Development Cheat Sheet

Agile Methodology

You will often hear Extreme Programming (XP) associated with Agile Development as well as Feature Driven Development. (FDD).

Turns out, Scrum is a framework for agile software development.

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

•Customer satisfaction by rapid, continuous delivery of useful software
•Working software is delivered frequently (weeks rather than months)
•Working software is the principal measure of progress
•Even late changes in requirements are welcomed
•Close, daily cooperation between business people and developers
•Face-to-face conversation is the best form of communication (co-location)
•Projects are built around motivated individuals, who should be trusted
•Continuous attention to technical excellence and good design
•Simplicity
•Self-organizing teams
•Regular adaptation to changing circumstances

Agile methods are sometimes characterized as being at the opposite end of the spectrum from "plan-driven" or "disciplined" methods. This distinction is misleading, as it implies that agile methods are "unplanned" or "undisciplined". A more accurate distinction is that methods exist on a continuum from "adaptive" to "predictive". Agile methods lie on the "adaptive" side of this continuum.

Scrum

Scrum is a “process skeleton” which contains sets of practices and predefined roles. The main roles in Scrum are:

1. the “**ScrumMaster**”, who maintains the processes (typically in lieu of a project manager)

2. the “**Product Owner**”, who represents the stakeholders, represents the business

3. the “**Team**”, a cross-functional group of about 7 people who do the actual analysis, design, implementation, testing, etc.

Sprints are about 4 weeks (with the length being decided by the team).

Chicken and Pig

So the “pigs” are committed to building software regularly and frequently, while everyone else is a “chicken”—interested in the project but really indifferent because if it fails they’re not the pigs—that is, they weren’t the ones that committed to doing it. The needs, desires, ideas and influences of the chicken roles are taken into account, but are not in any way allowed to affect, distort or get in the way of the actual Scrum project.

“Pig” Roles:

•ScrumMaster (or Facilitator)
•Team
•Product Owner

“Chicken” Roles

•Stakeholders (customers, vendors)
•Managers

Extreme Programming (XP)

As a type of agile software development, extreme programming advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.

Other elements of Extreme Programming include: programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure...

Critics have noted several potential drawbacks, including problems with unstable requirements, no documented compromises of user conflicts, and lack of an overall design spec or document.

My Opinion

The coding in paris aspect of this methodology is the lest applicable, as it requires more resources than most people have. I would say on a spectrum of development methodologies, this one heavily favors the development teams to the detriment of project timelines. Maybe if you have the right team of people working together, it’s not a problem. I would think you would need all of the right people that have a great re-pore with each other to make this a practical reality.

Waterfall Methodology (BLITZ?)

The waterfall model is a sequential software development process in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, (validation), Construction, Testing, Maintenance.

I would say this is what BLITZ strives to follows most of the time.

To follow the *waterfall model*, one proceeds from one phase to the next in a sequential manner. For example, one first completes requirements specification, which after sign-off are considered "set in stone." When the requirements are fully completed, one proceeds to design. The software in question is designed and a blueprint is drawn for implementers (coders) to follow — this design should be a plan for implementing the requirements given. When the design is fully completed, an implementation of that design is made by coders. Towards the later stages of this implementation phase, separate software components produced are combined to introduce new functionality and reduced risk through the removal of errors.

Thus the waterfall model maintains that one should move to a phase only when its preceding phase is completed and perfected.

Sound Familiar?

The waterfall model is argued by many to be a bad idea in practice. This is mainly because of their belief that it is impossible for any non-trivial project to get one phase of a software product's lifecycle perfected, before moving on to the next phases and learning from them.

For example, clients may not be aware of exactly what requirements they need before reviewing a working prototype and commenting on it; they may change their requirements constantly. Designers and programmers may have little control over this. If clients change their requirements after the design is finalized, the design must be modified to accommodate the new requirements. This effectively means invalidating a good deal of working hours, which means increased cost