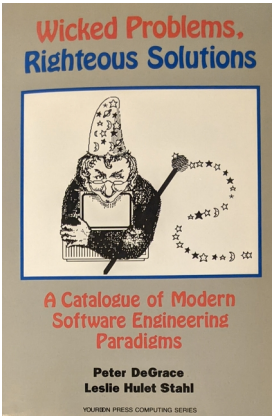

Origen del modo de desarrollo ágil Scrum.

1990
Peter DeGrace & Leslie Hulet Stahl
“Wicked Problems, Righteous Solutions”

El término scrum sólo se empleaba para definir la formación de los juigadores de rugby al disputar la pelota. En 1986[1] Hirotaka Takeuchi e Ikujiro Nonaka lo emplearon metafóricamente en la expresión ***"Moving the Scrum Downfield"*** *para referirse al modo de desarrollo que mostraba 6 características:*

1.- Inestabilidad inherente.
2.- Equipos autoorganizados.
3.- Solapamiento de las fases de desarrollo.
4.- Multiaprendizaje.
5.- Control sutil.
6.- Transferencia organizativa del conocimiento.

Muchos pensábamos, que los creadores de una propuesta de desarrollo iterativo, tomando el nombre de scrum con este sentido metafórico fueron Jeff Sutherland y Ken Scwaber en 1995[2].
Parece que no. Que no fueron ellos, que el origen de un modo de desarrollo iterativo con este nombre y sentido se debe a Peter DeGrace y Leslie Hulet Stahl, en 1990, en su libro "Wicked Problems, Righteous Solutions"[3].

(1) Hirotaka Takeuchi & Ikujiro Nonaka, 1986 “The New New Product Development Game”, Harvard Business Review.
(2) Scrum Development Process, 1995, Ken Schwaber OOPSLA Business Object Design and Implementation Workshop
(3) Peter DeGrace & Leslie Hulet Stahl, 1990 “Wicked Problems, Righteous Solutions”, 1990, Prentice Hall.

CAPÍTULO SIETE El modelo "todo a la vez”

La visión tradicional del desarrollo de software es asociar una fase con actividades del mismo nombre. Así, la fase de requisitos se asocia con las actividades de requisitos. Esta visión tradicional también sostiene lo contrario: que las actividades de requisitos están asociadas a la fase de requisitos y que las actividades están ligadas a la fase.

Sin embargo, esto no es del todo cierto. Como observó Zelkowitz, todas las fases incluyen actividades que normalmente se consideran realizadas en otras fases. [Zelkowitz 1988, pp.331, 336].

En la creación de prototipos, esto empieza a desvelarse un poco más. Si se realizan prototipos para recopilar requisitos, el diseño, la codificación y las pruebas forman parte de esa fase en gran medida. Esto era así aunque estas actividades se repiten cuando se iniciaban las fases "reales" asociadas a ellas.

En este capítulo, nos encontramos con enfoques de desarrollo de software en los que las actividades están vinculadas a otras actividades, y no a fases. Se denominan "todo a la vez" porque todas las "fases" parecen realizarse simultáneamente. En realidad, son una clase de enfoques monofásicos. (Véase la Figura 7-1.)

El modelo "todo a la vez" es similar al modelo de creación de prototipos en el sentido de que, durante un cierto periodo de tiempo, muchas actividades relacionadas con las fases tradicionales se desarrollan simultáneamente. Pero se diferencia en que un observador externo no vería las actividades como separadas. Para este observador externo, parece que todas las actividades se realizan al mismo tiempo. Al principio, esto parece el pirateo "indisciplinado" del que Boehm se queja con razón. Pero no es eso en absoluto. **Es la forma en que muchas personas desarrollan software cuando no sólo no conocen la solución, sino que tampoco conocen el problema, cuando intentan resolver problemas complejos**.

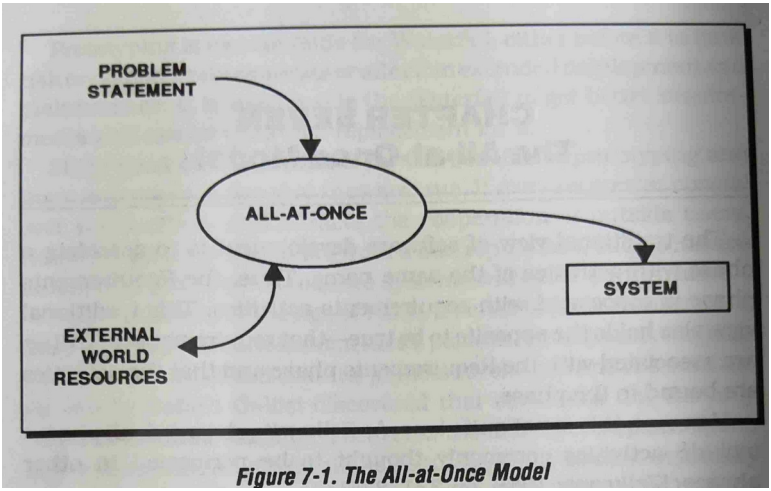

Figure 7-1. The All-at-Once Model

El modelo "todo a la vez" se presenta de varias formas. Hay enfoques de equipo, bipersonales y unipersonales. Todos tienen muchas de las mismas características generales, pero son únicos en aspectos importantes y tienen historias diferentes.

Enfoques de equipo-Sashimi y Scrum

Comencemos con los enfoques de equipo, Sashimi y Scrum. El enfoque Sashimi tiene su origen en los japoneses y sus experiencias con el modelo Waterfall. [Takeuchi y Nonaka 1986]
Los japoneses adoptaron el modelo Waterfall muy pronto y encontraron en él muchos de los problemas que otros han tenido. Además, vieron que **la velocidad y la flexibilidad eran tan importantes en el desarrollo de productos como la alta calidad tradicional, el bajo coste y la diferenciación del producto**. Así que, como han hecho con tantas otras cosas, lo adaptaron a su propio estilo. Redujeron el número de pasos a cuatro, pero no eliminaron ninguna actividad. Esto provocó, en efecto, un solapamiento entre las fases tradicionales. Además, hicieron que las

propias cuatro fases se solaparan. Lo llamaron Sashimi, por la forma japonesa de presentar el plato de pescado crudo en rodajas, en la que cada rodaja se apoya parcialmente sobre la anterior. (Véase la figura 7-2.)

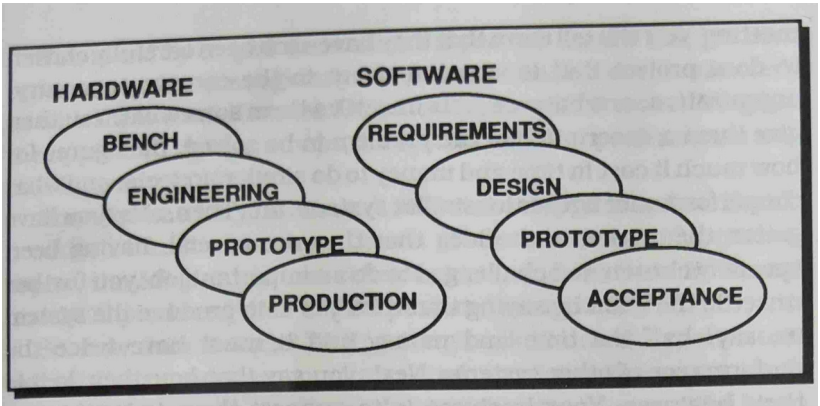

Figure 7-2. The Sashimi Approach

El modelo etiquetado como hardware representa el calendario de desarrollo de productos de Fuji-Xerox. El modelo etiquetado como software es mi interpretación de cómo se renombrarían las cuatro fases de hardware para un proyecto de software.
Otras empresas fueron un paso más allá. Redujeron el número de fases de cuatro a una y le dieron un nombre de rugby. En el rugby, todos los miembros del equipo actúan conjuntamente con los demás para "mover el balón por el campo". **Este pack de equipo se denomina Scrum**. (Ver Figura 7-3.)

Como se puede ver, el enfoque Sashimi es como el modelo Waterfall, salvo que gran parte del final de una fase coincide con la fase anterior. A primera vista, esto no es tan nuevo. Mucha gente ha sugerido un solapamiento como parte del modelo Waterfall. Pero, por lo general, se referían al procedimiento de validación que, si

fallaba, hacía que la entrada se devolviera a la fase anterior. Los practicantes de Sashimi pretenden que haya una estrecha comunicación e integración entre las fases solapadas, y que haya una estrecha cooperación entre las fases contiguas.
Si Scrum se aplicara al desarrollo de software, sería algo así: Supongamos que tiene que realizar un proyecto de desarrollo de software. Para cada fase tradicional, puede recurrir a un grupo de personas con experiencia. En lugar de tener varios diseñadores para la fase de diseño y varios programadores para la fase de construcción, etc., se forma un equipo seleccionando cuidadosamente a una persona de cada grupo.
Durante una reunión de equipo, les dirá que cada uno de ellos ha sido cuidadosamente elegido para realizar un proyecto muy importante para la empresa, el país, la organización o lo que sea. Esto les inquieta un poco. A continuación, les describe el problema que hay que resolver, les da las cifras de lo que ha costado en tiempo y dinero realizar proyectos similares y las cifras de rendimiento de sistemas similares. Después, una vez que se hayan hecho a la idea de que son especiales, de que han sido especialmente elegidos y de que se les ha retado a hacer un trabajo importante, se desestabiliza aún más al equipo diciéndoles que su trabajo consiste en producir el sistema en, digamos, la mitad de tiempo y dinero y que debe tener el doble de rendimiento que otros sistemas. A continuación, les dices que cómo lo hagan es asunto suyo. Tu trabajo consiste en ayudarles a conseguir nuevas fuentes. Después, les dejas en paz.

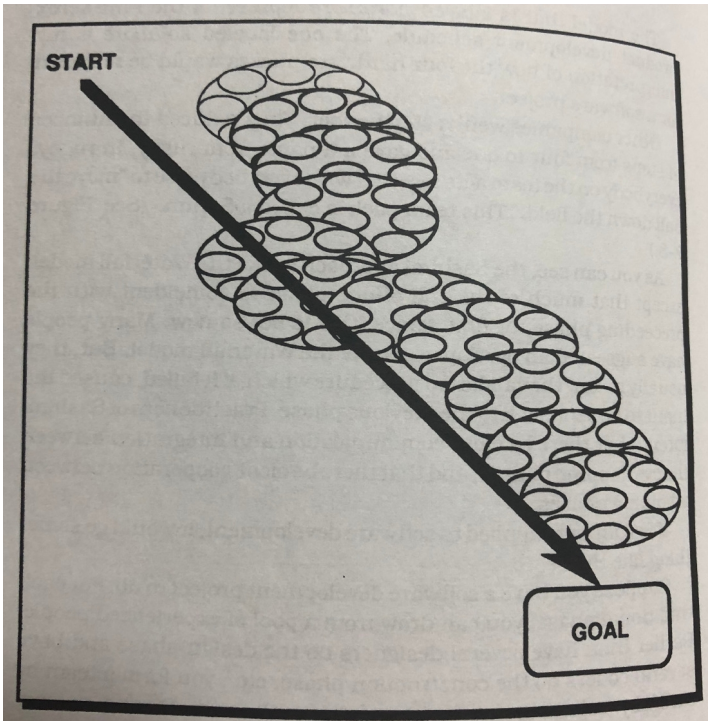

Figure 7-3. The Scrum Approach

Texto original

CHAPTER SEVEN
The All-at-Once Model

The traditional view of software development is to associate a phase with activities of the same name. Thus, the Requirements phase is associated with requirements activities. This traditional view also holds the opposite to be true—that requirements activities are associated with the Requirements phase and that the activities are bound to the phase.

However, this is not quite true. As Zelkowitz observed, all phases include activities commonly thought to be performed in other phases. [Zelkowitz 1988, pp.331, 336]

In prototyping, this begins to unravel a little more. If you did prototyping to gather requirements, then design, coding and testing were part of that step in a big way. This was true even though these activities were to be repeated when the "real" phases associated with them were begun.

In this chapter, we encounter approaches to software development where activities are bound to other *activities*, and not to phases. They are called "all-at-once" because all the "phases" seem to be done concurrently. Actually, they are a class of one-phase approaches. (See Figure 7-1.)

The All-at-Once model is similar to the Prototyping model in that during some time period many activities related to traditional phases are going on concurrently. But, it is different in that an outside observer would not see the activities as separate. To this outside observer, it looks like all the activities are done at the same time. This, at first, seems like the "undisciplined" hacking Boehm rightly complains about. However, it is not that at all. It is the way many people develop software when they not only do not know the solution, but they also do not know the problem—when they are trying to solve wicked problems.

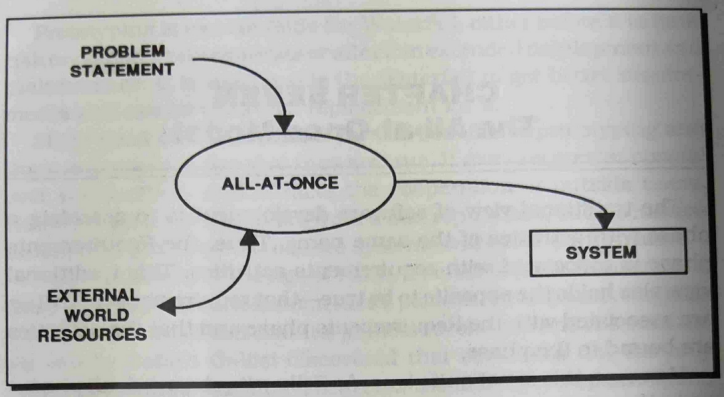

Figure 7-1. The All-at-Once Model

The All-at-Once model occurs in several forms. There are team, two-man, and one-man approaches. They all have many of the same general characteristics, but are they unique in important ways and have different histories.

Team Approaches—Sashimi and Scrum

Let us begin with the team approaches, Sashimi and Scrum. The Sashimi approach originated with the Japanese and their experiences with the Waterfall model. [Takeuchi and Nonaka 1986]

The Japanese adopted the Waterfall model early on and found many of the problems with it that others have. In addition, they saw that speed and flexibility were as important in product development as traditional high quality, low cost, and product differentiation. So, as they have done with so many other things, they adapted it to their own style. They reduced the number of steps to four, but did not remove any activities. This caused, in effect, an overlap among traditional phases. And, they made the four phases themselves overlap. They called it Sashimi, named for the Japanese way of presenting the sliced raw fish dish, where each slice rests partially on the slice before it. (See Figure 7-2.)

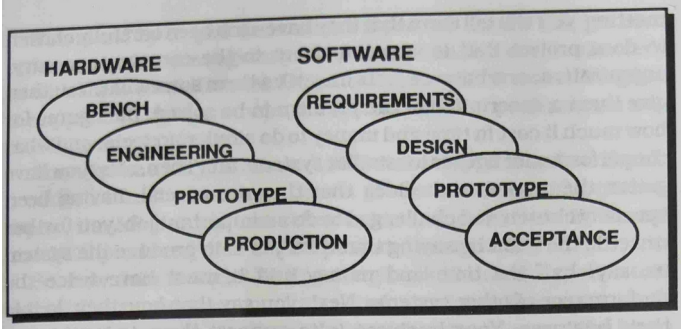

Figure 7-2. The Sashimi Approach

The model that is labeled *hardware* represents the Fuji-Xerox product development schedule. The one labeled *software* is my interpretation of how the four hardware phases would be renamed for a software project.

Other companies went a step further. They reduced the number of steps from four to one and gave it a name from rugby. In rugby, everybody on the team acts together with everybody else to "move the ball down the field." This team pack is called a Scrum. (See Figure 7-3.)

As you can see, the Sashimi approach is like the Waterfall model, except that much of the end of one phase is coincident with the preceeding phase. At first glance, this is not so new. Many people have suggested an overlap as part of the Waterfall model. But, they usually meant the validation procedure which, if it failed, caused the input to be sent back to the previous phase. Practitioners of Sashimi intend that there be close communication and integration between the overlapping phases, and that there be close cooperation between adjoining phases.

If Scrum were applied to software development, it would go something like this:

Suppose you have a software development project to do. For each traditional phase, you can draw from a pool of experienced people. Rather than have several designers do the design phase and have several coders do the construction phase, etc., you form a team by carefully selecting one person from each pool. During a team

meeting, you will tell them that they have each been carefully chosen to do a project that is very important to the company, country, organization, or whatever. This unsettles them somewhat. You then give them a description of the problem to be solved, the figures for how much it cost in time and money to do similar projects, and what the performance figures for similar systems are. Then, after you have gotten them used to the idea that they are special, having been specially chosen and challenged to do an important job, you further unsettle the team by saying that their job is to produce the system in, say, half the time and money and it must have twice the performance of other systems. Next, you say that *how* they do it is their business. Your business is to support them in getting resources. Then, you leave them alone.