05. Software Development Methodologies & Agile Frameworks - Comprehensive Master Reference

> **Module Focus**: Complete SDLC Models (Waterfall, V-Model, Spiral Risk Math & RRL), Agile Manifesto (4 Values & 12 Principles), Scrum Framework (Roles, Artifacts, Ceremonies & Commitments), Kanban Mechanics (Little's Law, WIP, Lead/Cycle Time), and Worked Numerical Examples for Velocity, Burndown/Burnup Charts, and Risk Assessment.

---

1. Traditional Software Development Life Cycle (SDLC) Models

1.1 The 6 Core Phases of SDLC

1. **Requirements Gathering & Analysis**: Captures functional and non-functional requirements. Produces the **Software Requirement Specification (SRS)** document.
2. **System Design**: Translates SRS into system architecture.
3. **High-Level Design (HLD)**: System architecture, database schemas, network topology, module relationships.
4. **Low-Level Design (LLD)**: Detailed module logic, algorithms, class diagrams, API signatures.
5. **Implementation / Coding**: Source code construction adhering to coding standards and guidelines.
6. **Testing**: Verification and Validation of software against SRS requirements (Unit, Integration, System, Acceptance).
7. **Deployment**: Release of software into staging/production environments (Blue-Green, Canary, Rolling).
8. **Maintenance**: Post-release support:
9. **Corrective**: Fixing bugs discovered in production.
10. **Adaptive**: Modifying software to fit changed environment/OS/hardware.
11. **Perfective**: Enhancing performance, maintainability, or adding new features.
12. **Preventive**: Refactoring code to prevent future defects.

1.2 Comprehensive SDLC Model Comparison Matrix

Model	Primary Principle	Strengths	Weaknesses	Ideal Use Case
Waterfall	Sequential, phase-by-phase execution	Simple, clear milestones, easy project management	Zero flexibility, high late-phase risk, software visible late	Small projects with locked, stable requirements
V-Model	Verification & Validation parallel planning	Early test planning, high defect detection, rigorous QA	Rigid, expensive to change requirements mid-way	Safety-critical software (Medical, Aerospace, Automotive)

Model	Primary Principle	Strengths	Weaknesses	Ideal Use Case
Iterative / Incremental	Build software in small manageable releases	Working product early, easier risk handling, early feedback	Architecture degradation over iterations if unmanaged	E-commerce applications, commercial software
Spiral Model	Risk-driven iterative prototyping	High risk control, systematic prototyping, adaptive	Expensive, requires high risk-assessment expertise	Mission-critical, large, high-budget complex software
Prototyping Model	Build UI/logic mock-ups to refine requirements	Reduces requirement ambiguity, high user involvement	Scope creep, developers attached to temporary prototype	Systems with unclear user interface or workflow rules

---

1.3 Detailed Model Mechanics & Formulas

1. Spiral Model Risk Math & Formulas:

The Spiral model is structured into 4 quadrant phases per loop: 1. **Determine Objectives & Alternatives**: Identify constraints and target performance goals. 2. **Identify & Resolve Risks**: Risk analysis, mathematical modeling, prototyping, and benchmark testing. 3. **Development & Testing**: Build and verify the next level of the software product. 4. **Plan Next Phase**: Review results with customer and plan the next spiral iteration.

Key Formulas: - **Risk Exposure (RE)**: $$RE = P(\text{Risk Event}) \times \text{Cost of Risk Impact}$$ *Where $P(\text{Risk Event})$ is probability ($0 \le P \le 1$), and Cost is monetary or schedule loss.*

- **Risk Reduction Leverage (RRL)**: $$RRL = \frac{RE_{\text{before}} - RE_{\text{after}}}{\text{Cost of Risk Reduction Strategy}}$$ *Decision Rule: If $RRL > 1.0$, the risk reduction strategy is cost-effective and recommended for implementation.*

2. V-Model Verification & Validation (V&V) Mapping:

```
Requirements Analysis <==================================> Acceptance Testing
       |                                                        ^
       v                                                        |
System Design <=========================================> System Testing
       |                                                        ^
       v                                                        |
Architecture Design <==============================> Integration Testing
       |                                                        ^
       v                                                        |
Module Design <======================================> Unit Testing
       |                                                        ^
       +------------------> [ CODING ] ------------------------+
```

- **Verification** ("Are we building the product right?"): Static analysis, code reviews, inspections, walkthroughs (Left side of V).

- **Validation** ("Are we building the right product?"): Dynamic testing, executing code against requirements (Right side of V).

---

2. Agile Methodology & Scrum Framework

2.1 The 4 Agile Values & 12 Principles

The 4 Core Agile Manifesto Values:

1. **Individuals and interactions** over processes and tools.
2. **Working software** over comprehensive documentation.
3. **Customer collaboration** over contract negotiation.
4. **Responding to change** over following a plan.

Summary of the 12 Agile Principles:

1. Highest priority is customer satisfaction through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently (weeks to months, preferring shorter timescales).
4. Business people and developers must work together daily.
5. Build projects around motivated individuals; give them support and trust.
6. Face-to-face conversation is the most effective method of conveying information.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development (constant pace indefinitely).
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. Self-organizing teams produce the best architectures, requirements, and designs.
12. At regular intervals, the team reflects on how to become more effective and tunes behavior accordingly.

2.2 Scrum Roles, Artifacts, Ceremonies & Commitments

```
+-------------------------------------------------------------------------------------------
-------+
|                                    SCRUM
FRAMEWORK                                          |
+-------------------+--------------------+-----------------------
+------------------------------+
| ROLES             | ARTIFACTS          | COMMITMENTS           |
CEREMONIES                   |
+-------------------+--------------------+-----------------------
+------------------------------+
| - Product Owner   | - Product Backlog  | -> Product Goal       | - Sprint
Planning            |
| - Scrum Master    | - Sprint Backlog   | -> Sprint Goal        | - Daily
Standup              |
```

```
| - Dev Team (3-9)  | - Increment        | -> Definition of Done | - Sprint
Review              |
|                   |                    |                       | - Sprint
Retrospective       |
|                   |                    |                       | - Backlog
Refinement         |
+-------------------+--------------------+-----------------------
+-----------------------------+
```

1. Scrum Roles:

- **Product Owner (PO)**: Maximizes product value. Accountable for Product Backlog management, ordering items by business value, defining user story acceptance criteria, and accepting/rejecting work increments.
- **Scrum Master (SM)**: Accountable for establishing Scrum. Facilitator and servant leader who removes impediments/blockers, shields the team from external distractions, and coaches the team on Scrum practices.
- **Development Team**: Cross-functional, self-organizing professionals (typically 3 to 9 members) responsible for delivering a potentially shippable Increment every Sprint.

2. Scrum Artifacts & Associated Commitments:

1. **Product Backlog**: Ordered list of everything needed in the product.
2. *Commitment*: **Product Goal** (Long-term target state of the product).
3. **Sprint Backlog**: Selected Product Backlog items for current Sprint + plan to deliver them.
4. *Commitment*: **Sprint Goal** (Single objective set for the Sprint).
5. **Increment**: Sum of all items completed during a Sprint, merged with previous increments.
6. *Commitment*: **Definition of Done (DoD)** (Formal quality standard required for work to be released).

3. Scrum Ceremonies (Timeboxes based on a 4-week Sprint):

- **Sprint Planning** (Max 8 hours for 4-week Sprint; ~4 hours for 2-week Sprint):
- *Topic 1*: Why is this Sprint valuable? (Sprint Goal)
- *Topic 2*: What can be Done? (Selecting backlog items)
- *Topic 3*: How will the chosen work get done? (Decomposing into tasks)
- **Daily Standup / Daily Scrum** (Strictly 15 minutes daily):
- Team sync focused on progress toward Sprint Goal.
- 3 Core Questions: What was completed yesterday? What will be done today? Are there any impediments?
- **Sprint Review** (Max 4 hours for 4-week Sprint; ~2 hours for 2-week Sprint):
- Demonstration of working increment to PO and key stakeholders. Collect feedback and adapt Product Backlog.
- **Sprint Retrospective** (Max 3 hours for 4-week Sprint; ~1.5 hours for 2-week Sprint):
- Internal inspection of process, team dynamics, continuous integration, and tools. Identifies top actionable improvements for the next Sprint.
- **Backlog Refinement / Grooming** (Ongoing, max 10% of team capacity):

- Breaking down large user stories (Epics), adding estimates (Story Points), and defining acceptance criteria.

2.3 Kanban Mechanics & Flow Metrics

- **Work-in-Progress (WIP) Limits**: Restricts the maximum number of task items allowed in any single workflow state simultaneously to prevent bottlenecks and multitasking overhead.
- **Little's Law for Software Flow**: $$\text{Cycle Time} = \frac{\text{Work-in-Progress (WIP)}}{\text{Throughput}}$$ $$\text{Throughput} = \frac{\text{WIP}}{\text{Cycle Time}}$$
- **Lead Time vs. Cycle Time**:
- **Lead Time**: Time elapsed from customer request creation to final delivery.
- **Cycle Time**: Time elapsed from when work actively starts on a task to its completion.

---

3. Worked Numerical Examples

✏️ Worked Numerical Example 1: Spiral Model Risk Math & Risk Reduction Leverage (RRL)

Scenario:

A financial transaction processing engine has a potential risk of security vulnerability (R_1) and database deadlock (R_2). - **Vulnerability (R_1)**: - Probability $P_1 = 0.20$ (20%) - Impact / Cost of Failure $C_1 = \$500,000$ - Mitigation Option A: Implement Web Application Firewall (WAF) & Security Audit costing $\$30,000$. After implementation, P_1 drops to 0.02 (2%). - **Deadlock (R_2)**: - Probability $P_2 = 0.15$ (15%) - Impact / Cost of Failure $C_2 = \$200,000$ - Mitigation Option B: Redesign concurrency locking algorithm costing $\$15,000$. After implementation, P_2 drops to 0.05 (5%).

Step 1: Calculate Initial Risk Exposure (RE_{before})

- $RE_{1,\text{before}} = P_1 \times C_1 = 0.20 \times \$500,000 = \$100,000$
- $RE_{2,\text{before}} = P_2 \times C_2 = 0.15 \times \$200,000 = \$30,000$
- **Total Initial Risk Exposure** $= \$100,000 + \$30,000 = \mathbf{\$130,000}$

Step 2: Calculate Post-Mitigation Risk Exposure (RE_{after})

- $RE_{1,\text{after}} = 0.02 \times \$500,000 = \$10,000$
- $RE_{2,\text{after}} = 0.05 \times \$200,000 = \$10,000$

Step 3: Calculate Risk Reduction Leverage (RRL) for both options

- **RRL for Option A (Vulnerability Mitigation)**: $$RRL_A = \frac{RE_{1,\text{before}} - RE_{1,\text{after}}}{\text{Cost of Risk Reduction}} = \frac{\$100,000 - \$10,000}{\$30,000} = \frac{\$90,000}{\$30,000} = \mathbf{3.0}$$

- **RRL for Option B (Deadlock Mitigation)**: $$RRL_B = \frac{RE_{2,\text{before}} - RE_{2,\text{after}}}{\text{Cost of Risk Reduction}} = \frac{\$30,000 - \$10,000}{\$15,000} = \frac{\$20,000}{\$15,000} = \mathbf{1.33}$$

Conclusion & Decision:

- Both options have $RRL > 1.0$, meaning both mitigations are cost-effective.
- **Option A** has a higher leverage (3.0 vs 1.33), yielding $\$3.00$ of risk reduction for every $\$1.00$ spent. Option A should be prioritized in Spiral Phase 2.

✏️ Worked Numerical Example 2: Team Velocity & Release Forecasting

Scenario:

An Agile team completes the following work across 4 historical Sprints: - **Sprint 1**: 30 Story Points committed, 22 Story Points completed. - **Sprint 2**: 28 Story Points committed, 25 Story Points completed. - **Sprint 3**: 25 Story Points committed, 25 Story Points completed. - **Sprint 4**: 32 Story Points committed, 28 Story Points completed.

The remaining Product Backlog is **160 Story Points**. Midway through planning, the Product Owner adds **20 Story Points** of urgent new user stories due to market changes.

Step 1: Calculate Average Velocity (V_{avg})

$$V_{\text{avg}} = \frac{\text{Completed Points in Sprints } (1 + 2 + 3 + 4)}{4}$$ $$V_{\text{avg}} = \frac{22 + 25 + 25 + 28}{4} = \frac{100}{4} = \mathbf{25 \text{ Story Points / Sprint}}$$

Note: Velocity is always calculated strictly using completed story points, NOT committed points.

Step 2: Calculate Release Duration (Sprints)

- Total Product Backlog after scope addition $= 160 + 20 = 180 \text{ Story Points}$. $$\text{Estimated Sprints to Completion} = \frac{\text{Total Backlog Points}}{V_{\text{avg}}} = \frac{180}{25} = \mathbf{7.2 \text{ Sprints}}$$

Since Sprints are full iterations, round up: **8 Sprints required to deliver the backlog**.

✏️ Worked Numerical Example 3: Burndown Chart Mechanics & Ideal vs. Actual Tracking

Scenario:

- **Sprint Length**: 10 Working Days (2 weeks).
- **Committed Sprint Backlog**: 50 Story Points.

Step 1: Calculate Ideal Daily Burndown Rate

$$\text{Ideal Daily Rate} = \frac{\text{Total Story Points}}{\text{Total Days}} = \frac{50}{10} = \mathbf{5 \text{ Story Points / day}}$$

Step 2: Daily Tracking Table

Day	Ideal Remaining (pts)	Actual Remaining (pts)	Daily Burn (pts)	Status / Analysis
0	50	50	-	Sprint Start
1	45	50	0	Team blocked on environment setup
2	40	46	4	Slow progress
3	35	40	6	Catching up
4	30	33	7	Good momentum
5	25	28	5	Mid-Sprint: **Behind schedule by 3 pts**
6	20	20	8	Large user story completed
7	15	14	6	Ahead of schedule by 1 pt
8	10	18	-4 (Scope added)	**Scope Creep**: PO added 8 pts story
9	5	10	8	High effort push
10	0	2	8	**2 pts incomplete** (Carried to next Sprint)

Key Insights & Metrics:

- **Day 5 Evaluation**: Actual (28 pts) > Ideal (25 pts), indicating the team is **3 pts behind**.
- **Day 8 Scope Change**: Actual remaining increased from 14 to 18 points due to $+8$ pts scope addition.
- **Sprint Outcome**: 48 points completed out of 58 total (50 initial + 8 added). Velocity for this Sprint $= 48 \text{ Story Points}$.

✏️ Worked Numerical Example 4: Burnup Chart vs. Burndown Chart Scope Addition Comparison

Scenario:

- Initial Backlog $= 100 \text{ pts}$.
- At Sprint 3, total backlog increases by $+30 \text{ pts}$ to 130 pts.
- Team completes 20 pts per Sprint.

Comparative Metric Analysis:

Feature	Burndown Chart	Burnup Chart
Primary Line	Shows **Remaining Work** decaying to 0	

Feature	Burndown Chart	Burnup Chart
		Shows **Completed Work** rising to Total Scope line
Handling Scope Increase	The remaining work line spikes upward, masking team velocity progress	The **Total Scope line shifts upward**, keeping completed work trend line clean
Visibility	Excellent for day-to-day sprint remaining effort	Excellent for multi-sprint project/release scope change tracking

```
BURNUP CHART (Sprint 1 to 5):
Story Points
140 |--------------------------------------- [ Total Scope Line = 130 ]
120 |                                       /
100 |------------------- [ Initial Scope = 100 ]
 80 |                                 /
 60 |                          /
 40 |                   /
 20 |            / (Completed Work)
  0 +------------+------------+------------+------------+------------>
     Sprint 0     Sprint 1     Sprint 2     Sprint 3     Sprint 4
```