The Orders of Ignorance:

- Software is not a product, but rather it's a medium for storing knowledge.
- Therefore, software development is not a product producing activity–it is a knowledge acquiring activity.
- Since knowledge is just the other side of the coin of ignorance, software development is an ignorance-reduction activity.
- Armour's Order of Ignorance:
    0. **Zeroth Order Ignorance (0OI) (Lack of ignorance):**
    - I have 0OI when I provably know something.
    - I have Zeroth Order Ignorance (0OI) when I know something and can demonstrate my lack of ignorance in some tangible form.
    - 0OI is provable and proven knowledge that is deemed "correct" by some qualified agency. In software this means that the knowledge is invariably factored into usable form. In all forms of knowledge there must be some external "proof" element that qualifies the knowledge as being correct.
    - E.g.
      Prove a theorem
      Building a system that satisfies the user
      Ability to use Excel
    1. **First Order Ignorance (1OI) (Lack of knowledge):**
    - I have 1OI when I do not know something.
    - I have First Order Ignorance (1OI) when I do not know something and I can readily identify that fact. 1OI is basic ignorance or lack of knowledge.
    - E.g.
      I do not know how to speak the Russian language.
    2. **Second Order Ignorance (2OI) (Lack of awareness):**
    - I have 2OI when I do not know that I do not know something.
    - I have Second Order Ignorance (2OI) when I do not know that I do not know something. That is to say, not only am I ignorant of something (I have 1OI), I am unaware of what it is I am ignorant about. I do not know enough to know what it is that I do not know.
    3. **Third Order Ignorance (3OI) (Lack of Process):**
    - I have 3OI when I do not know of a suitably efficient way to find out that I do not know that I do not know something.
    - I have Third Order Ignorance (3OI) when I do not know of a suitably efficient way to find out that I do not know that I do not know something, which is lack of a suitable knowledge gathering process.
    - This presents me with a major problem: If I have 3OI, I do not know of a way to find out that there are things that I do not know that I do not know. Therefore, I cannot change those things that I do not know into either things that I know, or at least things that I know that I do not know, as a step toward converting the things that I know that I do not know into things that I know
    4. **Fourth Order Ignorance (4OI) (Meta ignorance):**
    - I have 4OI when I do not know about the Five Orders of Ignorance.
    - Knowledge is highly and intrinsically recursive. To know about anything, you must first know about other things which define what you know.
- You can ask questions to reduce the level of ignorance.
- Asking questions will:
    a. Reveal ignorance and intelligence
    b. Reduce ignorance and assumptions.

Minimum Viable Product (MVP):

- A **minimum viable product (MVP)** is a product with enough features to attract early-adopter customers and validate a product idea early in the product development cycle. In industries such as software, the MVP can help the product team receive user feedback as quickly as possible to iterate and improve the product.
- The MVP is a product with the highest return on investment versus risk.
- We need to understand the users who will interact with it.
  I.e.
  What are the different types of users?
  What are their needs, preferences, and constraints?
  Which users are we targeting first?

Personas:

- Personas are fictional characters, which you create based upon your research in order to represent the different user types that might use your service, product, site, or brand in a similar way. Creating personas helps the designer to understand users' needs, experiences, behaviors and goals.
- A tool/technique to help us better understand our users.
- A short description or biography of fictitious, archetypal customers.
- Originally, a concept from the marketing & advertising world.
- The personas should be detailed and should include the below information:
  - Name, Photo, Age, Gender.
  - Personality
  - Skills
  - Environment (where do they live, what do they usually do, at what time, and where)
  - What drives/motivates them.

User Stories:

- User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:
  As a <type of user>, I want <some goal> so that <some reason>.
- E.g.
  1. As a student, I would like to see a summary of results for past quizzes I completed so that I can track my progress.
  2. As Cathy (a teacher), I would like to be able to quickly view the quiz grades on my phone of my students as well as the number of questions in which they have answered, so that I may be able to better focus my attention appropriately.
- User stories must be small enough to fit into a sprint.
- During the early stages of a project, some user stories may be left as large features that encompass many stories.
- Too many stories make it difficult for the product owner to manage.
- As the feature gets closer to a point where it will actually be worked on, it can be broken down into smaller stories.
- E.g.
  As a vice president of marketing, I want to select a holiday season to be used when reviewing the performance of past advertising campaigns so that I can identify profitable ones.

  Details are somewhat vague at this point. What qualifies as a holiday? Does the time span of the holiday matter?
  One approach to fill in the details is to expand this user story into several smaller stories.

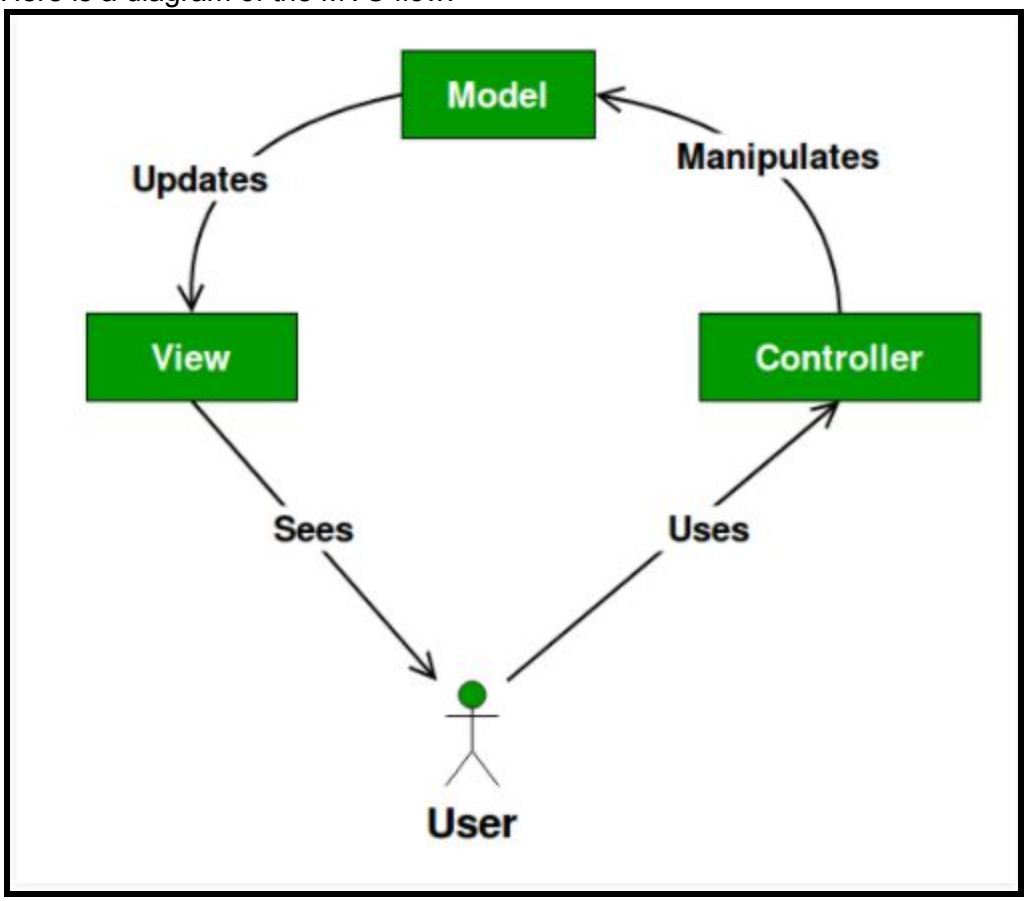
Another approach is to offer more details in the **criteria of satisfaction (COS)**.

- CoS helps the team understand the ultimate goal of each story.
- CoS must be testable.
- CoS is additional objectives and high-level tests about each feature.
- E.g.
  For a user story of "As a user, I want to be able to cancel a reservation" this is a possible CoS:
  - A user who cancels more than 24 hours in advance gets a complete refund.
  - A user who cancels less than 24 hours in advance is refunded all but a $25 cancellation fee.
  - A cancellation code is displayed on the site and is emailed to the user.
- We can use 3x5 index cards to represent user stories.
  The card size constrains the amount of detail in a story.
- The backside of the card is ideal for listing the cos.
- Story details/CoS should always be limited to not present all design details. The card size ensures this.
- A good user story follows the **INVEST** rule:
  - **Independent:**
  - Stories should be independent from one another.
  - Dependencies create problems and make it harder to prioritize and estimate.
  - **Negotiable:**
  - Stories are not contracts or detailed requirements.
  - They act as placeholders for conversations between the team and stakeholders.
  - Stories that are too detailed create an illusion that all details are known and that further discussion isn't required.
  - **Valuable:**
  - Need to communicate the values of a feature, not only to the user, but also to the team and other stakeholders.
  - The product owner determines the priority of the stories based on their value.
  - If they are not expressed in terms of their value it is difficult to determine how important they are.
  - **Estimable:**
  - We need to be able to estimate stories. This requires us to know what we are building and how.
  - If the story's scope is too large, or we do not have sufficient knowledge about the problem we will have difficulty making estimates.
  - **Sized appropriately:**
  - Stories need to fit a sprint.
  - Larger stories will be disaggregated.
  - A group of small stories can be grouped together in a larger story.
    It may be more manageable.
  - Example: minor bug fixes, polishing tasks do not need a separate story for each one.
  - **Testable:**
  - A story should be verified before the end of the sprint.
  - Allows us to determine if we have satisfied the stakeholders.
  - The cos on the back of the index cards will help create the tests.

Model-View-Control Pattern (MVC Pattern):

- This pattern is used to separate an application's concerns.
- The MVC design pattern specifies that an application consist of a model, a view, and a control. The pattern requires that each of these be separated into different objects.

- **Model:**
  - Represents an object carrying data. It can also have logic to update the controller if its data changes.
  - Specific to your domain (the application object).
  - Usually kept very simple. They just encapsulate a number of attributes/properties in a single object.
- **View:**
  - Represents the visualization of the data that model contains.
    I.e. The screen representation of the data.
  - What the user sees (e.g. HTML page, the GUI of your mobile app, a text-based console).
  - **Note:** The user is not necessarily a human. For example, programs want to view the data using some text format.
- **Controller:**
  - Acts on both model and view. It controls the data flow into the model object and updates the view whenever data changes. It keeps view and model separate.
  - Controls the flow of the application.
  - Connects between the model and the view.
  - Defines the way the user interface reacts to the user input.
- Here is a diagram of the MVC flow:



NoSQL:

- The concept of NoSQL databases became popular with internet giants like Google, Facebook, Amazon, etc who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could **scale up** our systems by upgrading our existing hardware. This process is expensive.
The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as **scaling out**.

- The growth of the web raised the need for larger, more scalable storage solutions.
- A variety of key-value storage solutions were designed for better availability, simple querying, and horizontal scaling.
- This new kind of data store became more and more robust, offering many of the features of the relational databases.
- Different storage design patterns emerged, including key-value storage, column storage, object storage, and the most popular one, document storage.
- In a common relational database, data is stored in different tables, often connected using a primary to foreign key relation.
  A program will later reconstruct the model using various SQL statements to arrange the data in some kind of hierarchical object representation.
- Document-oriented databases handle data differently.
  Instead of using tables, they store hierarchical documents in standard formats, such as JSON and XML.
- Advantages of NoSQL DBs:
  - **High scalability:** NoSQL databases use sharding for horizontal scaling. Sharding is partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved. Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement but horizontal scaling is easy to implement. NoSQL can handle huge amounts of data because of scalability, as the data grows NoSQL scales itself to handle that data in an efficient manner.
  - **High availability:** Auto replication feature in NoSQL databases makes it highly available because in the case of any failure data replicates itself to the previous consistent state.
- Disadvantages of NoSQL DBs:
  - **Narrow focus:** NoSQL databases have very narrow focus as it is mainly designed for storage but it provides very little functionality. Relational databases are a better choice in the field of Transaction Management than NoSQL.
  - **Open-source:** NoSQL is an open-source database. There is no reliable standard for NoSQL yet. In other words two database systems are likely to be unequal.
  - **Management challenge:** The purpose of big data tools is to make management of a large amount of data as simple as possible. But it is not so easy. Data management in NoSQL is much more complex than a relational database. NoSQL, in particular, has a reputation for being challenging to install and even more hectic to manage on a daily basis.
  - **GUI is not available:** GUI mode tools to access the database are not flexibly available in the market.
  - **Backup:** Backup is a great weak point for some NoSQL databases.

Neo4j:

- Neo4j is a graph database.
- It uses graphs to store and process the data.
- Data is organized into nodes and relationships.
- Properties are stored in either nodes or relationships.
- Neo4j's native data manipulation language is Cypher.

- A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. It is composed of two elements
    1. Nodes (vertices)
    2. Relationships (edges).
- A graph database is a database used to model the data in the form of graphs. In here, the nodes of a graph depict the entities while the relationships depict the association of these nodes.
- ACID Properties:
    - ATOMIC: The whole transaction or nothing.
    - CONSISTENT: Upon completion of a transaction, the db is structurally sound.
    - ISOLATION: Transactions appear to apply in isolation from one another.
    - DURABLE: Once a transaction is complete, it persists, even in case of various failures.
- Cypher is:
    - Declarative, readable, expressive.
    - Made for CRUD on graphs.
    - Based on patterns.
    - Interacts safely with the remote database using a binary protocol called Bolt.
- E.g.

A PROPERTY GRAPH HAS

- Nodes `(:PERSON)`
    - have properties (`{name: "Donald"}`)
- Relationships [:`WORKS_WITH`]
    - also have properties (`{company: "Bluecat"}`)

AN EXAMPLE OF CREATE

```
CREATE
    (: PERSON {name:"Donald"})
        -[:WORKS_WITH {company: "Bluecat"}]->
    (: PERSON {name: "Jasvir"})
```

WHO WORKS WITH JASVIR AT BLUECAT?

```
MATCH
  (p1: PERSON)
        -[:WORKS_WITH {company:"Bluecat"}]->
  (:PERSON {name:"Jasvir"})
RETURN
  p1
```