Divide & Conquer is Recursive Design

Recursion is the process of "looping" where the problem becomes simpler every time through the "loop" and where there is a terminating case (something that stops the loop).

Designing an application using divide & conquer is recursion. The design is successively broken down into (1) a simple problem to be solved and, (2) the rest of the problem.

The terminating case for divide & conquer is the decision that (2) is "so simple" that it can be solved without further division.

When we (my consulting company) built credit terminal applications, we used diagrams and a terminating case.

The diagrams gave us a way to visualize divisions in divide and conquer. Each sub-problem – (1) – became a box (literally, on a whiteboard, then on a diagram) containing a simple problem that could be solved.

Our terminating case was: whether everyone in the room (e.g. 5-10 developers) agreed that all sub-problems could be coded up and solved in about 1/2 day. That included alpha-testing, etc.

In my opinion, a grain-size of 3 weeks for a "sprint" in Agile methodology is too large. The grain-size should be reduced to about 1/2 day (that is 3 weeks = 21 days * 2 = 42 chunks * number of developers). Obviously, the customer cannot be consulted for every completed chunk (every grain, every terminating condition, every box). This is where Agile does not use *enough* divide and conquer and recursion. A "sprint" presented to a customer, should consist of many smaller chunks (grains, boxes). Intermediate chunks can be "reviewed" by the developer, every group of chunks can be reviewed by the team about once every few days, every group of group-of-chunks can be reviewed by technical management every week (or so) and every group of groups-of-groups-of-chunks can be reviewed by the product owner every couple of weeks.

Programmers and technical leads and CTO's already do something like this, implicitly. They draw out their "design" on a whiteboard in a meeting. No one sits in a meeting longer than one hour. The whiteboard represents the chunks – the grains – developers and tech leads in the meeting "approve" of the chunks as being implementable in a finite amount of time, etc.

One hour is usually not enough to subdivide a problem into enough pieces. The "terminating case" is fuzzy and further subdivision is left up to the programmers. In other words, the programmers do not just write code, but they, also finish designing the solution. Some are better at this than others.[1]

---

1 The people who are good at finishing the design should be designated as Architects and should not waste their time coding. The others should be designated coders and should not waste everyones' time doing design work. There are many more levels of Architects/Engineers/Implementors - see my essay "Software Development Roles".