Software Process Modeling

**Function:** To determine the order of the stages involved in software development and evolution and to establish the transition criteria for progressing from one stage to the next.

**Importance:** Providing guidance on the order (phase, increments, prototypes, validation tasks, etc) in which a project should carry out its major tasks.
Evolution of Process Models

1. The code-and-fix model.

2. The stagewise and waterfall models.

3. The rapid Prototyping model.

4. The evolutionary development model.

5. The incremental model.

6. The synchronize-and-stabilize model.

7. The transform model.

8. The spiral model.
The code-and-fix model

1. After a number of fixes, the code became so poorly structured that subsequent fixes were very expensive.

2. Frequently, even well-designed software was such a poor match to users’ needs that it was either rejected outright or expensively redevelopment.

3. Code was expensive to fix because of poor preparation for testing and modification.
The waterfall models

1. The software development process should be subject to discipline.

2. Implementing the product should be postponed until after the objectives of doing so are well understood.
The waterfall models

1. It is difficult to estimate resources accurately.

2. The verification of the requirements specification document performed by the customer is not effective.

3. The assumption that all requirements should be frozen before any development starts is unrealistic.

4. The need for anticipating changes was not stressed.

5. It enforces software engineer to pay more attention to the syntax imposed by the standard than to its semantics.
The waterfall models

- Adding extensions to cover incremental development, parallel developments, program families.

- Accommodation of evolutionary changes.

- Adding formal software development and verification, and stagewise validation and risk analysis.
The evolutionary development model

- Its stages consist of expanding increments of an operational experience.

- Assumption: The user’s operational system will be flexible enough to accommodate unplanned evolution paths.
The evolutionary development model

The assumption that is unjustified in three primary circumstances:

1. Several independently evolved applications must be closely integrated.

2. Temporary work-arounds for software deficiencies increasingly solidify into unchangeable constraints on evolution.

3. The new software is incrementally replacing a large existing system. If the existing system is poorly modularized, it is difficult to provide a good sequence of "bridges" between the old software and the expanding increments of new software.
The transform model

- A formal specification of the best initial understanding of the desired product.

- Automatic transformation of the specification into code.

- An iterative loop, if necessary, to improve the performance of the resulting code by giving optimization guidance to the transformation system.

- Exercise of the resulting product.

- An outer iterative loop to adjust the specification based on the resulting operational experience, and to rederive, reoptimize, and exercise the adjusted software product.
The spiral model

1. Its range of options accommodates the good features of existing software process models, while its risk-driven approach avoids many of their difficulties.

2. It focuses early attention on options involving the reuse of existing software.

3. It accommodates preparation for life-cycle evolution, growth, and changes of the software product.

4. It provides a mechanism for incorporating software quality objectives into software product development.

5. It focuses on eliminating errors and unattractive early.

6. It does not involve separate approaches for software development and software enhancement or maintenance.

7. It provides a viable framework for integrated hardware-software system development.
1. Matching to contract software.

2. Relying on risk-assessment expertise.

3. Need for further elaboration of spiral model steps.
Software Risk Management

- For customers and developers, budget overruns and schedule slips are unsatisfactory.

- For users, products with the wrong functionality, user-interface shortfalls, performance shortfalls, or reliability shortfalls are unsatisfactory.

- For maintainers, poor-quality software is unsatisfactory.
Risk Management

1. Risk assessment.

2. Risk control.
Risk assessment

**Risk identification:** produces lists of the project-specific risk items likely to compromise a project’s success.

1. Checklists
2. Decision-driver analysis
3. Assumption analysis
4. Decomposition

**Risk analysis:** assesses the loss probability and loss magnitude for each identified risk item.

1. Performance models
2. Cost models
3. Network analysis
4. Decision analysis
5. Quality-factor analysis

Risk prioritization: produces a ranked ordering of the risk items identified and analyzed.

1. Risk exposure: \( RE = P(UO) \times L(UO) \)
2. Risk leverage
3. Compound-risk reduction
Risk control

Risk management planning: helps prepare you to address each risk item.

1. Buying information
2. Risk avoidance
3. Risk transfer
4. Risk reduction
5. Risk-element planning
6. Risk-plan integration
Risk resolution: produces a situation in which the risk items are eliminated or otherwise resolved.

1. Prototypes

2. Simulations

3. Benchmarks

4. Analysis

5. Staffing
Risk control

Risk monitoring: involves tracking the project’s progress toward resolving its risk items and taking corrective action where appropriate.

1. Milestone tracking
2. Top 10 tracking
3. Risk reassessment
4. Corrective action
Top 10 Software Risk Items.

Personnel shortfalls:
   Staffing with top talent, job matching, team building, key personnel agreements, cross training.

Unrealistic schedules and budgets:
   Detailed multisource cost and schedule estimation, design to cost, incremental development, software reuse, requirements scrubbing.

Developing the wrong functions and properties:
   Organization analysis, mission analysis, operations-concept formulation, User surveys and user participation, prototyping, early users’ manuals, off-nominal performance analysis, quality-factor analysis.

Developing the wrong user interface:
   Prototyping, scenarios, task analysis, user participation.

Gold-plating:
   Requirements scrubbing, prototyping, cost-benefit analysis, design to cost.
Top 10 Software Risk Items.

Continuing stream of requirements changes:
High change threshold, information hiding, incremental development (deferring changes to later increments).

Shortfalls in externally furnished components:
Benchmarking, inspections, reference checking, compatibility analysis.

Shortfalls in externally performed tasks:
Reference checking, preaward audits, award-fee contracts, competitive design or prototyping, team-building.

Real-time performance shortfalls:
Simulation, benchmarking, modeling, prototyping, instrumentation, tuning.

Straining computer-science capabilities:
Technical analysis, cost-benefit analysis, prototyping, reference checking.