Relationship with Customers:

- Customer Specific:
    - One customer with a specific problem.
    - The customer may be another company with a contractual agreement or a division within the same company.
- Market-based:
    - Selling the product to a general market.
    - In some cases, the product must generate customers.
    - The marketing team may act as a substitute customer.
- Community-based:
    - Intended as a general benefit to some community.
    - Examples include open-source tools and tools for scientific research.
- Hybrid:
    - A mix of the above.

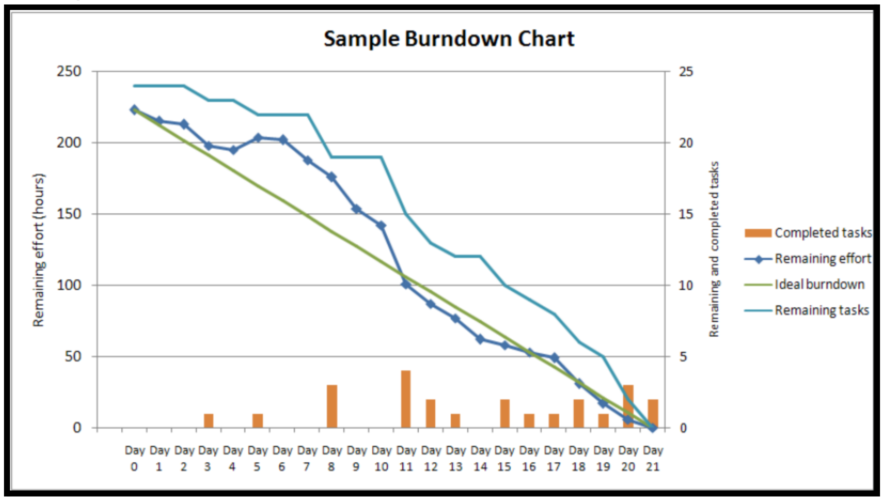
Project Planning:

- **Parts of Project Planning:**
- Given:
    - A list of customer requirements.
    - Examples include a set of use cases or a set of change requests.
- Estimate:
    - How long each one will take to implement (cost).
    - How important each one is (value).
- Plan:
    - Which requests should be included in the next release.
- Complication:
    - Customers care about other stuff, such as security and quality, too.
- **Key Principles of Management:**
- A manager can control 4 things:
    1. Resources - Can get more money, personnel, etc
    2. Time - Can vary the schedule and delay milestones
    3. Product - Can vary the amount of functionality
    4. Risk - Can decide which risks are acceptable
- Approach:
    - Understand the goals and objectives.
    - Understand the constraints.
    - Plan to meet the objectives within the constraints.
    - Monitor and adjust the plan.
    - Preserve a calm, productive, positive work environment.
- **Note:** You cannot control what you cannot measure.
- **Strategies:**
- Fixed Product:
    1. Identify the customer requirements.
    2. Estimate the size of software needed to meet them.
    3. Calculate the time required to build the software.
    4. Get the customer to agree to the cost and schedule.
- Fixed Schedule:
    1. Fix a date for the next release.
    2. Obtain a prioritized list of requirements.
    3. Estimate the effort for each requirement.
    4. Select requirements from the list until there's no more left or you don't think you can work on more tasks.

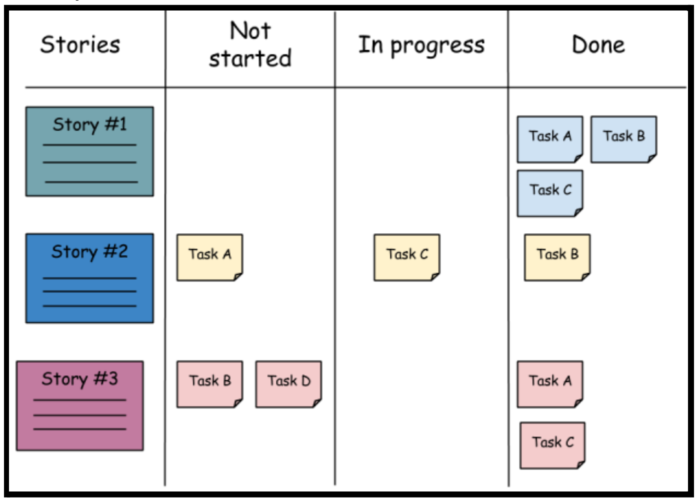
- Fixed Cost:
    1. Agree with the customer on how much they wish to spend.
    2. Obtain a prioritized list of requirements.
    3. Estimate the cost for each requirement.
    4. Select requirements from the list until the “cost” is used up.
- **Estimating Effort - Constructive Cost Model (COCOMO):**
- Predicts the cost of a project from a measure of size (lines of code).
- The basic model is: $E = aL^b$
  E = effort
  a & b = project specific folders
  L = lines of code
- Modelling process:
    1. Establish the type of project (organic, semidetached, embedded, etc). This gives sets of values for a and b.
    2. Identify the component modules and estimate L for each module.
    3. Adjust L according to how much code is reused.
    4. Compute E using the formula above.
    5. Adjust E according to the difficulty of the project.
    6. Compute time using $T = cE^d$, where c and d are provided for different project types, like a and b.
- **Estimating Size - Function Points:**
- Used to calculate the size of software from a statement of the problem.
- Tries to address the variability in lines of code estimates used in models such as COCOMO.
- Basic model is: $FP = a_1I + a_2O + a_3E + a_4L + a_5F$
- Each a_i is a weighting factor for their respective metric.
- I is the number of user inputs (data entry).
- O is the number of user outputs (reports, screens, error messages).
- E is the number of user queries.
- L is the number of files.
- F is the number of external interfaces.
- **Three-Point Estimating:**
- W = worst possible case
- M = Most likely case
- B = Best possible case
- $E = \sum_i \frac{Wi + 4Mi + Bi}{6}$
- **Story Points:**
- We need a common way to compare story sizes.
- It can be hard to find common ground between a programming story and a database management story.
- **Story points** are a relative measure of a feature’s size or complexity.
  They are not durations nor a commitment to when a story will be completed.
  Different teams have different velocities, so they may complete stories at different rates depending on experience.
- A good tool to do the estimation is **planning poker**. It is a series similar to the Fibonacci Series that can be a useful range for story points. Here, each number is almost the sum of the two preceding numbers: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100.

- 0-points estimates are used for trivial tasks that require little effort, though too many zero-pointers can add up.
- Only use numbers within the set and avoid averages. We avoid averages because if a user story turns out to be harder than expected, then the people who picked a higher number will say "I told you" to the people who picked a lower number. The average does not convince other people.
- What happens is this:
  1. A feature is mentioned.
  2. Each person in the team takes a number from the set, but doesn't show/tell anyone else yet.
     They choose the number based on how difficult they think implementing the feature will be.
     A feature with point 0 means that it requires very little effort.
  3. After 3 seconds, everyone shows their number.
  4. If everyone or most people have the same number, it's good.
  5. If everyone or most people have different numbers, then each person has to defend why they picked their number.
     Once the discussion has been carried out, there is a second round of voting.
  6. The process repeats until everyone agrees.
     If people consistently do not agree with one another, then the user story is not a good user story. This is because one of the features of a user story is that it must be estimable.
- Powers of 2 is also an effective tool to do the estimation.
- **Ideal Days:**
- Another unit of measure. It can be used as a transition for teams that are new to agile.
- It represents an ideal day of work with no interruptions (phone calls, questions, broken builds, etc.)
  However, it doesn't mean an actual day of work to finish.
- Tasks are estimated in hours.
  An estimation is an ideal time (without interruptions/problems).
  Smaller task estimates are more accurate than large.
- After all tasks have been estimated, the hours are totaled up and compared against the remaining hours in the sprint backlog. If there is room, the PBI is added and the team commits to completing the PBI.
  If the new PBI overflows the sprint backlog, the team does not commit and
  - the PBI can be returned to the product backlog and a smaller PBI chosen instead or
  - we can break the original PBI into smaller chunks or
  - we can drop an item already in the backlog to make room or
  - the product owner can help decide the best course of action.
- **Tracking Progress:**
- Information about progress, impediments and sprint backlog of tasks needs to be readily available.
- How close a team is to achieving their goals is also important.
- Scrum employs a number of practices for tracking this information:
  1. Task cards
  2. Burndown charts
  3. Task boards
  4. War rooms (standups)

- **Burndown Charts:**
- Example of a burndown chart:



- Indicates how much work has been done in terms of how many user stories have been completed and when.
- Burndown charts are on Jira.
- On the beginning of the sprint, on the vertical axis, is the number of user stories you'd like to implement.
- At the end of the sprint, the number of user stories should be decreased to 0. That means all the stories have been implemented.
- A burndown chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum.
- Typically, in a burndown chart, the outstanding work is often on the vertical axis, with time along the horizontal. It is useful for predicting when all of the work will be completed.
- **Taskboard:**
- Example of a taskboard:

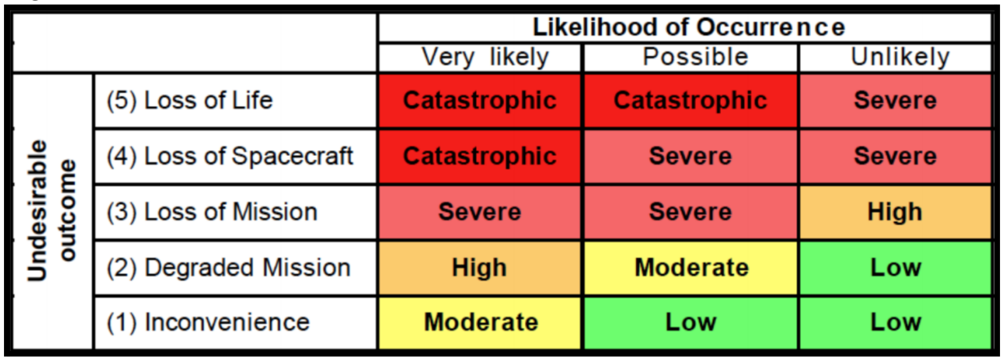


- Both taskcards and taskboards are on Jira.
- The leftmost column are the stories to be implemented and there are 3 columns describing the progress of the tasks.

- In scrum the **task board** is a visual display of the progress of the scrum team during a sprint. It presents a snapshot of the current sprint backlog allowing everyone to see which tasks remain to be started, which are in progress and which are done.
- Simply put, the task board is a physical board on which the user stories which make up the current sprint backlog, along with their constituent tasks, are displayed. Usually this is done with index cards or post-it notes.
- The task board is usually divided into the columns listed below.
  Stories: This column contains a list of all the user stories in the current sprint backlog.
  Not started: This column contains sub tasks of the stories that work has not started on.
  In progress: All the tasks on which work has already begun.
  Done: All the tasks which have been completed.

<u>**Risk Management:**</u>

- **Introduction to Risk:**
- **Risk** is the possibility of suffering loss.
- Risk itself is not bad; it's essential to progress.
- The challenge is to manage the amount of risk.
- **Risk Exposure (RE):**
- RE = Probability of risk occurring * Total loss if risk occurs
- Calculates the effective current cost of a risk and can be used to prioritize risks that require countermeasures.
  I.e. Can help find which countermeasures work best.
- Higher RE means more serious risk.
- **Risk Reduction Leverage (RRL):**
- RRL = $\dfrac{Reduction\ in\ Risk\ Exposure}{Cost\ of\ the\ countermeasure}$
- Calculates a value for the return on investment for a countermeasure and can be used to prioritize possible countermeasures.
- Higher RRL indicates more cost-effective countermeasures.
- **Risk Assessment:**
- Quantitative:
    - Measures risk exposure using standard cost and probability measures.
    - **Note:** Probabilities are rarely independent.
- Qualitative:
    - Create a risk exposure matrix.
    - E.g. for NASA



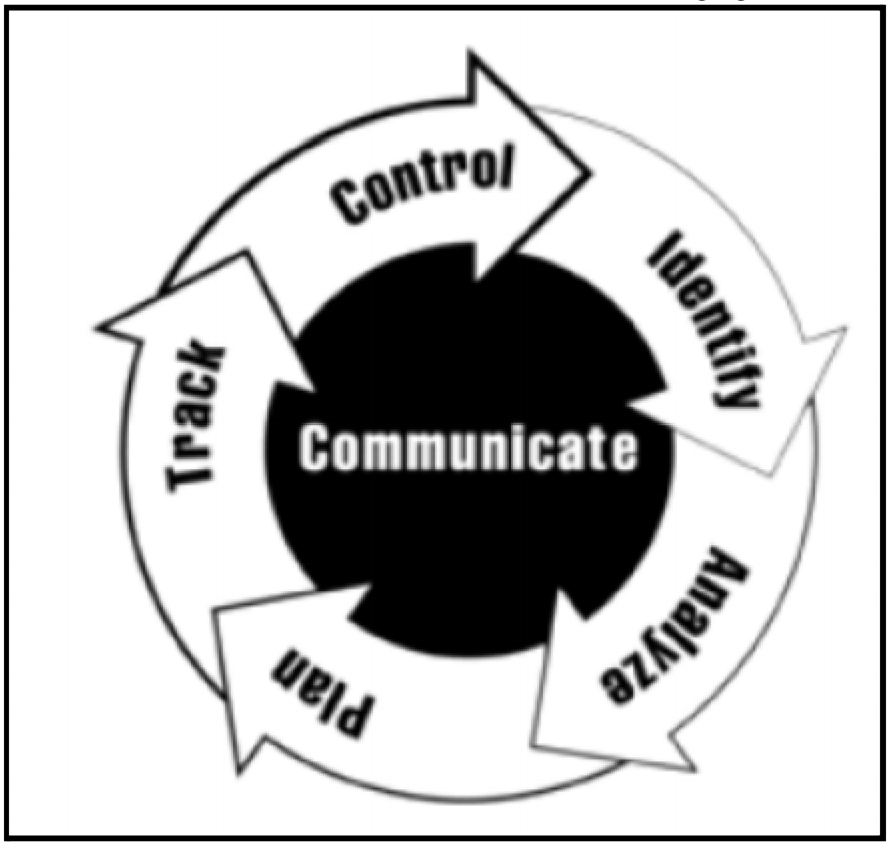
		Likelihood of Occurrence		
		Very likely	Possible	Unlikely
Undesirable outcome	(5) Loss of Life	Catastrophic	Catastrophic	Severe
	(4) Loss of Spacecraft	Catastrophic	Severe	Severe
	(3) Loss of Mission	Severe	Severe	High
	(2) Degraded Mission	High	Moderate	Low
	(1) Inconvenience	Moderate	Low	Low

- **Top Software Engineering Risks With Countermeasures:**

Risks	**Countermeasures**
Personnel Shortfalls	Use top talent. Team building. Training.
Unrealistic schedule/budget	Multisource estimation. Designing to cost. Requirements scrubbing.
Developing the wrong software function	Better requirements analysis. Organizational/Operational analysis.
Developing the wrong user interface	Prototypes, scenarios, task analysis.
Gold plating	Requirements scrubbing. Cost benefit analysis. Designing to cost.
Continuing stream of requirement changes	High change threshold. Information hiding. Incremental development.
Shortfalls in externally furnished components	Early benchmarking. Inspections, compatibility analysis.
Shortfalls in externally performed tasks	Pre-award audit. Competitive designs.
Real-time performance shortfalls	Targeted analysis. Simulations, benchmarks, models.
Straining computer science capabilities	Technical analysis. Checking scientific literature.

- **Principles of Risk Management:**
- Global perspective:
    - View software in the context of a larger system.
    - For any opportunity, identify both potential value and potential impacts of adverse results.
- Forward looking view:
    - Anticipate possible outcomes.
    - Identify uncertainty and manage resources accordingly.
- Open communications:
    - Free-flow information at all project levels.
    - Value the individual voice. Everyone has unique knowledge and insights.
- Integrated management:
    - Project management is risk management.
- Continuous process:
    - Continually identify and manage risks.
    - Maintain constant vigilance.

- Shared product vision:
    - Everyone understands the mission.
    - Focus on results.
- Teamwork:
    - Work cooperatively to achieve the common goal.
- **Continuous Risk Management:**
- Control:
    - Correct for deviations from the risk mitigation plans.
- Identify:
    - Search for and locate risks before they become problems.
    - Use systematic techniques to discover risks.
- Analyze:
    - Transform risk data into decision-making information.
    - For each risk, evaluate:
        - Probability
        - Impact
        - Timeframe
    - Classify and prioritize risks.
- Plan:
    - Choose risk mitigation actions.
- Track:
    - Monitor risk indicators.
    - Reassess risk.
- Communicate:
    - Share information on current and emerging risks.


Control
Identify
Analyze
Plan
Track
Communicate