Software Life-Cycle Models

A software life-cycle model is a structure imposed on the development of a software product. There are several different models available, each describing relations and orders of a variety of tasks or activities that take place during the life-cycle.

- Life-cycle model (formerly, process model)
- The steps through which the product progresses are:
  - Requirements phase
  - Specification phase
  - Design phase
  - Implementation phase
  - Integration phase
  - Maintenance phase
  - Retirement

Software Life-Cycle Models

- Build-and-fix model
- Waterfall model
- Prototyping model
- Incremental model
- eXtreme programming
- Synchronize-and-stabilize model
- Spiral model
- Object-Oriented model
- Unified Process Model
- Open source development
Build and Fix Model

- Problems
  - No specifications -- No design
  - Totally unsatisfactory
  - Need life-cycle model
    - "Game plan"
    - Phases
    - Milestones

The Waterfall Model

- Characterised by
  - feedback loops
  - documentation-driven
- Advantages
  - documentation
  - maintenance easier
- Disadvantages
  - customers can't read your documents and understand what software will be delivered
  - difficulties of accommodating change after the process is underway
  - one phase has to be complete before moving onto the next phase.

Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements
- Therefore, this model is only appropriate when the requirements are well understood and changes will be fairly limited during the design process
- Few business systems have stable requirements
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites
Prototyping Model

Exploratory development
- Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements.

Throw-away prototyping
- Objective is to understand the system requirements. Should start with poorly understood requirements.

Evolutionary Development

- Problems
  - Lack of process visibility
  - Systems are often poorly structured
  - Special skills (e.g. in languages for rapid prototyping) may be required

- Applicability
  - For small or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime systems

Three Key Points

- Do not turn rapid prototype into product
- Rapid prototyping may replace specification phase — never the design phase
- Comparison:
  - Waterfall model — try to get it right first time
  - Rapid prototyping — frequent change, then discard

Waterfall and Rapid Prototyping Models

- Waterfall model
  - Many successes
  - Client needs
- Rapid prototyping model
  - Not proved
  - Has own problems
- Solution
  - Rapid prototyping for requirements phase
  - Waterfall for rest of life cycle
Incremental Model

- Divide project into “builds”

Incremental Delivery

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental Model, cont’d

- Waterfall, rapid prototyping models
  - Operational quality complete product at end
- Incremental model
  - Operational quality portion of product within weeks
  - Less traumatic
  - Smaller capital outlay, rapid return on investment
  - Need open architecture – maintenance implications
  - Different approaches – top-down, bottom-up, middle-out, use-case-driven

Incremental Development Advantages

- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.
Incremental Model, cont’d

- Problems:
  - Build-and-fix danger
  - Contradiction in terms
  - Same problems as waterfall model, only a little at a time

Experts handle different parts of development

More risky version -- pieces may not fit

eXtreme Programming

- Somewhat controversial “new” approach
- An approach to development based on the development and delivery of very small increments of functionality
- Relies on constant code improvement, user involvement in the development team and pair-wise programming
Unusual Features of XP

- XP team work in the same room
- Client representative is always present
- No overtime work
- Pair programming
- Refactoring

The Agile Manifesto

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

Synchronise-and-stabilise Model

- Microsoft's life-cycle model
- Requirements analysis—interview potential customers
- Draw up overall product specifications
- Divide project into 3 or 4 builds
- Each build is carried out by small teams working in parallel

Synchronise-and-stabilise Model, cont'd

- At the end of the day—synchronise (test and debug)
- At the end of the build—stabilise (freeze build)
- Components always work together
  - Get early insights into operation of product
Spiral Model

- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

Simplified Spiral Model

- If risks cannot be resolved, project is immediately terminated.

Full Spiral Model

- Radial dimension: cumulative cost to date
- Angular dimension: progress through the spiral

Analysis of Spiral Model

- **Strengths**
  - Easy to judge how much to test
  - No distinction made between development and maintenance
- **Weaknesses**
  - For large-scale software only
  - For internal (in-house) software only
Unified Process Model

Unified Process aims to:
- Meet user needs
- Accommodate risks
The method makes heavy use of:
- Use cases
- Iteration
- Software architecture
Four phases:
- Inception
- Elaboration
- Construction
- Transition

Unified Process Model, cont’d

Techniques used:
- Iteration
- Use cases
- Working architecture early on
- Components
- Effective team
- Quality

Activity cost distribution

<table>
<thead>
<tr>
<th>Method</th>
<th>0</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>5%</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>100</td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration and testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iteration development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>0%</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>100</td>
</tr>
<tr>
<td>Iterative development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component-based software engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>0%</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>100</td>
</tr>
<tr>
<td>Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration and testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development and evolution costs for long-lifetime systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System evolution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Waterfall model

<table>
<thead>
<tr>
<th>Method</th>
<th>0</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>5%</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>100</td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration and testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iteration development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>0%</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>100</td>
</tr>
<tr>
<td>Iterative development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component-based software engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>0%</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>100</td>
</tr>
<tr>
<td>Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration and testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development and evolution costs for long-lifetime systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System evolution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pros and Cons of Lifecycle Models

- Manager’s view:
  - defines a set of deliverables
  - sounds good to be able to tell upper management that “design stage is completed”
  - required by many organisations to get contracts

- Programmer’s view:
  - doesn’t represent what really happens “on the floor”
  - customers can’t adequately state requirements up-front
  - stages become intermixed

Conclusions

- Different life-cycle models
- Each with own strengths
- Each with own weaknesses
- Criteria for deciding on a model include
  - The organisation
  - Its management
  - Skills of the employees
  - The nature of the product
- Best suggestion
  - “Mix-and-match” life-cycle model

Discuss with your three closest neighbours:

- Have you ever worked in a project using, or trying to use a life-cycle model to structure the work?
- What kind of life-cycle model did you use?
- Did the life-cycle model support your work or was it a burden?

The Software Process

The software process includes a Software Life-Cycle Model; methodologies, tools and techniques used and the people performing the work.
Overview

- A structured set of activities required to develop a software system
  - Specification
  - Design
  - Validation
  - Evolution
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

The Software Process

- The life-cycle model
- Tools and methodologies
- The individuals

Planning Phase?

- There is NO planning phase
- Project begins with preliminary planning to manage requirements and analysis phases
- Once it is known what is to be developed, a project plan is created
  - Budget
  - Staffing requirements
  - Schedule
- This project plan is watched and updated during the whole project

Testing Phase?

- There is NO testing phase
- Testing is an activity performed throughout software production
  - Verification
    - Performed at the end of each phase
  - Validation
    - Performed before delivering the product to the client
Documentation Phase?

- There is NO documentation phase
- Every phase must be fully documented before starting the next phase
  - Postponed documentation may never be completed
  - The responsible individual may leave
  - The product is constantly changing — we need the documentation to do this
  - The design (for example) will be modified during development, but the original designers may not be available to document it

Requirements Phase

- Assumption
  - The software being considered is economically justifiable
- Concept exploration
  - Determine what the client needs, not what the client wants
- Moving target problem

Requirements Phase Testing

- Rapid prototyping

Requirements Phase Documentation

- Rapid prototype, or
- Requirements document
Specification Phase

- Specifications document ("specifications")
  - Legal document
  - Must not have phrases like "optimal," or "98% complete"
- Specifications must not be:
  - Ambiguous
  - Incomplete
  - Contradictory
- Once the specifications have been signed off:
  - The project plan is drawn up
  - This is the earliest possible time for the project plan

Specification Phase Documentation

- Specification document (specifications)
- Project plan

Specification Phase Testing

- Traceability
- Review
- Check the project plan

Design Phase

- Specification—what
- Design—how
- Retain design decisions:
  - When a dead-end is reached
  - For the maintenance team
  - Ideally, the design should be open-ended
- Architectural design:
  - Decompose the product into modules
- Detailed design:
  - Design each module: data structures, algorithms
Design Phase Testing

- Traceability
- Review

Design Phase Documentation

- Design
  - Architectural design
  - Detailed design

Implementation Phase

- Implement the detailed design in code

Implementation Phase Testing

- Review
- Test cases
  - Informal testing (desk checking)
  - Formal testing (SQA)
Implementation Phase Documentation

- Source code
- Test cases (with expected output)

Integration Phase

- Combine the modules and check the product as a whole

Integration Phase Testing

- Product testing
- Acceptance testing

Integration Phase Documentation

- Commented source code
- Test cases for the product as a whole
Maintenance Phase

- Maintenance
  - Any change once the client has accepted the software
- The most money is devoted to this phase
- The problem of lack of documentation

Testing

- Regression testing

Documentation

- Record of all changes made, with reasons
- Regression test cases

Retirement

- Good software is maintained
- Sometimes software is rewritten from scratch
  - Software is now unmaintainable because
    - A drastic change in design has occurred
    - The product must be implemented on a totally new hardware/operating system
    - Documentation is missing or inaccurate
  - Hardware is to be changed—it may be cheaper to rewrite the software from scratch than to modify it
- True retirement is a rare event
Inherent Problems of Software Production

- Hardware has inherent limits
- So does software
- No Silver Bullet
  - Complexity
  - Conformity
  - Changeability
  - Invisibility

Complexity

- Software is far more complex than hardware
  - Traditional abstraction will not work
  - We cannot understand the whole, so we cannot understand any part
  - Management is difficult
  - Maintenance is a nightmare (documentation, too)

Changeability

- Software is easier to change than hardware
- Pressure to change
  - Reality
  - Useful software
  - Easier to change
  - Software has a long lifetime (~15 yrs) compared to hardware (~4 yrs)

Invisibility

- Software is invisible and unvisualisable
- Complete views are incomprehensible
- Partial views are misleading
- However, all views can be helpful
Is There a Silver Bullet?

- What about
  - High-level languages
  - Time sharing
  - CASE tools
- These did not solve the intrinsic problems
- But, no “silver bullet” (order-of-magnitude increase) is possible

Improving the Software Process

- U.S. Department of Defense initiative
- Software Engineering Institute (SEI)
- The fundamental problem with software
  - The software process is badly managed
  - Software process improvement initiatives
  - Capability maturity model (CMM)
  - ISO 9000-series
  - ISO/IEC 15504

Capability Maturity Model

- Not a life-cycle model
- Set of strategies for improving the software process
  - SW–CMM for software
  - P–CMM for human resources ("people")
  - SE–CMM for systems engineering
  - IPD–CMM for integrated product development
  - SA–for software acquisition
- These strategies are being unified into CMMI (capability maturity model integration)

SW–CMM

- A strategy for improving the software process
- Put forward in 1986 by the SEI
- Fundamental idea:
  - Improving the software process leads to
    - Improved software quality
    - Delivery on time, within budget
  - Improved management leads to
    - Improved techniques
- Five levels of “maturity” are defined
  - Organisation advances stepwise from level to level
SW-CMM Level 1. Initial Level

- Ad hoc approach
  - Entire process is unpredictable
  - Management consists of responses to crises
- Most organisations world-wide are at level 1

SW-CMM Level 2. Repeatable Level

- Basic software management
  - Management decisions should be made on the basis of previous experience with similar products
  - Measurements (“metrics”) are made
  - These can be used for making cost and duration predictions in the next project
  - Problems are identified, immediate corrective action is taken

SW-CMM Level 3. Defined Level

- The software process is fully documented
  - Managerial and technical aspects are clearly defined
  - Continual efforts are made to improve quality and productivity
  - Reviews are performed to improve software quality
  - CASE-tools are applicable now (and not for level 1 or 2)

SW-CMM Level 4. Managed Level

- Quality and productivity goals are set for each project
  - Quality, productivity are continually monitored
  - Statistical quality controls are in place
SW-CMM Level 5. Optimizing Level

- Continuous process improvement
- Statistical quality and process controls
- Feedback of knowledge from each project to the next

Experience

- It takes:
  - 3 to 5 years to get from level 1 to level 2
  - 1.5 to 3 years from level 2 to level 3
  - SEI questionnaires highlight shortcomings, suggest ways to improve the process
- Original idea: U.S. Defense contracts would be awarded only to capable firms

SW-CMM Key Process Areas

- There are key process areas (KPAs) for each level
- Level 2 KPAs include:
  - Requirements management
  - Project planning
  - Project tracking
  - Configuration management
  - Quality assurance
- Compare
  - Level 2: Detection and correction of faults
  - Level 5: Prevention of faults

SW-CMM Praise

- It has arguably been successful in this role, even reputedly causing some software sales people to clamour for their organisations to "implement CMM."
- Economic development agencies in India, Ireland, Egypt, Syria, and elsewhere have praised the CMM for enabling them to be able to compete for U.S. outsourcing contracts on an even footing.
- The CMM provides a good framework for organisational improvement. It allows companies to prioritise their process improvement initiatives.
SW-CMM Criticism

- CMM failed to take over the world.
- CMM is well suited for bureaucratic organisations such as government agencies, large corporations and regulated monopolies.
- The use of auditors and executive reports may influence the entire IT organisation to focus on perfectly completed forms rather than application development, client needs or the marketplace.

Other SPI Initiatives

- Other software process improvement (SPI) initiatives:
  - ISO 9000-series
  - ISO/IEC 15504

ISO 9000

- Set of five standards for industrial activities
- ISO 9001 for quality systems
- ISO 9000-3, guidelines to apply ISO 9001 to software
- There is an overlap with CMM, but they are not identical -- ISO 9000 is not process improvement
- Stress on documenting the process
- Emphasis on measurement and metrics
- ISO 9000 is intended to make sure that the product—any product—has been produced in the most efficient and effective manner possible.
- ISO 9000 is required to do business with the E.U.
ISO 9000 Criticism

- The compliance process is costly and time-consuming.
- Lots of administration is needed to implement it.
- Adhering to ISO 9000 makes processes more consistent; to some proponents of continuous improvement, it also makes it harder to improve and readapt the processes.

Agile manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value

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

That is, while there is value in the items on the right, we value the items on the left more.


Agile principles

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals.
- Give them the environment and support they need and trust them to get the job done.

Agile principles, forts.

- Working software is the primary measure of progress.
- Agile processes promote sustainable development.
- The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximising the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organising teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.
What’s the Difference?

- Approach built on adaptation rather than prediction
- Focus on change rather than trying to prevent it from happening
- This doesn’t mean that the work lacks neither structure or discipline
  - Well-defined processes are followed, which makes a big difference between agile development and “wild-west-hacking”

XP -- eXtreme Programming

- Combines a number of well-known methodologies for software development to a new whole
- Uses programming and programmer work as its starting point, not management and manager work

XP -- Well-known Methodologies

- Fine-grained feedback
  - Pair-programming
  - “Planning Game”
  - Test-driven development
  - The whole team
- Process constantly running
  - Constant integration
  - Design improvements
  - Small releases

XP -- Well-known Methodologies, cont’d

- Common understanding
  - Code standard
  - Common code ownership
  - Simple design
  - System metaphor
- Programmers well-being
  - Sustainable work pace
End of Today’s Lecture

Thanks for your attention!