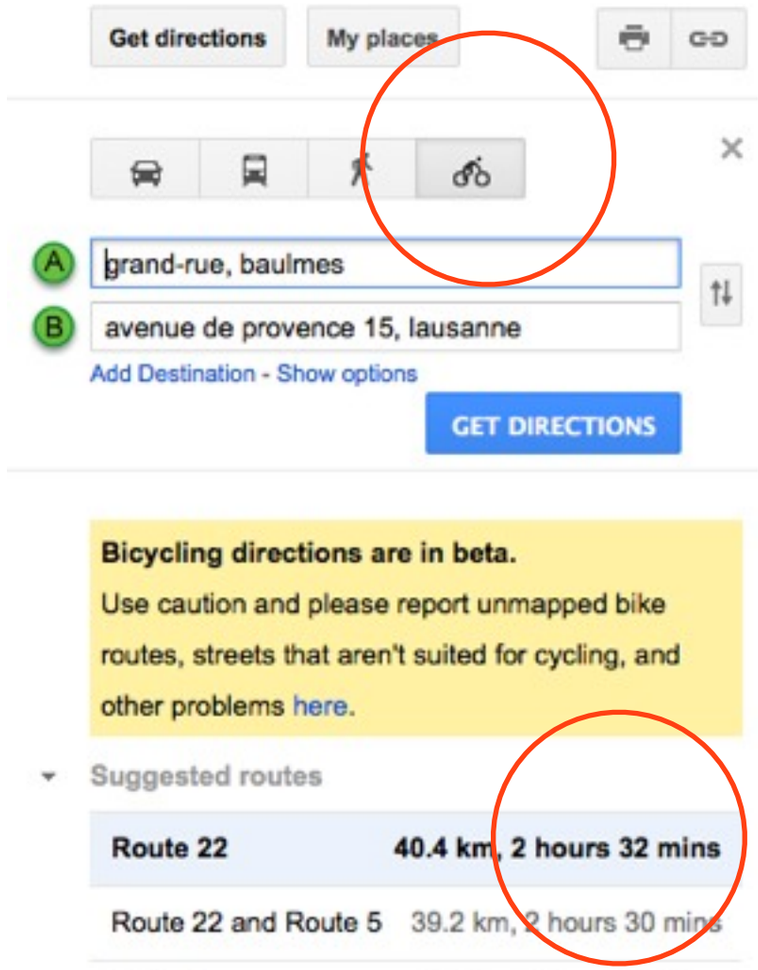


Agile teams plan at the innermost three levels

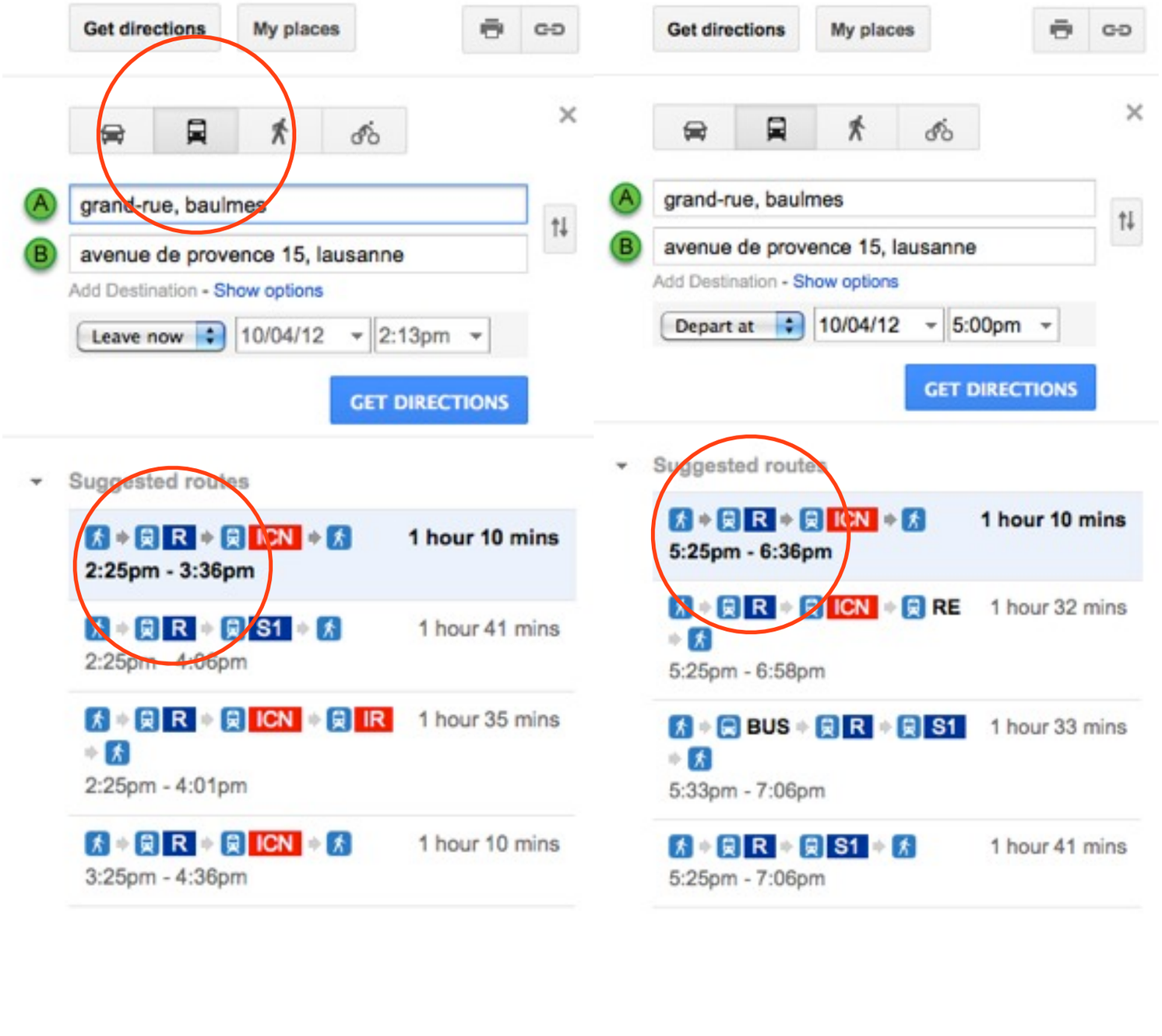
High-level planning & decision making

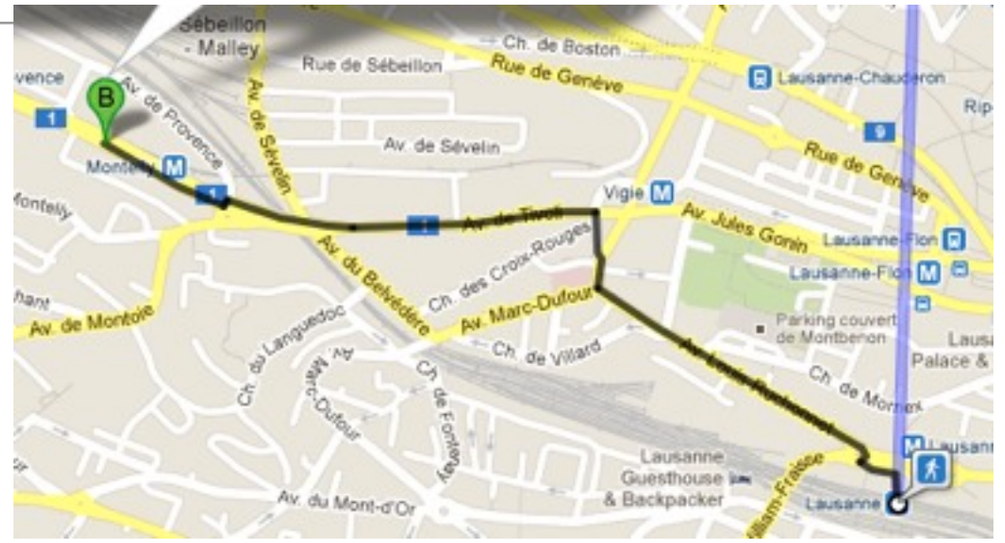


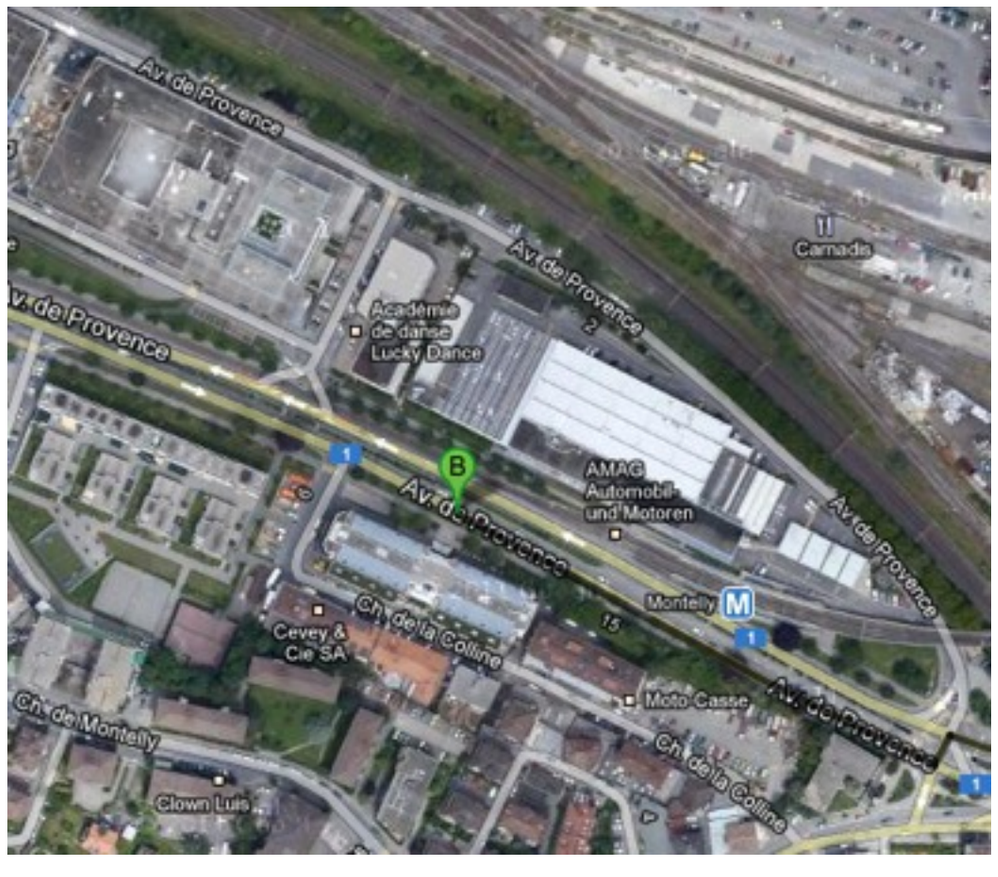




Detailed & updated planning







An agile approach to planning

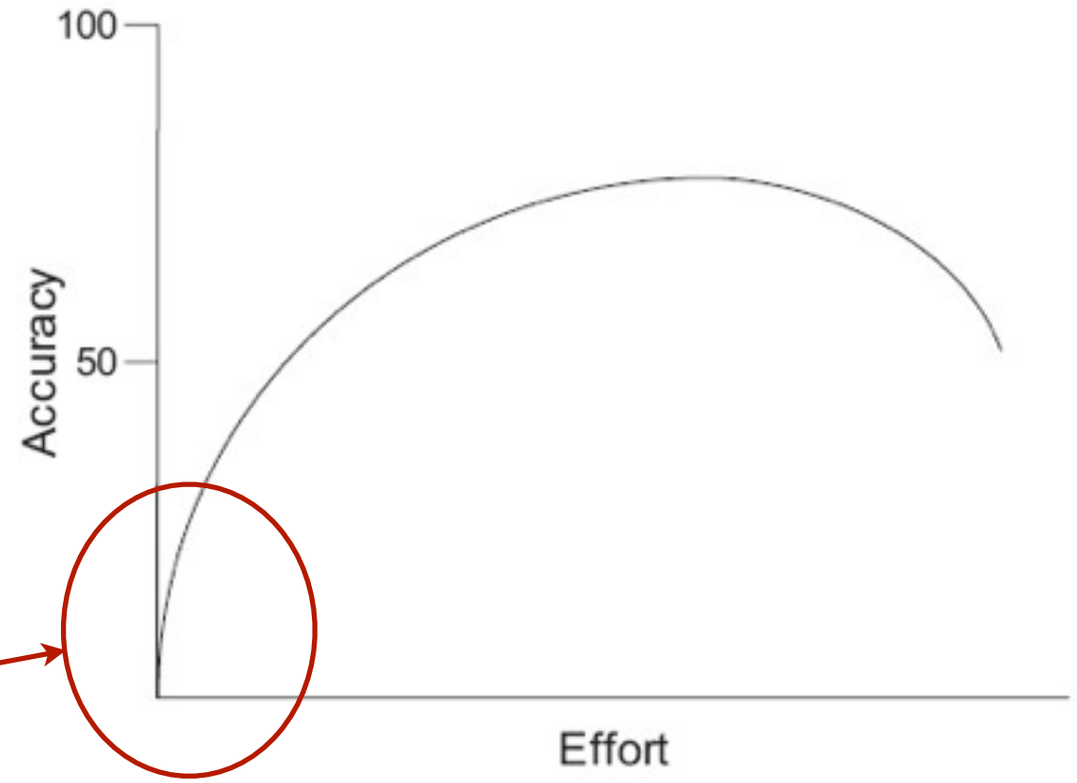
- **Key idea**: A **project** rapidly and reliably generates a **flow** of useful new **capabilities** and new **knowledge**.
- New capabilities are delivered in the product.
- New knowledge is used to make the product the best it can be
  - New product knowledge helps us know more about what the product should be.
  - New project knowledge is information about the team, the technologies in use, the risk, and so on.
- **Failing to plan to acquire new knowledge leads to plans built on the assumption that we know everything necessary to create an accurate plan.**
- Is this assumption true in the world of software development?

3. Estimating

http://www.flickr.com/photos/bb_matt/306544780/sizes/m/#cc_license

Techniques for Estimating

- “Prediction is very difficult, especially about the future” — Niels Bohr, Danish physicist.
- The more effort we put in something, the better the result. Right?
- Agile teams choose to be closer to the left of the Effort-Accuracy-Graph [Cohn06]



http://www.mountaingoatsoftware.com/system/hidden_asset/file/15/aep_sample.pdf

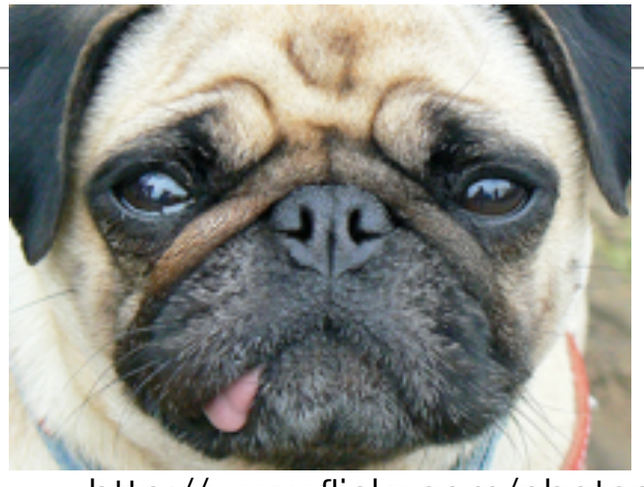
Let’s try to estimate...

- What is the weight of a giraffe?

http://www.flickr.com/photos/badjonni/527455832/sizes/m/#cc_license

Let’s estimate

- **What is the weight of these animals:**
  - a giraffe
  - an elephant
  - a fly
  - a cow
  - a rhinoceros
  - a dog
  - a tyrannosaurus
  - a cat
  - a frog
  - a horse
  - a lion
  - a bull



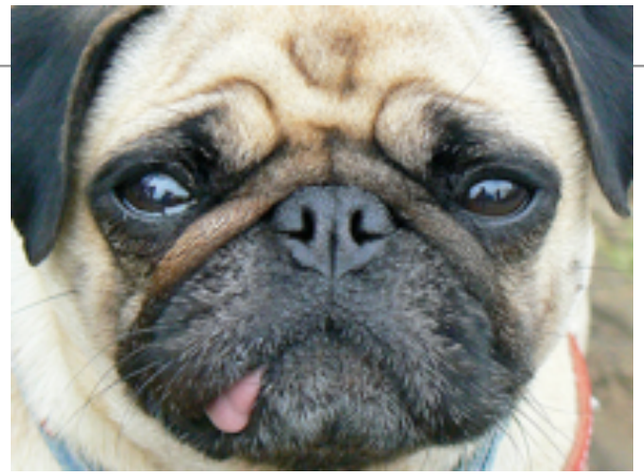
http://www.flickr.com/photos/e3000/2104850919/sizes/m/#cc_license

http://www.flickr.com/photos/badjonni/527455832/sizes/m/#cc_license

http://www.flickr.com/photos/digitalart/2101765353/sizes/s/#cc_license

Observations

- It is easier to estimate in relative than in absolute terms.
- For instance, even if I don't know how much a dog, a giraffe and an elephant weight, I may guesstimate that an elephant is about twice as heavy as a giraffe, and that a giraffe is as heavy as 30 dogs.
- The more animals I consider, the more correct the relative weights are likely to be.
- If I know that a dog weights "5 points", then I can infer that a giraffe is about "150 points" and an elephant is about "300 points".
- If I then measure the absolute weight of one (or ideally several) animals, I can get a pretty good idea of the absolute weight of all animals.
- In agile terms, translating relative values into absolute values is based on the notion of velocity.



http://www.flickr.com/photos/e3000/2104850919/sizes/m/#cc_license

http://www.flickr.com/photos/digitalart/2101765353/sizes/s/#cc_license

Techniques for estimating

- **Absolute** vs. **relative** estimation
- Selecting a **unit** (story points, ideal days, etc.)
- Selecting a **scale**
- **Collecting metrics over time** - computing the velocity of the Team

Estimating with “story points”

- **Key ideas:**
  - estimating relative size is easier than estimating absolute size;
  - story points are not a direct measure of time (vs. hours or days);
  - story points measure the relative time to develop a set of features.
- **Story points are not a direct measure of time:**
  - If we estimate that a particular user story has a weight of “10”, we mean that it will require twice the effort of another story that has a weight of “5”.
  - Iteration after iteration, we look at how many “story points” can be delivered in one iteration.
  - This gives us the velocity of the Team.
  - We can use that value to estimate when we will be finished with a release (or whether we should drop some features from the release).

Estimating with “ideal days”

- Key ideas:
  - in a typical day, how much time can a Team Member really spend on the project-related tasks (vs. coordination, administrative tasks, etc.)?
  - How do you take this overhead time in your estimates?
  - An ideal day represents the number of hours that could theoretically be devoted to the project, if there was nothing else to do.
- How does it work:
  - Estimates are done in ideal days
  - Collection of metrics over time tell us how many ideal days fit within one week (elapsed time)
  - We can thus determine how many iterations are needed to complete a given set of features.

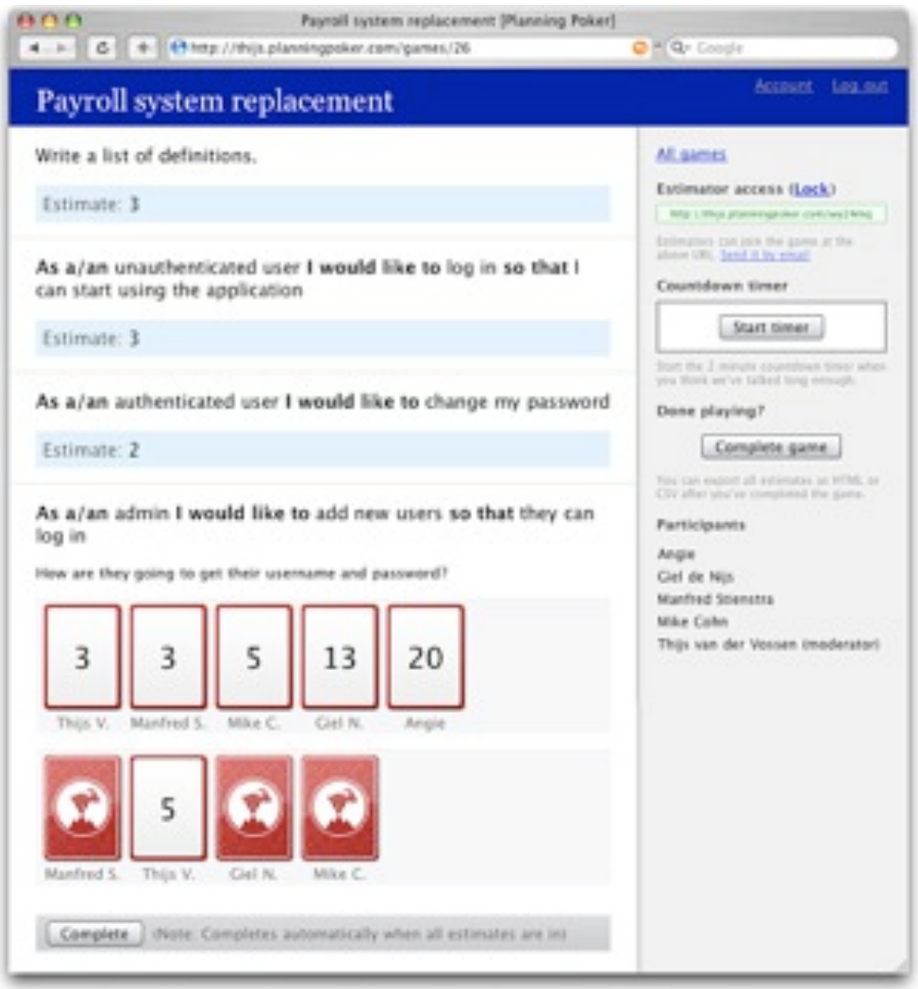
The Planning Poker

- **An iterative approach to estimating**
  - Each estimator is given a deck of cards, each card has a valid estimate written on it (number of story points, ideal days, etc.).
  - The customer/product owner reads a story, which is then briefly discussed.
  - Each estimator selects a card with his or her estimate for the “size” of the story.
  - All participants turn their cards at the same time, so that everyone sees all estimations.
  - The team discusses the differences, which is useful to identify things that were not obvious and to remove ambiguities.
  - People redo estimations until they converge.

The Planning Poker

- Planning Poker combines expert opinion, analogy, and disaggregation (Grenning J. 2002. Planning Poker, www.objectmentor.com/resources/articles/PlanningPoker.zip)

- Online Planning Poker
  www.planningpoker.com



Summary

- User stories are useful both to **capture requirements** and to **support planning** activities.
- In a user story, the paper or digital "**card**" is only the visible part of the iceberg. The **conversations** that are triggered by the card are very important. The **acceptance criteria** that allow one to confirm that a story is done are also very important.
- **Plans** (as artifacts) are useful, but **planning** (as an iterative and recurrent activity) is much more important.
- An **agile planning process** recognizes that it is not possible to create a perfect plan at the beginning of the project. As **new knowledge** is gathered, both about the **users, product** and the **team**, the plan has to be revised iteratively.
- There are different ways to **size** and **estimate** user stories. By collecting data and observing the team over time, one can get a sense of the "**velocity**" of the team. In other words, it becomes easier to estimate how many stories can be "burnt" during one sprint.