Case Studies: Examples of Good and Bad Software Projects

By Philip Liew

October 27, 2003

Software Project Gone Bad!

- Case study: Therac-25

What Is a Therac-25?

- Linear accelerators used for medical purposes
- Created by Atomic Energy Canada Limited (AECL)
- Earlier models: Therac-6 and Therac-20

Older Models

- Previous models designed as stand-alone machines
- Software functionality was limited
- Added convenience
- Used a PDP 11

Therac-25 Features

- Designed to take advantage of software monitoring and control mechanisms
- Removal of hardware safety mechanisms
- Faith in software reliability rather than hardware
- Due to lack of complexity, and digital nature

Accident Timeline

- June 3, 1985 - first overdose in Marietta Georgia
- Patient suffered serious radiation burn (est. 15,000 – 20, 000 rads)
- Breast removed, and lost use of shoulder and arm
- Manufacturer response:
  - Improper scanning not possible

Accident Timeline

- July 26, 1985 – second overdose in Hamilton, Ontario
- Patient dies November 3, 1985
- Manufacturer response:
  - Hardware micro-switch assumed to be cause
  - AECL makes changes and notifies users of increased safety of 5 orders of magnitude

Accident Timeline

- December, 1985 – Third overdose in Yakima, Washington
- Abnormal skin reaction
- Manufacturer response:
  - Two page report ruling out operator and manufacturer error
  - Including statement indicating no previous occurrence of overdose

Accident Timeline

- March 21, 1986 – fourth overdose in Tyler, Texas
- Lost function of legs, arms, and vocal cords
- Died 5 months after accident
- Manufacturer response:
  - Insistence that no prior accidents occurred
  - Suggestion issue is an electrical problem

Accident Timeline

- April 7, 1986 – machine put back in use
- April 11, 1986 – Fifth overdose occurs again in Texas
- Severe overdose causes death 3 weeks later
- Manufacturer response:
  - Discovery of bug in user input entry routines
- FDA requires corrective action plan (CAP) and notification of Therac-25 users

Accident Timeline

- Multiple CAP revisions necessary due to user and FDA concern

Due to:

- Lack of redundant safety mechanisms
- Lack of proper error messages
- Lack of software evaluation
- Lack of software requirements, specifications, documentation, and test plans

Accident Timeline

- January 17, 1987 – sixth overdose occurs again at Washington
- FDA labels Therac-25 as defective under US law
- Manufacturer response:
  - Issues fix for specific software errors
  - Adds additional hardware safety checks suggested by user groups and FDA
  - Fifth revision of CAP approved

Specific Bugs

- Hard to isolate specific bugs due to lack of documentation of accidents
- User entry interface allowed for improper entry
- Increment roll-over which allowed safety checks to be bypassed
- Software allowed concurrent access to shared memory without synchronization. Race conditions emerged

Overall Issues

- Particular coding error is not as important as the general unsafe design of the system
- Look at overall contributing factors for failure

Attitudes

- Software considered trivial part of system
- Overconfidence in software
- Non-qualified person hired

Software Engineering Practices

- Unrealistic risk assessments
- Fault analysis did not include software
- Lack of independent software evaluation
- Little documentation on software specifications and software test plans
- Lack of documentation on product and User error messages
- Lack of software auditing

Management Process

- Lack of formal management and quality control processes
- Lack of procedures on following through on reported accidents
- Bug fixes corrected only when occurred

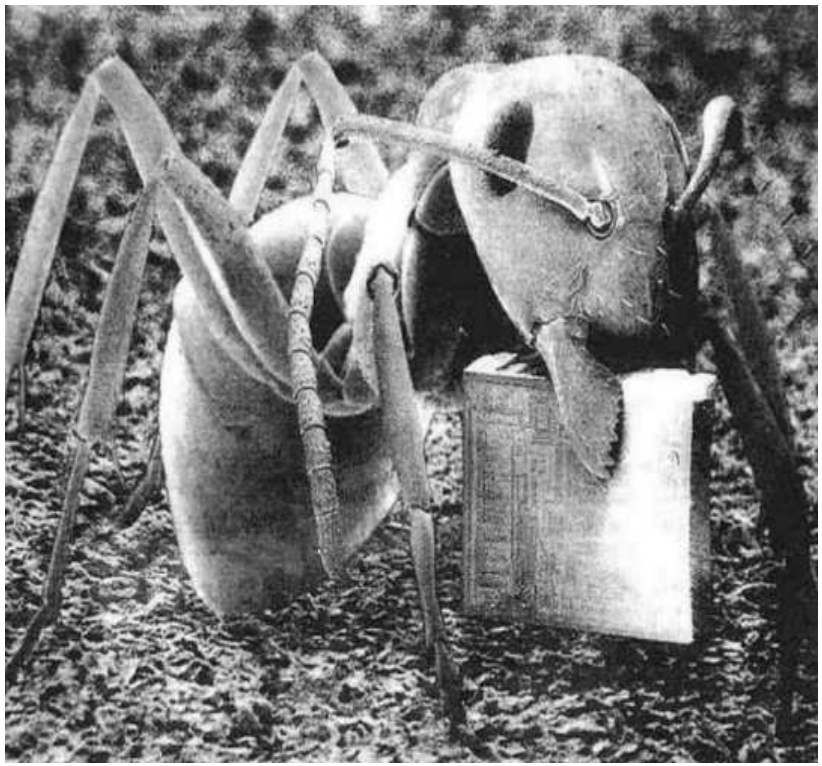
Lessons Learned

- Accidents are seldom simple : usually involve a complex web of interacting events with multiple contributing technical, human, and organizational factors
- Fixing a particular error never prevents further accidents
- Software quality assurance standards should be established
- Designs should be simple

Reality

- A significant amount of software for life-critical systems come from small firms that fit the profile of those resistant to principles of software engineering

Software Engineering Gone Well



- Case study: Raytheon

Case for Improvement

- Software is a major element of products developed by Raytheon
- Functionality of complex systems moving more towards software
- Latter part of decade, software problems translated into contract performance issues
- Lack of success in delivering projects on schedule and within budget
- Customers using SEI process maturity framework as selection criteria

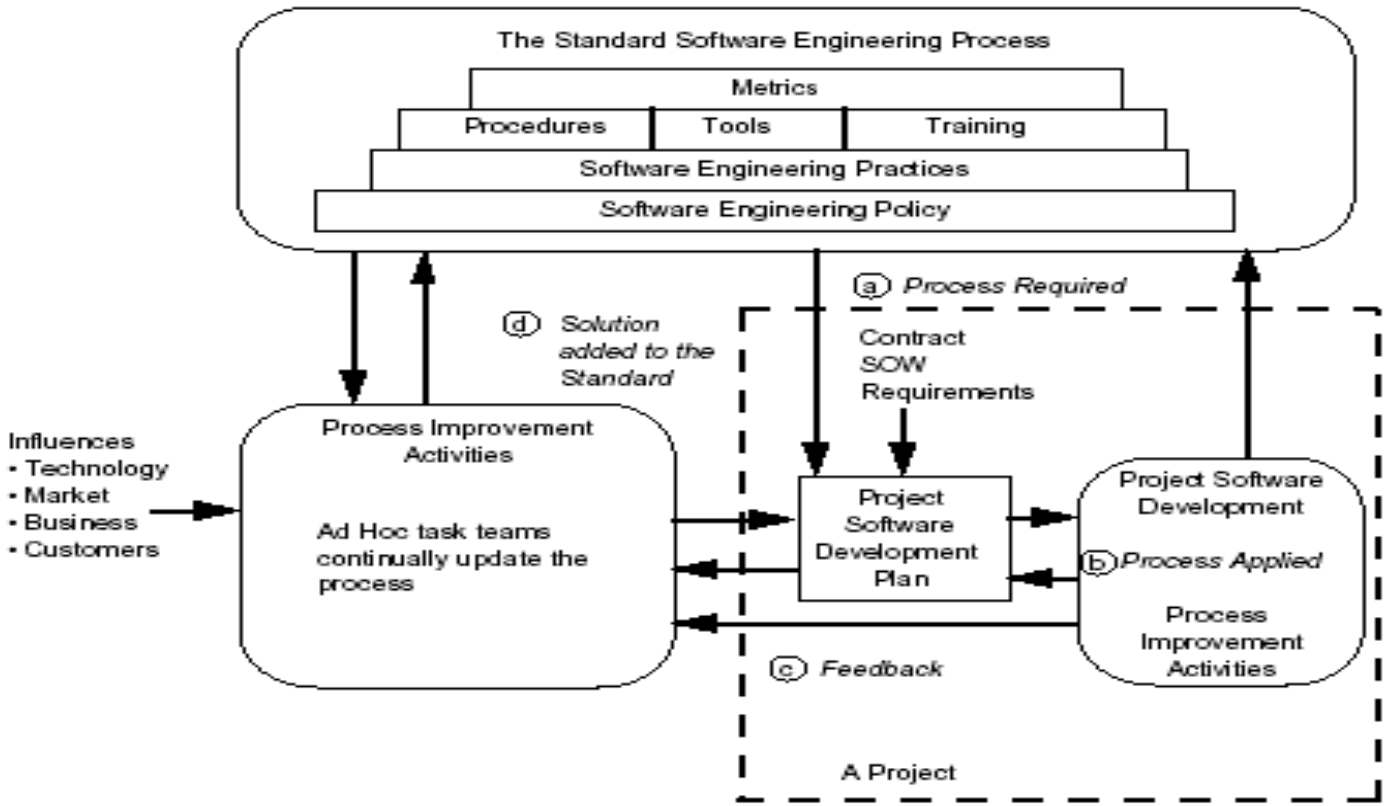
Process Improvement

- Three critical activities:
  - establish a strong and effective infrastructure and maintain enthusiasm over time
  - identify risks and develop mitigation strategy
  - measure and analyze project data to determine benefits

Measurement of Improvement

- Measure return on investment (ROI) through goal of reducing the amount of rework
- Analysis of software productivity on projects
- Used (cost at completion/budget) to measure predictability
- Defect density analysis for measuring software quality

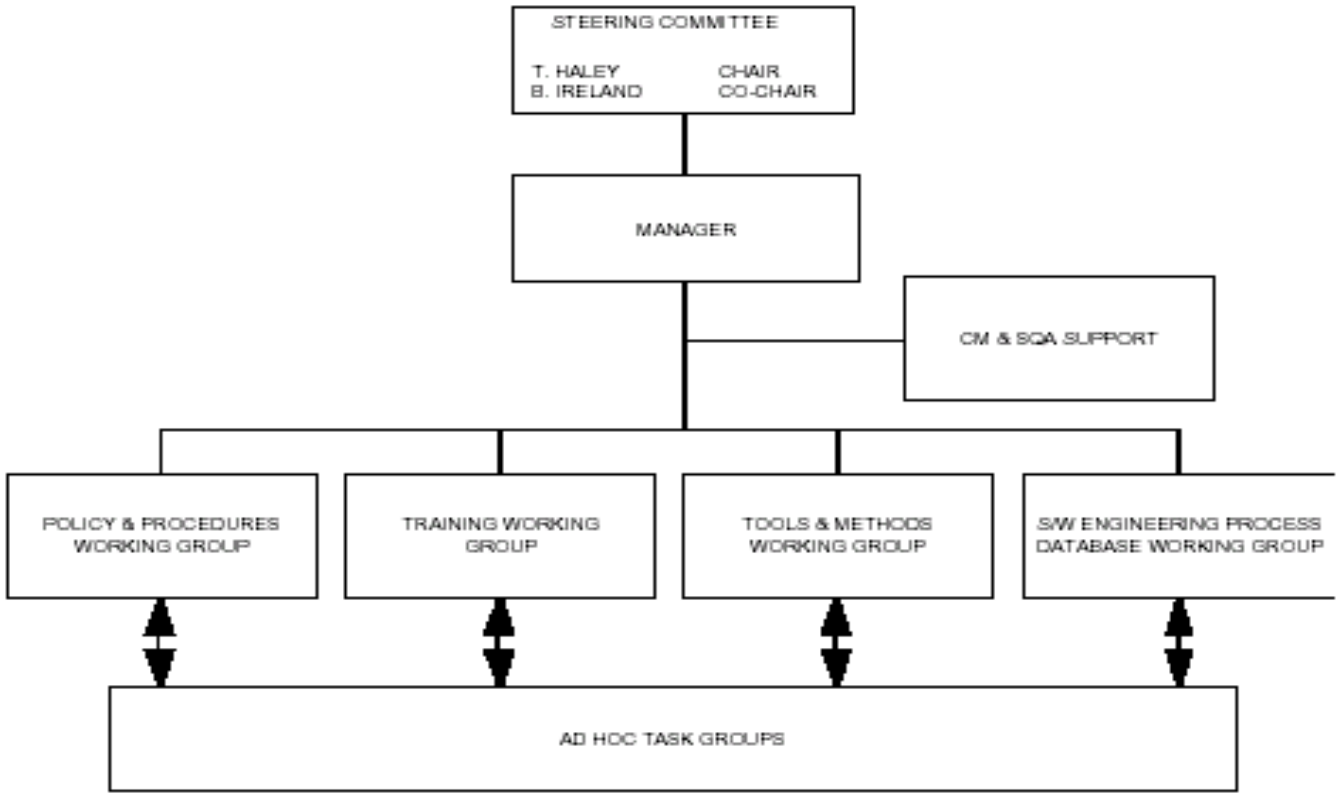
Process Improvement Model



Process Improvement Model

- a) selected project uses organization process to develop software development plan
- b) implementation occurs process is applied
- c) feedback for continual refinement of process

Infrastructure



Policy and Procedures Group

- Responsible for developing documents describing process
- Based on “best-of-best” practices
- Existing standards like Capability Maturity Model

Training Group

- Responsible for developing the comprehensive training program
- Take feedback and examine effectiveness of training

Tools and Methods Group

- Responsible for implementing CASE tools available to software development and management team
- CASE tools aid in capturing requirements, design data and publish resulting documents

Process Database Group

- Responsible for Process Data Center
- Repository for project data and metrics used in root cause analysis
- Gives feedback through improvement recommendation to both projects and overall process
- Previous projects can give pertinent information to project manager

Process Binding

- Obstacles to the success of program:
  - Sponsorship and Commitment
  - Recruiting right personnel
  - Resources
  - Cultural issues

Measurement and Analysis

- Data collected monthly
- Data measurement:
  - Progress metrics

    data used to gauge progress or current status of project
  - System technical performance metrics

    describes the current measured or estimated utilization of software system resources

Impact - Cost of Quality

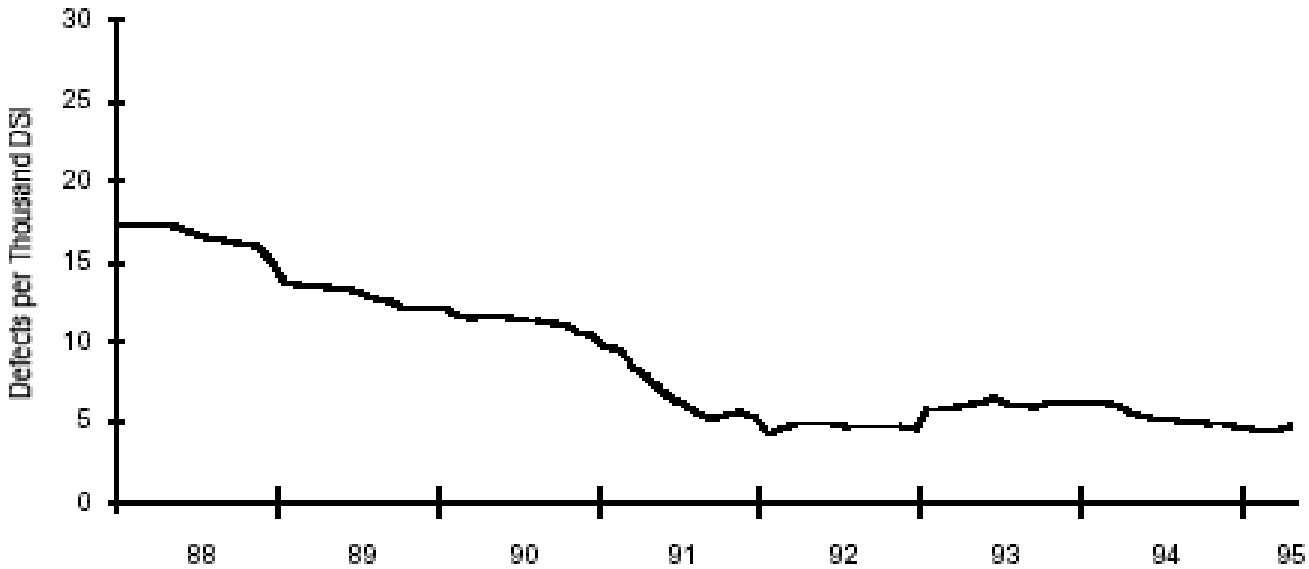
- Average cost of rework had decreased
- Two years prior rework averaged 41% of project costs
- Reduced to 20%
- Largest reduction from fixing problems during source code integration
- Second largest was cost of retesting

Impact - Software Productivity

- Data measured by delivered source instructions per person-month of development effort
- Using staff size as weighting function, results showed average productivity increasing

Impact - Overall Product Quality

- Assessed through defect density in final software product



Results

- FAA Terminal Doppler Weather Radar Program
- Using CMM Level 3 process delivered 6 months ahead of time
- Resulted in predictable development, increased productivity and quality
- Increased business

Summary

- Software Engineering is less the code but more on the process involved
- focusing on standardized software engineering processes results in:
  - increased reliability
  - less debugging
  - improved quality control
  - less dead people!