Scrum Team:

- **Scrum Master:**
    - Educates the team about scrum.
    - Ensures practices are followed.
    - Facilitates problem solving and runs interference for the team.
    - Not a traditional lead or management role.
    - Improves the use of scrum through coaching, facilitation and rapid elimination of any distractions to the team.
    - Ensures the principles behind scrum remain intact even as the company's own implementation evolves.
    - Sometimes scrum may be difficult to follow - the team may want to cut corners to deliver a sprint, and the scrum master needs to remind them about what they must do.
    - Nurture sense of ownership.
- **Product Owner:**
    - Communicates the vision of the product.
    - Maximizes the return on investment (ROI) by prioritizing desirable features.
    - Establishes a shared vision for the product among the customers/developers.
    - Knows what to build and in what order.
    - Owns the product backlog.
    - Creates release plans and establishes delivery dates.
    - Supports sprint planning and reviews.
    - Represents the customers.
- **Developers and Domain Experts:**
    - Everyone from every discipline needed to complete the sprint goals.
    - Members of the team are primarily developers and testers, however in many projects, specialists from other disciplines may also join the team.

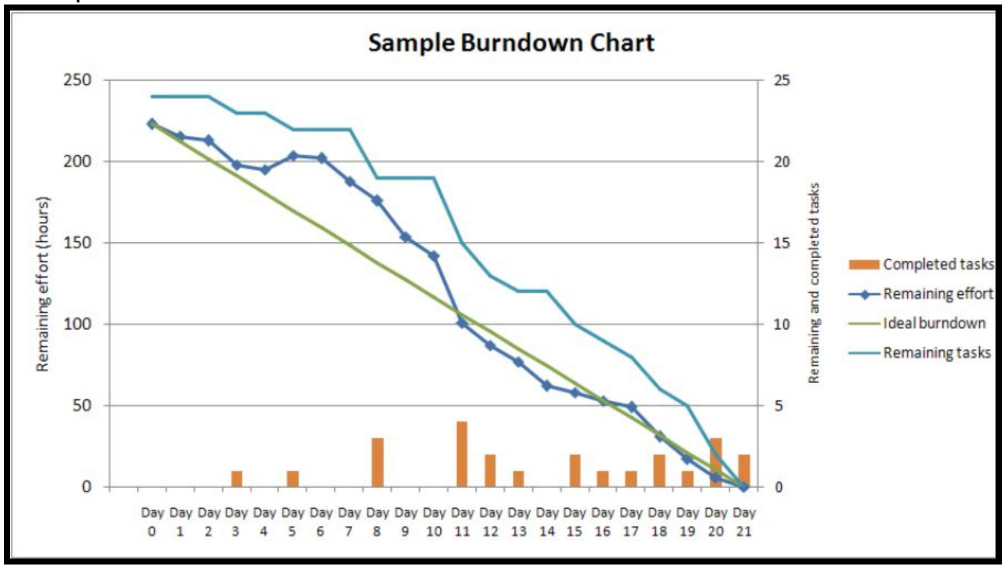
Agile Planning:

- **Sprint:**
- A sprint length is usually 2 - 4 weeks long.
- Here are some factors to consider when determining your sprint length:
    1. **Frequency of customer feedback:**
    - How long can the stakeholders go without seeing progress/giving input?
    2. **Team's level of experience:**
    - New, inexperienced teams may be using a Sequential approach, which is a small waterfall approach while more experienced teams may be using a Parallel approach.
    3. **Time overhead for planning and reviews:**
    - Review and planning require a good chunk of the day regardless of sprint length.
    4. **Ability to plan the entire sprint:**
    - If there is uncertainty about how to achieve the sprint goal a shorter duration is advisable.
    5. **Intensity of the team over the sprint:**
    - Avoid mini-crunch periods
- Traditional software projects place the bulk of project planning at the start of the project, whereas agile spreads project planning out over the entire duration of the project. However, agile teams may actually spend more time overall in planning, but it is not as concentrated.

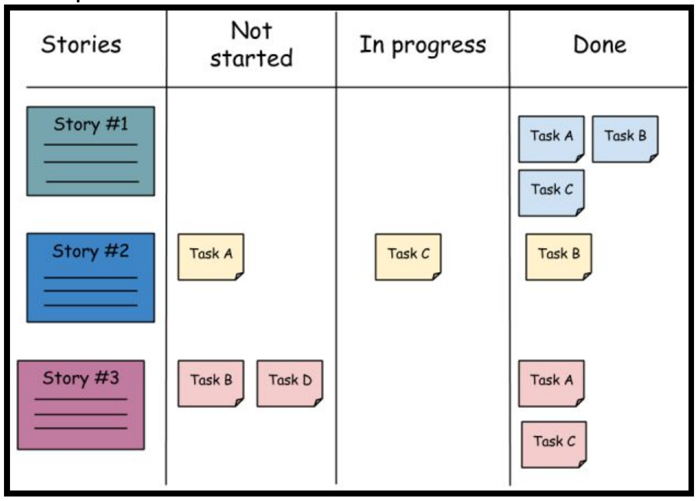
- **Product Backlog:**
- The goal of the product backlog is to:
    1. Prioritize stories so teams are always working on the most important features.
    2. Support continual planning so the plan matches reality instead of being a stale design document.
    3. Improves forecasts so stakeholders can make better decisions about the project.
- Prior to each sprint, the product owner is allowed to change the priorities in the product backlog.
- Decisions can be based on the following criteria:
    1. **Value:** What value does the story add to the player buying the game, helps maximize ROI.
    2. **Cost:** Some features may prove too costly to implement and affect ROI.
    3. **Risk:** Uncertainty about value/cost.
    4. **Knowledge:** If the product owner doesn't have enough information about features to do a proper estimate they can introduce a **spike** to explore it and get more info. A spike is a user story for which the team cannot estimate the effort needed. In such a case, it is better to run time-boxed research, exploration to learn about the issue or the possible solutions. As a result of the spike, the team can break down the features into stories and estimate them. Thus, you may consider a spike as an investment for a product owner to figure out what needs to be built and how the team is going to build it. The product owner allocates a little bit of the team's capacity now, ahead of when the story needs to be delivered, so that when the story comes into the sprint, the team knows what to do.
- **Velocity:**
- **Velocity** is measured by calculating the size of the user stories completed each sprint.
- Changes in velocity are an effective way to see how changes/improvements to the agile process affect the team.
- The raw numbers of Agile velocity will not reveal much; it is the trends that will help you measure and improve efficiency. A misconception about velocity in Agile is that it should be used as an efficiency goal, which is not the intended use case.
- Here is a common mistake that gets made because of this misconception: When a team sees velocity numbers decreasing, they ask "How can we get the number back up to where it was?" This runs the risk of pressuring developers to cut corners in order to achieve a particular velocity goal.
- Instead, when your Agile velocity numbers are trending down, it should signal you to dig deeper into possible inefficiencies that may be causing the downward trend. If you are confident that there aren't any, then you may want to plan for a lower velocity number going forward. This would yield a more accurate budget and timeline.
- Likewise, an increase in Agile velocity numbers should be looked into as well. It could be a sign that the team is moving too fast and quality has declined.
- The best way to use velocity in Agile is to keep it simple and let it guide you to uncover inefficiencies in your process.
- **When to create the estimates:**
- During story workshops or sprint planning meetings.
- Estimates should be a relatively quick process involving the following:
    1. **Expert opinion:** Domain experts can help inform the group about issues and effort required.
    2. **Analogy:** Stories are compared to each other to help determine size to provide better results.
    3. **Disaggregation:** Larger stories may be difficult to estimate, so they can be broken down into smaller ones to help make the estimates more accurate.

- Having to come up with a physical number usually removes any fuzziness about the requirements of the story.
- **Converting story points to hours:**
- Suppose for some reason you track how long every one-story-point story takes a given team to develop. If you did, you'd likely notice that while the elapsed time to complete each story varied, the amount of time spent on one-point stories takes on the shape of the familiar normal distribution.
- A tool used to convert story points to hours is **ideal days**.
    - It is another unit of measure and it can be used as a transition for teams that are new to agile.
    - Represents an ideal day of work with no interruptions such as phone calls, questions, broken builds, etc.
    - Associated with something really so easier to grasp initially.
    - Note that this doesn't mean an actual day of work to finish.
    - It is useful for relative measurements when compared to other stories.
- **Note:** Converting story points to hours is something to be done carefully.
- **Breaking user stories into tasks:**
- Tasks are estimated in hours.
  An estimation is an ideal time (without interruptions/problems).
  Smaller tasks 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)

- **Burn down charts:**
- Example of a burn down 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.

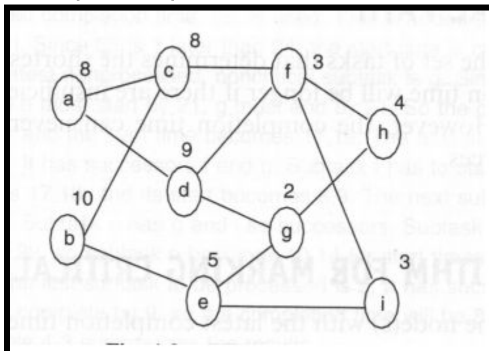
- In scrum the **task board** is a visual display of the progress of the scrum team during a sprint. It presents a snapshot of the current sprint backlog allowing everyone to see which tasks remain to be started, which are in progress and which are done.
- Simply put, the task board is a physical board 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.
- **Daily Scrum Meeting:**
- It is a 15 minute meeting that everyone must attend.
- No sitting down, team stands in a circle and answers the following questions:
  1. What have I done since the last meeting?
  2. What am I going to accomplish between now and the next meeting?
  3. What are the problems or impediments that are slowing me down?
- It is not for solving problems. The Scrum Master must ensure that all side conversations are kept to a minimum. Solving problems happens throughout the rest of the day.
- It can be evolved to meet a specific team's requirements, but the purpose must remain the same (status, commitment, improvement).
- **Sprint Reviews:**
- Occur on the last day of the sprint.
- The team and stakeholders come together to play the game and discuss the work accomplished.
- The product owner accepts or declines the results of the sprint.
- If a feature is declined, the owner will decide if it is returned to the backlog or simply dropped.
- Honesty is crucial.
- Cannot discourage criticism simply because a lot of work was put in.

Critical Path Method (CPM):

- **Critical path** is the sequential activities from start to the end of a user story. Although many user stories have only one critical path, some may have more than one critical paths depending on the flow logic used in the user story implementation.
- A critical path is determined by identifying the longest stretch of dependent activities and measuring the time required to complete them from start to finish.
- If there is a delay in any of the activities under the critical path, there will be a delay of the user story delivery.
- Key steps in critical path method:
  1. **Activity specification:**
     - Break down a user story in a list of activities.
  2. **Activity sequence establishment:**
     - Need to ask three questions for each task of your list.
       1. Which tasks should take place before this task happens.
       2. Which tasks should be completed at the same time as this task.
       3. Which tasks should happen immediately after this task.
  3. **Network diagram:**
     - Once the activity sequence is correctly identified, the network diagram can be drawn.

4. **Identify critical path:**
   - The critical path is the longest path of the network diagram. If an activity of this path is delayed, the user story will be delayed.

- Example of cpm:



TaskID	Time	Dep.	CP
a	8		
b	10		*
c	8	a,b	*
d	9	a	
e	5	b	
f	3	c	*
g	2	d	
h	4	f,g	*
i	3	e,f	

TaskID	Start Time	Comp.Time	CP
a	0,1	8,9	
b	0	10	*
c	10	18	*
d	8,9	17,18	
e	10,14	15,19	
f	18	21	*
g	17,19	19,21	
h	21	25	*
i	21,22	24,25	

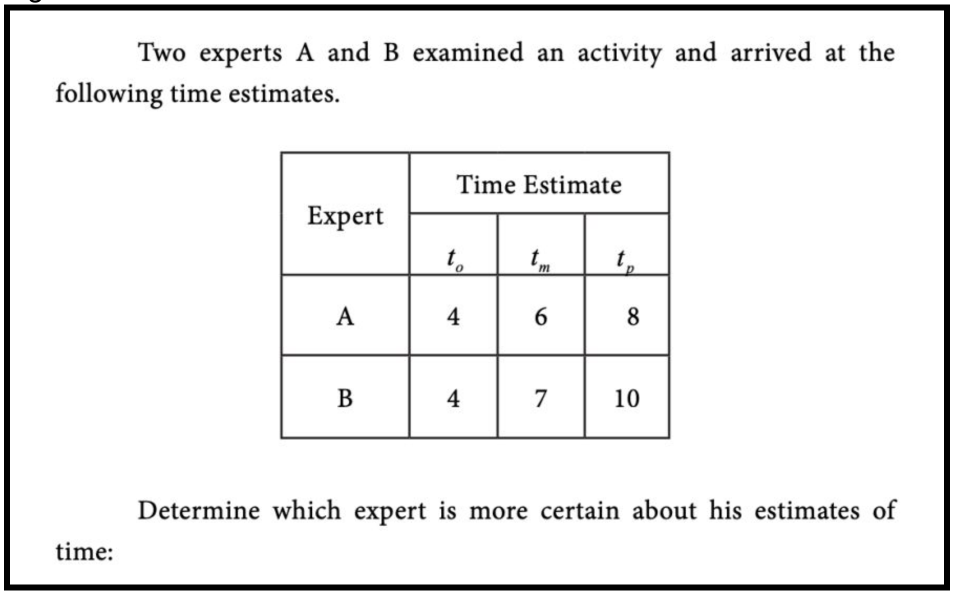
- Tasks on the critical path have to start as early as possible or else the whole project will be delayed. However tasks not on the critical path have some flexibility on when they are started. This flexibility is called the **slack time**.

Program Evaluation and Review Technique (Pert):

- PERT (Program Evaluation and Review Technique) is one of the successful and proven methods used in task scheduling.
- PERT was initially created by the US Navy in the late 1950s. The pilot project was for developing Ballistic Missiles and there have been thousands of contractors involved.
- After PERT methodology was employed for this project, it actually ended two years ahead of its initial schedule.
- At the core, PERT is all about management probabilities. Therefore, PERT involves many simple statistical methods as well.
- Sometimes, people categorize and put PERT and CPM together. Although CPM shares some characteristics with PERT, PERT has a different focus.
- While it is similar to most other estimation techniques, PERT also breaks down the tasks into detailed activities.
- In PERT, we assume that it is not possible to have precise time estimates for each activity and instead, probabilistic estimates of time alone are possible. A multiple time estimate approach is followed here. In probabilistic time estimate, the following 3 types of estimate are possible:
    1. Optimistic time estimate: to.
    2. Most likely time estimate: tm.
    3. Pessimistic time estimate: tp.

  Note: to < tm < tp
- Time estimate is given by: $\boxed{t_e = \frac{t_o + 4t_m + t_p}{6}}$
- The standard deviation for te is given by: $\boxed{\sigma = \frac{t_p - t_o}{6}}$
- The variance is given by: $\boxed{\sigma^2 = \left(\frac{t_p - t_o}{6}\right)^2}$

- E.g.



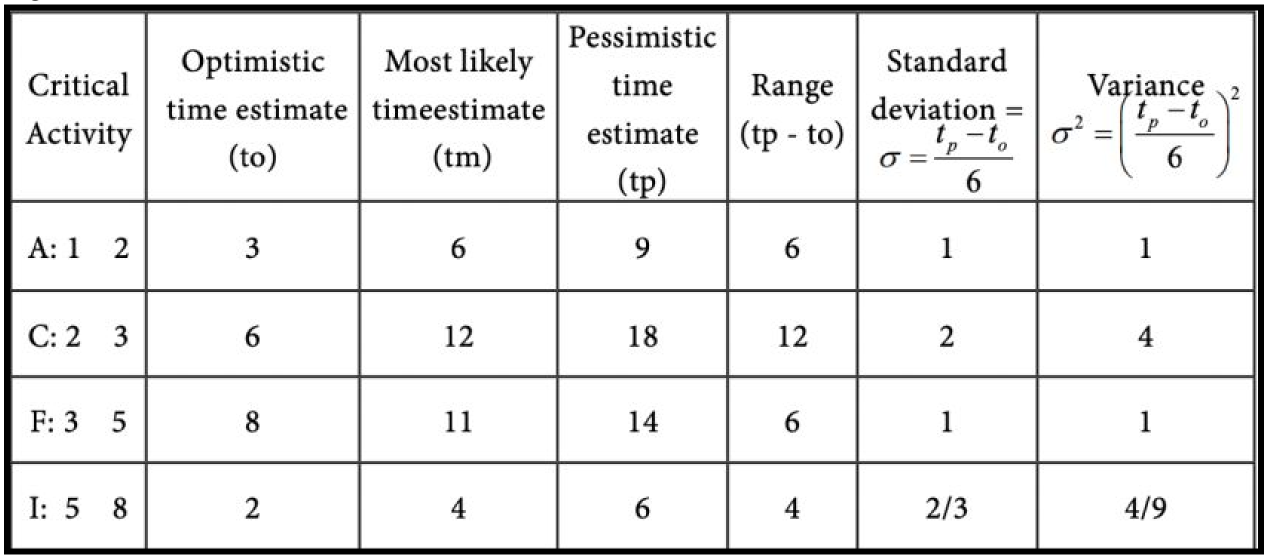
Two experts A and B examined an activity and arrived at the following time estimates.

Expert	Time Estimate		
	t_o	t_m	t_p
A	4	6	8
B	4	7	10

Determine which expert is more certain about his estimates of time:

Soln:
Expert A is more certain about his estimates because his standard deviation is smaller.

- E.g.



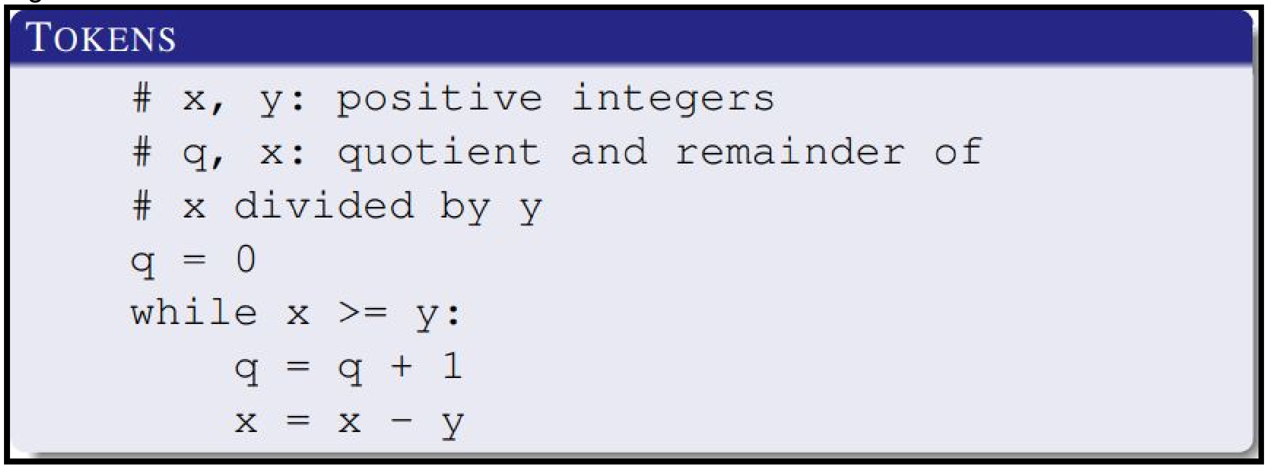
Critical Activity	Optimistic time estimate (to)	Most likely timeestimate (tm)	Pessimistic time estimate (tp)	Range (tp - to)	Standard deviation = $\sigma = \frac{t_p - t_o}{6}$	Variance $\sigma^2 = \left(\frac{t_p - t_o}{6}\right)^2$
A: 1 2	3	6	9	6	1	1
C: 2 3	6	12	18	12	2	4
F: 3 5	8	11	14	6	1	1
I: 5 8	2	4	6	4	2/3	4/9

Variance of project time/Variance of project length = Sum of the variances for the critical activities = 1+4+1+ 4/9 = 58/9 Weeks.
Standard deviation of project time = √Variance = √(58/9) = 2.54 weeks.

<u>Measuring Cohesion:</u>

- **Cohesion** refers to what the class or module can do. Low cohesion would mean that the class does a great variety of actions - it is broad, unfocused on what it should do. High cohesion means that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class.
- A cohesive function is a function that makes use of all of its parameters in most of its lines. You get the highest amount of cohesion if you have all the parameters mixed together in 1 line.

- A cohesive function will be small.
  I.e. The bigger the cohesion, the shorter the code.
- To measure cohesion, we will define some terms first:
  - **Output Slice:** Statements that affect the value of the output (initialization does not count).
  - **Input Slice:** Statements that are affected by the value of the input.
    Either output slice or input slice can be used to define cohesion. We will use output slice.
  - **Token:** A variable or a constant.
  - **Glue token:** The tokens present in more than one slice.
  - **Superglue token:** The tokens present in all slices.
- Here are some functional cohesion measurements:
  1. **Weak functional cohesion:** The ratio of glue tokens versus total tokens.
  2. **Strong functional cohesion:** The ratio of superglue tokens versus total tokens.
  3. **Adhesiveness of a token:** Percentage of output slices containing that token.
  4. **Code adhesiveness:** The average adhesiveness of all tokens.
- E.g.



TOKENS

```
# x, y: positive integers
# q, x: quotient and remainder of
# x divided by y
q = 0
while x >= y:
    q = q + 1
    x = x - y
```

In the code above:
  - There are 2 output slices. (q = q + 1 and x = x - y)
  - The tokens are 0, 1, q, x, y.
  - The glue tokens are x, y.
  - There are no superglue tokens.
  - The adhesiveness of tokens are:
    1. A(x) = 1/2 because it occurs in 1 output slice.
    2. A(y) = 1/2 because it occurs in 1 output slice.
    3. A(q) = 1/2 because it occurs in 1 output slice.
    4. A(1) = 1/2 because it occurs in 1 output slice.
  - The code adhesiveness is 4*(50%)/4 = 50% = 1/2. This is because the function does 2 things, computes the quotient and computes the remainder.
    The closer to 100% for the code adhesiveness the better.

Measuring coupling:

- **Coupling** may be defined as the degree of interdependence that exists between software modules and how closely they are connected to each other.
- There are several dimensions of coupling:
    1. **Content coupling:** This is a type of coupling in which a particular module can access or modify the content of any other module.
    I.e. Content coupling occurs when one module uses the code of another module.
    2. **Common coupling:** This is a type of coupling in which you have multiple modules having access to a shared global data.
    I.e. Common coupling occurs when several modules have access to the same global data.
    3. **Control coupling:** This is a type of coupling in which one module can change the flow of execution of another module.
    I.e. Control coupling is one module controlling the flow of another, by passing it information on what to do.
    4. **Data coupling:** In this type of coupling, two modules interact by exchanging or passing data as a parameter.
    I.e. Data coupling occurs when modules share data through, for example, parameters.
- We want high cohesion and low coupling.
- For data and control flow coupling, let:
    - $\mathbf{d_i}$: number of input data parameters
    - $\mathbf{c_i}$: number of input control parameters
    - $\mathbf{d_o}$: number of output data parameters
    - $\mathbf{c_o}$: number of output control parameters
- For global coupling, let:
    - $\mathbf{g_d}$: number of global variables used as data
    - $\mathbf{g_c}$: number of global variables used as control
- For environmental coupling, let:
    - **w**: number of modules called (fan-out)
    - **r**: number of modules calling the module under consideration (fan-in)
- Then,

$$\boxed{\text{Coupling(C)} = 1 - \frac{1}{di + 2ci + do + 2co + gd + 2gc + w + r}}$$