Software Life Cycle

“What happens in the ‘life’ of software”

Lecture Objectives

❖ To look at the ‘life cycle’ of a software
❖ To understand the software process and its related elements
❖ To relate to the different software engineering paradigms

Software Engineering Layers

- tools
- methods
- process model
- a “quality” focus

Software Process

❖ Activities in software projects
❖ Characterised by a common process framework
  ❖ Framework activities - task sets
  ❖ Umbrella activities
❖ “Process maturity” enables development of quality software products

Common Process Framework

<table>
<thead>
<tr>
<th>Common process framework</th>
<th>Framework activities</th>
<th>Work tasks</th>
<th>Work products</th>
<th>Milestones &amp; deliverables</th>
<th>QA checkpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Umbrella Activities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Umbrella Activities

❖ Software project management
❖ Formal technical reviews
❖ Software quality assurance
❖ Software configuration management
❖ Document preparation and production
❖ Reusability management
❖ Measurement
❖ Risk management
**Process as Problem Solving**

![Diagram: Process as Problem Solving]

**The Process Model: Adaptability**

- the framework activities will always be applied on every project ... BUT
- the tasks (and degree of rigor) for each activity will vary based on:
  - the type of project (an "entry point" to the model)
  - characteristics of the project
  - common sense judgment; concurrence of the project team

**The Primary Goal: High Quality**

Remember:

- High quality = project timeliness
- Why?
- Less rework!

**The Systematic Process**

- Analysis → Problem
- Design → Models
- Development → Solution
- Testing

**Generic Phases**

- Definition Phase
  - Focus on 'what' the software is
- Development Phase
  - Focus on 'how' the software works
- Maintenance Phase
  - Focus on 'change' to the software

**Definition Phase**

- Identify information to be processed
- Identify system behaviour - functions and performance
- Determine constraints, interfaces, validation criteria
- Major tasks:
  - System engineering
  - Software project planning
  - Requirements analysis
Development Phase

- Define data structures, function implementation, procedural details, interfaces
- Translate design to programming language
- How testing is performed
- Major tasks:
  - Software design
  - Code generation
  - Software testing

Maintenance Phase

- Reapplies definition and development phases to existing software
- Types of changes:
  - Correction
  - Adaptation
  - Enhancement
  - Prevention

Waterfall Model

- System Engineering
- Analysis
- Design
- Code
- Testing
- Maintenance

Waterfall Model Characteristics

- The classic life cycle - oldest and most widely used paradigm
- Activities 'flow' from one phase to another
- If there are corrections, return to a previous phase and 'flow' from there again
- Major advantages: Good for planning and well-defined/repeated projects

Problems of Waterfall Model

- Real projects often follow the sequence
- All requirements may not be stated explicitly by customer
- Customer only sees the results after some time
- Developers are often delayed at certain phases

Prototyping Model

- Start
- Requirements gathering and refinement
- Quick design
- Building prototype
- Customer evaluation
- Customer feedback
- Refining prototype
- Stop
Prototyping Model Characteristics

- Developer and customer determine objectives and draft requirements
- Prototype quickly produced and evaluated by customer
- Prototype then refined, and re-evaluated
- Process iterated, before final product development
- Advantages: Customer participation and better requirements

Problems of Prototyping Model

- Problem 1: Customer may see prototype as working model and expects fast results
- Problem 2: Developer compromised when producing prototype quickly, e.g. different operating system or programming language

Rapid Application Development (RAD)

- "High-speed" version of waterfall model
- Primarily for information systems applications
- Requirements well-understood, fully functional system produced in short time
- The application modularised - major functions can be completed in 3 months
- Separate teams complete the functions, then integrated as a whole
- Requires human resource and commitment

Incremental Model Characteristics

- Software separated into different "increments" - complete working portions
- Focus on delivery of operational product with each increment - can be evaluated
- Useful when insufficient staff and can be planned to manage technical risks, e.g. waiting for new hardware
**Spiral Model**

- **Characteristics**
  - Originally proposed by Boehm, couples iterative nature of prototyping and the systematic aspects of waterfall model
  - Software is developed in series of incremental releases
  - Each iteration produces a more complete product
  - Better management through risk analysis

---

**Problems of Spiral Model**

- May be difficult to convince customers that evolution is controllable
- Demands risk assessment expertise - major risk will cause problems if not identified
- Relatively new and not widely used - cannot determine performance

---

**Component Assembly Model**

- Use of object-oriented technology
- Components - classes that encapsulate both data and algorithms
- Components developed to be reusable
- Paradigm similar to spiral model, but engineering activity involves components
- System produced by assembling the correct components

---

**Fourth Generation Techniques (4GT)**

- Requirements gathering
- "Design" strategy
- Implementation using 4GL
- Testing
4GT Characteristics

- Use of software tools that allow software engineer to specify s/w characteristics at higher level
- The tools generate codes based on specification
- More time in design and testing - increase productivity
- Tools may not be easy to use, codes generated may not be efficient

Other Process Models

- Component assembly model—the process to apply when reuse is a development objective
- Concurrent process model—recognizes that different part of the project will be at different places in the process
- Formal methods—the process to apply when a mathematical specification is to be developed
- Cleanroom software engineering—emphasizes error detection before testing

Conclusion

- The paradigm used for development of software depends on a number of factors
  - People - staff & users
  - Software product
  - Tools available
  - Environment
- Existing models makes development process clearer, but they can be evolved to become new paradigms

References