What is Software and how is it built?

Gathering requirements for Software projects is about asking questions, potentially thousands of questions, about what the user requires. I distinctly remember the first time I was given the task of gathering software requirements. I started my career in IT as a programmer, which meant a Systems Analyst fed me specifications, I created code based on those specifications and passed it back to him. I had no direct contact with the user. I then received a promotion to a Programmer Analyst. That meant I could talk to people.

One of my first tasks upon receiving that promotion was to gather the requirements for a new Order Tracking system. It was assumed at that time that gathering requirements was an unskilled job. "***How difficult can it be to find out what someone wants? All you have to do is ask***". I scheduled an interview with the user and asked one questions: "***What do you want?***". They gave me lots of information which I dutifully wrote down and took back to my office. I wrote some code based on the information I received and presented it to the user. I got lots of "Where did you come up with this?" "That's not what I asked for!" "That's not what I want!". So where did I go wrong.

Software is a product to be created like any other product. If you are building software for yourself, sure, build whatever you want. However, if you are building it for someone else, you need to find out what that person or persons want. That is what we mean by ***Gathering Requirements***.

Let's say we want a house built for us. We are going to hire someone to design it based on our requirements and then build it. So we go and talk to an architect. That architect will not say to us "What do you want?". He will ask many questions based on the context of houses. "How many bedrooms do you want?" "How many floors?" "How many bathrooms?" "Do you want a fireplace?" "Gas or wood burning?" "Do you do a lot of entertaining?" "Yes, so you'll want a large dining room?" "Two or 3 car garage?" "Attached or detached" etc. etc. etc. That is because he knows what a house is, in the minutest of detail, and how one is built. He has "***Domain Knowledge***".

In order for us to know what questions to ask our clients, we must also possess this Domain Knowledge about Software.

We can understand a house, or plane, or ship etc. because they are physical. We can ask questions based on our inherent understanding of 3 dimensions and our senses. If I want someone to build a wall for me I can describe my requirements in terms of length, width, height, location, colour, materials weight, etc. All things we are familiar with. Software is different. It has no length, weight, height, colour etc. Except for the interfaces, it is, for all intents and purposes, invisible. For decades we had this problem. Since the product we need to build is invisible, what questions do we ask to find out our client's needs? If we had a very clear picture of what Software was and how it was built, we could then ask specific questions about it in context, just like our architect.

I remember distinctly the first time I was introduced to the idea of what software really was. A way of thinking about it, describing it, that was clear and precise. It was a revelation, an epiphany... *the lightbulb went on.*

When you think of a house, or a bridge, or a wall... you think of its characteristics. Software also consists of characteristics, but it consists of something else, distinctly different. It has ***behaviour***. It is not static! It is dynamic! It performs! It ***does stuff***!

So that means when we describe software we need to describe what it is, meaning its characteristics, but also what it can do, meaning its functionality.

Software does one, and only one, thing. It responds! It does not initiate anything by itself. It responds to events. Our most fundamental description of software can then become "What events can occur that our software should respond to?". If we can come up with that, then we have come a long ways towards understanding software.

The fundamental events

There are 3 basic types of events in life; the decision, the circumstance and the temporal.

People decide things every day. Customers decide to buy products. Managers decide to hire, promote or fire employees. Employees decide to quit. Managers decide to change the prices of products, etc. The list is possibly endless. If the response to that event involves software then the event, as well as the response, is also in scope.

Things also happen that are not decisions, they just happen. The phone rings. A box arrives on our desk. We need to respond to that event as well. We need to "answer the phone" or "receive inventory".

Things also happen periodically. The 15th and last day of the month we pay people. The 20th we pay approved vendor invoices. The 10th we pay salesman commissions. Three O'clock every Wednesday afternoon we get the mail. Lots of things we do based on time.

So, at the most fundamental level, if we want to describe a piece of software, it would answer the question "What events does it respond to?".

That of course is not sufficient to allow us to create the software. We must also answer the question "How must it respond to those events?"
Now we're getting somewhere. If we could find out from our business what events can occur that the software must respond to and how must it respond, then we could relay that information to our developers and they could build the software according to the user requirements.