# Waterfall

|  |  |
| --- | --- |
| Phase | Description |
| System Requirements | Define and document system requirements |
| Software Requirements | Define and document software requirements |
| Preliminary Design | Program Designers design, define, and allocate data processing modes and document all of it. |
| Analysis | Analyze the requirements and design for defects. |
| Program Design | Design the program |
| Coding | Code the program |
| Testing | Test the program |
| Operations | Deliver and maintain the program. |

Waterfall consists of 8 phases. Unlike the name suggests, you do not proceed from one step to the next until you are finished. Royce suggests that you do the whole process twice. Also, if a defect in a previous phase is found, revert to the previous phase and correct the problem.

# Rapid Prototype

A Type I prototype is iterated on until the program is ready and shippable. For Type I models, the SDLC consists of requirements, training, project planning, rapid analysis, DB design, design of prototype, coding, testing, implementation and maintenance. If during testing the prototype is found wanting, you start again at designing the prototype.

A Type II prototype is a throw-away prototype that is used as a model for the final production system, but is discarded at the project delivery. Type II porotypes are used in a traditional SDLC.

# Spiral Model

The spiral model is an iterative, risk-based development model that is quite flexible when it comes to the actual phases of development. Risks are identified and analyzed, and then a development methodology is chosen that best manages the risks. Most often, the spiral model uses prototypes to aid the process. The process spirals as many times as is necessary to accomplish the goal of the project.

|  |  |
| --- | --- |
| Strategy | Description |
| Planning game: | determines the scope and timing of the next release. |
| Small releases: | puts a simple system into production quickly. |
| Metaphor: | guides all development via a simple story of the overall system. |
| Simple design | the system should be designed as simply as possible at any moment. |
| Testing: | is constantly undertaken for development to continue. |
| Refactoring: | restructures the system without changing its behavior. |
| Pair programming: | all production code is written by two programmers at one machine. |
| Collective ownership: | anyone can change any code anywhere at any time. |
| Continuous integration: | integrate and build the system many times a day. |
| 40-hour work week: | work no more than 40 hours a week as a rule. |
| On-site customer: | must be available full-time |
| Coding standards: | emphasizes communication throughout the code |

# Agile – XP

Extreme programming takes the principles of traditional SDLCs to their extremes. They espouse the strategies to the right and take a unique approach to requirements with user stories replacing requirements.

# Agile – SCRUM

|  |  |
| --- | --- |
| Manifesto | Process |
| |  |  | | --- | --- | | Individuals/Interactions | Processes/Tools | | Working software | comprehensive documentation | | Customer collaboration | contract negotiation | | Responding to change | following a plan |   Prefer items on the left, but don’t neglect  the items on the right! | |  |  | | --- | --- | | Sprint Planning Meeting | Plan next 2-4 week iteration | | Daily Scrum Meeting | Everyone reports yesterday’s progress and today’s plan | | Sprint Execution | Design, Code, Test | | Sprint Review Meeting | Live product demonstration | | Sprint Retrospective | Reflect on the process | | Backlog Refinement | Prep backlog for next sprint | |

Roles: Product Owner, Scrum Development Team, Scrum Master.

# Formal Methods

A formal method in software development is a method that provides a formal language for describing a software artifact (e.g. specifications, designs, source code) such that formal proofs are possible, in principle, about properties of the artifact so expressed. Typically, the proof is that an implementation fulfills its specification.

# Cleanroom

Cleanroom software engineering is a practical process to place software development under statistical quality control. Its first priority is defect prevention rather than defect removal. Its next priority is to provide statistical certification of the software quality through representative-user testing at the system level. It calls for the development of software in increments that permit realistic measurements of statistical quality during development, with provision for improving the measured quality by adding testing, by process changes (such as increased inspections and configuration control), or by both methods. Mathematical verification replaces program debugging before release to statistical testing.

# CMM

CMM measures and evolves the maturity of software development process for an organization.

|  |  |  |
| --- | --- | --- |
| Levels | Focus | Key Process Areas |
| 1 Initial | Ad-hoc/Chaos | none |
| 2 Repeatable | Project Management | Software Project Planning and Oversight, Software Subcontract Management,  Software Quality Assurance, Software Configuration Management,  Requirements Management |
| 3 Defined | Engineering Process | Organization Process Focus, Organization Process Definition, Peer Reviews,  Training Program, Intergroup Coordination, Software Product Engineering  Integrated Software Management |
| 4 Managed | Product and Process Quality | Software Quality Management,  Quantitative Process Management |
| 5 Optimizing | Continuous Improvement | Process Change Management,  Technology Change Management Defect Prevention |