5
The Second-System Effect

The Second-System Effect

- Interactive Discipline for the Architect
- Self-Discipline – The Second-System Effect

Negotiation

- Architects propose projects
- Implementers propose bids
- Architects negotiate function up and cost down
- Implementers are the implementation authority
- Architects
  - Only suggests implementation
  - Must accept viable alternatives
  - Should deal quietly; Forego credit

Self-discipline

- First system requires humility from ignorance
- Second system is the most dangerous
  - Confidence from first success dominates
  - Designer attempts all former forgone features
- Third and subsequent gain wisdom from second

Contrast with Extreme Programming

- All programmers participate in all technical roles
- Programmers work in pairs, i.e., one screen
- Function is developed incrementally
- Refactoring addresses architectural issues

A Development Episode in XP

- From Chapter 2 of Kent Beck's *Extreme Programming Explained* (1st ed.)
- "Day-to-day programming proceeds from a task clearly connected to a feature the customer wants, to test, to implementation, to design, and through to integration. A little of each of the activities of software development are packed into each episode."

The scenario

- Larger team project, but focus on ...
- Two programmers, Kent (*I*) and Beck (*You*)
- One task, *Export Quarter-to-date Witholding*
- After the daily morning stand-up meeting (that all programmers attend), Kent asks Beck, "Can you help me on the *Export* task?"
- Beck says "Yes,"
  - All programmers help when asked

New test cases

- Beck asks, "What are the test cases?"
- Kent answers, "The values in the export record should match the values in the bins."
- Beck asks, "Which fields have to be populated?"
- Interrupting briefly, Eddie explains the five fields related to quarter-to-date

Existing test cases

- Kent and Beck examine existing export test cases
- They
  - Find one that is almost what they need
  - Abstract a superclass
  - Refactor the dependent code
  - Run all existing test cases successfully

Further improvements

- They
  - See other export test cases that could use the new superclass
  - Record “Refactor AbstractExportTest” on their to-do card

New test case for the new code

- They
  - Use the new superclass
  - Create a new test case for the new code
- Kent suggests, “I thought of an implementation.”
- Beck answers, “Let's finish the test case.”
- Kent records three ideas on the to-do card
- They
  - Run the new test case; It fails

New code for the new test case

- Kent writes the new code
- They
  - Notice a few more applicable test cases
  - Record them on the to-do card
  - Run the first test case; It passes
- Kent writes new code for each new test case
  - Each case successively passes

Refactoring new code

- Beck observes opportunities to simplify
- Kent hands Beck the keyboard
- Beck
  - Refactors the new code
  - Reruns the existing test cases; They all pass
  - Continues to implement code for each new test case

Refactoring old test cases

- The to-do card eventually only shows “Refactor AbstractExportTest”
- They
  - Restructure the existing test cases to use the AbstractExportTest class
  - Run the restructured test cases; All pass

Integration

- They
  - Notice that the integration machine is free
  - Load the latest release from the repository
  - Load their changes
  - Run all test cases, old and new; One fails
- Beck remarks, “It's been a month since a test case failed at integration.”
- They isolate and correct the faulty code

Release

- They
  - Rerun all test cases; All Pass
  - Release the new code and test cases into the central repository

Extreme programming summary

- Pairs programming
- Test-driven development
  - Create test cases, which fail until code exists
  - Code until all test cases pass
  - Design until test cases cover all functionality
- Refactoring
  - Simplify old and new code and test cases
- Immediate integration

IEEE Std 1058 Project Management

- IEEE Std 1058-1998 *IEEE Standard for Software Project Management Plans (SPMP)*
  - "may be applied to any type of project"
  - "not restricted by the size, complexity or criticality"
  - "identifies the elements that should be in all SPMP's"

IEEE Std 1058 (cont. 2 of 3)

Front matter

Project summary
- Purpose, scope, objectives
- Assumptions and constraints
- Project deliverables
- Schedule and budget summary

Project organization
- External interfaces
- Internal structure
- Roles and responsibilities

Managerial process plans
- Start-up plan
  - Estimation plan
  - Staffing plan
  - Resource acquisition plan
  - Project staff training plan
- Work plan
  - Work activities
  - Schedule allocation
  - Resource allocation
  - Budget allocation

IEEE Std 1058 (cont. 3 of 3)

Managerial process plans (cont.)
- Control plan
  - Requirements control plan
  - Schedule control plan
  - Budget control plan
  - Quality control plan
  - Reporting plan
  - Metrics collection plan
- Risk management plan
- Closeout plan

Section 8. Additional plans

Technical process plans
- Process models
- Methods, tools and techniques
- Infrastructure plan
- Product acceptance plan

Supporting process plans
- Configuration management plan
- Verification and validation plan
- Documentation plan
- Quality assurance plan
- Reviews and audits
- Problem resolution plan
- Subcontractor management plan
- Process improvement plan

Waterfall lifecycle stages

Analysis

Specification

Design

Implementation

Integration

Testing

Release

Maintenance