The Economics of Software Reliability

Barry Boehm, USC
ISSRE 2003 Keynote Address
November 19, 2003
(boehm@sunet.usc.edu)
(http://sunet.usc.edu)

Outline

• The business case for software reliability
  – Example: net value of test aids
  – Value depends on stakeholder value propositions
• What are stakeholders really relying on?
  – Safety, accuracy, response time, …
  – Attributes often conflict
• Software dependability in a competitive world
• Conclusions
Software Reliability Business Case

• Software reliability investments compete for resources
  – With investments in functionality, response time, adaptability, speed of development, ...

• Each investment option generates curves of net value and ROI
  – Net Value NV = PV(benefit flows-cost flows)
  – Present Value PV = Flows at future times discounted
  – Return on Investment ROI = NV/PV(cost flows)

• Value of benefits varies by stakeholder and role

Software Testing Business Case

• Vendor proposition
  – Our test data generator will cut your test costs in half
  – We’ll provide it to you for 30% of your test costs
  – After you run all your tests for 50% of your original costs, you’re 20% ahead

• Any concerns with vendor proposition?
Software Testing Business Case

- Vendor proposition
  - Our test data generator will cut your test costs in half
  - We’ll provide it to you for 30% of your test costs
  - After you run all your tests for 50% of your original costs, you’re 20% ahead

- Any concerns with vendor proposition?
  - Test data generator is value-neutral*
  - Every test case, defect is equally important
  - Usually, 20% of test cases cover 80% of business value

* As are most current software engineering techniques

20% of Features Provide 80% of Value: Focus Testing on These (Bullock, 2000)
Value-Based Testing Provides More Net Value

There is No Universal Dependability-Value Metric

- Different stakeholders rely on different value attributes
  - Protection: safety, security, privacy
  - Robustness: reliability, availability, survivability
  - Quality of Service: performance, accuracy, ease of use
  - Adaptability: evolvability, interoperability
  - Affordability: cost, schedule, reusability
- Value attributes continue to tier down
  - Performance: response time, resource consumption (CPU, memory, comm.)
- Value attributes are scenario-dependent
  - 5 seconds normal response time; 2 seconds in crisis
- Value attributes often conflict
  - Most often with performance and affordability
Major Information System Dependability Stakeholders

- Dependents
  - passengers, patients

- Information Suppliers
  - citizens, companies

- Information Brokers
  - financial services, news media

- Information Consumers
  - decisions, education, entertainment

- Mission Controllers
  - pilots, distribution controllers

Developers, Acquirers, Administrators

Overview of Stakeholder/Value Dependencies

- Strength of direct dependency on value attribute
  ** - Critical; * - Significant; blank - insignificant or indirect

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Protection</th>
<th>Robustness</th>
<th>Quality of Service</th>
<th>Adaptability</th>
<th>Affordability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info. Suppliers, Dependents</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Info. Brokers</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>Info. Consumers</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mission Controllers, Administrators</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td></td>
<td>**</td>
</tr>
<tr>
<td>Developers, Acquirers</td>
<td>*</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

11/19/03 ©USC-CSE
Elaboration of Stakeholder/Value Dependencies

Implications for Dependability Engineering

- There is no universal dependability metric to optimize
- Need to identify system’s success-critical stakeholders
  - And their dependability priorities
- Need to balance satisfaction of stakeholder dependencies
  - Stakeholder win-win negotiation
  - Dependability attribute tradeoff analysis
- Need value-of-dependability models, methods, and tools
Outline

• The business case for software reliability
• What are stakeholders really relying on?
• Software dependability in a competitive world
  – Is market success a monotone function of dependability?
  – Is quality really free?
  – Is “faster, better, cheaper” really achievable?
  – Value-based vs. value-neutral methods
• Conclusions

Competing on Cost and Quality
- adapted from Michael Porter, Harvard

ROI

Too cheap
(acceptable Q)
Stuck in the middle
Quality leader
(acceptable C)
Cost leader
(acceptable Q)
Over-priced
Price or Cost Point
Competing on Schedule and Quality
- A risk analysis approach

• Risk Exposure \( RE = \text{Prob (Loss)} \times \text{Size (Loss)} \)
  
  – “Loss” – financial; reputation; future prospects, …

• For multiple sources of loss:
  \[ RE = \sum_{\text{sources}} [\text{Prob (Loss)} \times \text{Size (Loss)}] \]

Example RE Profile: Time to Ship
- Loss due to unacceptable dependability

\[ RE = P(L) \times S(L) \]

- Many defects: high \( P(L) \)
- Critical defects: high \( S(L) \)
- Few defects: low \( P(L) \)
- Minor defects: low \( S(L) \)

Time to Ship (amount of testing)
Example RE Profile: Time to Ship
- Loss due to unacceptable dependability
  - Loss due to market share erosion

\[ RE = P(L) \times S(L) \]

- Few rivals: low P(L)
- Weak rivals: low S(L)
- Many defects: high P(L)
- Critical defects: high S(L)
- Many rivals: high P(L)
- Strong rivals: high S(L)
- Few defects: low P(L)
- Minor defects: low S(L)

Time to Ship (amount of testing)

Example RE Profile: Time to Ship
- Sum of Risk Exposures

\[ RE = P(L) \times S(L) \]

- Few rivals: low P(L)
- Weak rivals: low S(L)
- Many defects: high P(L)
- Critical defects: high S(L)
- Many rivals: high P(L)
- Strong rivals: high S(L)
- Few defects: low P(L)
- Minor defects: low S(L)

Time to Ship (amount of testing)
Comparative RE Profile:
Safety-Critical System

\[ RE = P(L) \times S(L) \]

Time to Ship (amount of testing)

Comparative RE Profile:
Internet Startup

\[ RE = P(L) \times S(L) \]

Time to Ship (amount of testing)
Interim Conclusions

- Unwise to try to compete on both cost/schedule and quality
  - Some exceptions: major technology or marketplace edge
- There are no one-size-fits-all cost/schedule/quality strategies
- Risk analysis helps determine how much testing (prototyping, formal verification, etc.) is enough
  - Buying information to reduce risk
- Often difficult to determine parameter values
  - Some COCOMO II values discussed next
“Quality is Free”

- Did Philip Crosby’s book get it all wrong?
- Investments in dependable systems
  - Cost extra for simple, short-life systems
  - Pay off for high-value, long-life systems
Software Life-Cycle Cost vs. Dependability

Relative Cost to Develop, Maintain

COCOMO II RELY Rating

Very Low  Low  Nominal  High  Very High

Very Low

0.8  0.9  1.0  1.1  1.2  1.3  1.4

Very Low

0.02  0.02  0.02  0.02  0.02

Very Low

1.10  1.10  1.10  1.10  1.10

Very Low

1.23  1.26

11/19/03  ©USC-CSE
Outline

• The business case for software reliability
• What are stakeholders really relying on?
• Software dependability in a competitive world
  – Is market success a monotone function of dependability?
  – Is quality really free?
  → – Is “faster, better, cheaper” really achievable?
    – Value-based vs. value-neutral methods
• Conclusions
Cost, Schedule, Quality: Pick any Two?

- Consider C, S, Q as Independent Variable
  - Feature Set as Dependent Variable
C, S, Q as Independent Variable

- Determine Desired Delivered Defect Density (D4)
  - Or a value-based equivalent
- Prioritize desired features
  - Via QFD, IPT, stakeholder win-win
- Determine Core Capability
  - 90% confidence of D4 within cost and schedule
  - Balance parametric models and expert judgment
- Architect for ease of adding next-priority features
  - Hide sources of change within modules (Parnas)
- Develop core capability to D4 quality level
  - Usually in less than available cost and schedule
- Add next priority features as resources permit
- Versions used successfully on 32 of 34 USC digital library projects

Value-Based vs. Value Neutral Methods

- Value-based defect reduction
- Constructive Quality Model (COQUALMO)
- Information Dependability Attribute Value Estimation (iDAVE) model
Value-Based Defect Reduction
Example:
Goal-Question-Metric (GQM) Approach

Goal: Our supply chain software packages have too many defects. We need to get their defect rates down

Question: ?

---

Value-Based GQM Approach – I

Q: How do software defects affect system value goals?
   ask why initiative is needed
   - Order processing
   - Too much downtime on operations critical path
   - Too many defects in operational plans
   - Too many new-release operational problems

G: New system-level goal: Decrease software-defect-related losses in operational effectiveness
   - With high-leverage problem areas above as specific subgoals

New Q: ?
Value-Based GQM Approach – II

New Q: Perform system problem-area root cause analysis:
ask why problems are happening via models
Example: Downtime on critical path

- Where are primary software-defect-related delays?
- Where are biggest improvement-leverage areas?
  - Reducing software defects in Scheduling module
  - Reducing non-software order-validation delays
  - Taking Status Reporting off the critical path
  - Downstream, getting a new Web-based order entry system
- Ask “why not?” as well as “why?”

Value-Based GQM Results

- Defect tracking weighted by system-value priority
  - Focuses defect removal on highest-value effort
- Significantly higher effect on bottom-line business value
  - And on customer satisfaction levels
- Engages software engineers in system issues
  - Fits increasing system-criticality of software
- Strategies often helped by quantitative models
  - COQUALMO, iDAVE
Current COQUALMO System

COQUALMO

Defect Introduction Model

Defect Removal Model

Software development effort, cost and schedule estimate

Number of residual defects
Defect density per unit of size

Software Size Estimate

Software platform, Project, product and personnel attributes

Defect removal profile levels

Automation, Reviews, Testing

Defect Removal Rating Scales

<table>
<thead>
<tr>
<th></th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automated Analysis</td>
<td>Simple compiler syntax checking</td>
<td>Basic compiler capabilities</td>
<td>Compiler extension Basic req. and design consistency</td>
<td>Intermediate-level module Simple req. design</td>
<td>More elaborate req./design Basic dis-processing</td>
<td>Formalized specification, verification, Advanced dis-processing</td>
</tr>
<tr>
<td>Peer Reviews</td>
<td>No peer review</td>
<td>Ad-hoc informal walk-through</td>
<td>Well-defined preparation, review, minimal follow-up</td>
<td>Formal review roles and Well-trained people and basic checklist</td>
<td>Root cause analysis, formal follow Using historical data</td>
<td>Extensive review checklist Statistical control</td>
</tr>
<tr>
<td>Execution Testing and Tools</td>
<td>No testing</td>
<td>Ad-hoc test and debug</td>
<td>Basic test criteria based on checklist</td>
<td>Well-defined test req. and basic test coverage tool system</td>
<td>More advance test tools, preparation Dr ease monitoring</td>
<td>Highly advanced tools, model-based test</td>
</tr>
</tbody>
</table>

COCOMO II p.263
Defect Removal Estimates
- Nominal Defect Introduction Rates

![Graph showing delivered defects per KSLOC against composite defect removal rating.]

iDAVE Model

iDAVE

- Cost estimating relationships (CER's): Cost = f(IP capabilities (size), project attributes)
- Dependability attribute estimating relationships (DAR's): D = g(dependability investments, project attributes)
- Value estimating relationships (VER's): V = h(IP capabilities, dependability levels D)

Time-phased
- Cost
- Dependability attribute levels D
- Value components V
- Return on investment
**ROI Analysis Results Comparison**

- Business Application vs. Space Application

![Chart](chart.png)

**Conclusions: Software Reliability (SWR)**

- There is no universal SWR metric to optimize
  - Need to balance stakeholder SWR value propositions
- Increasing need for value-based approaches to SWR
  - Methods and models emerging to address needs
- “Faster, better, cheaper” is feasible
  - If feature content can be a dependent variable
- “Quality is free” for stable, high-value, long-life systems
  - But not for dynamic, lower-value, short life systems
- Future trends intensify SWR needs and challenges
  - Criticality, complexity, decreased control, faster change