A MARKET-BASED APPROACH TO SOFTWARE EVOLUTION

DAVID F. BACON *
YILING CHEN
DAVID PARKES
MALVIKA RAO

HARVARD UNIVERSITY

* IBM RESEARCH
Bugs are Everywhere
annoying, costly, dangerous

“Software Crisis” (F. L. Bauer)
First NATO Software Engineering Conference, 1968
A Tradition of Failure

Formal Methods
- Specs & Proofs
- Model Checking
- Fatal Flaws:
  - Rely on Spec
  - Don’t Scale

Software Engineering
- Methodology
- Process
- Fatal Flaws:
  - Not Quantitative
  - Degenerates to Religion
Bugs Have a “Long Tail”

How do bugs get sorted??

How are costs determined??

Bugs sorted by Value

These get fixed… maybe

security bugs

These don’t
Users and Developers Are Isolated From Each Other

...deliberately
because feedback can’t be accumulated automatically
Can a Market Help Solve This Problem?

- Large supply of work
- Large supply of capable workers
- Real value for performing the work
The application Safari quit unexpectedly.
Mac OS X and other applications are not affected.

Click Reopen to open the application again. Click Report to see details or send a report. Click Offer Bounty to contribute to a bounty for fixing this bug.
Select an amount to offer as a bounty for fixing this bug.

Your bounty will be held in escrow until the bug is fixed or the time limit expires. The default time limit is 6 months.

Currently, 875 users have offered a total of $2298.45 for fixing this bug. You have been affected by this bug 7 times.

$0.99  Avg. $2.63  Max: $50  Other
Correctness Demand

- Sum of rewards for a bug is the demand to fix it
- Sum of all rewards is the correctness demand

- When correctness demand = 0 either
  - software is bug free or...
  - no one cares about it anymore.
Correctness Potential

- Set of possible workers
- For each bug, each worker has a cost to fix it
- If cost < reward, “worth fixing” for that worker

- Potential of bug: profit by most efficient worker
- Correctness Potential = the sum of bug potentials
Market is in correctness equilibrium when correctness potential = 0

In “living” software that never happens:
- new bugs are found
- bug bids change
- workers come and go

Goal: design a system that tends towards dynamic equilibrium
Is it a Bug or a Feature?
Who Cares?!
How do we Design such a Market?

- GUIDING PRINCIPLES:
  - Autonomy: all actions are market-driven
  - Inclusiveness: all contributors are rewarded
  - Transparency: “financial disclosure”
  - Reliability: robustness to manipulation

- Apply both market pressure and software tools
What are the Components?

- Funding
- Workflow Process
- Reputation System
Show me the Money!

Cash or scrip or votes?

Sources of real cash:
- direct user bids
- escrow from sale (closed source)
- escrow from contribution (shareware)
- escrow from registration (open source)

Time limit on bids - money reverts to source
WORKFLOW: Bug

- Report
- Bid
- Categorize
- Reproduce
- Fix
- Test
- Commit
- Distribute

Everyone Shares Reward

Humans vs Tools?
Reputation System

- Ratings based on past performance
- Control certain activities (e.g. commits)
  - May also affect reward distribution
- Adjusted with information about software lifetime
- Can be seeded by central organization
  - Useful when project is small
- Occasional escape hatch
It's Started: App Store

**Player X**

**Pang Mobile**
Category: Games
Released: Oct 12, 2009
Seller: Player X
© 2009 Player X
Version: 1.0.0 (iPhone OS 3.0 Tested)
7.0 MB

$2.99  BUY APP

**RATED 4+**

**APPLICATION DESCRIPTION**

“Check it out for some throwback arcade goodness” – GEARDIARY.COM

A bubble blast from the past, Pang is the game you love and remember from the 80's, even better today on the iPhone and iPod touch.

Test your reflexes in this hugely addictive official remake, "Pang", also known as "Buster Bros" in the US. There's a reason this game was huge – it's great!

Travel from Japan to the tropical shores of Hawaii as you take up the task of saving civilization from dangerously bouncing bubbles that threaten city landmarks across the planet.

Can you save Mount Fuji from being flattened by spherical dangers? Will your reactions be good enough to defend the Eiffel Tower or the Statue of Liberty? Or can you wear them down while saving your own skin?

Everyone's favourite from the arcades, Pang is easy to learn but difficult to master as you progress through 17 locations saving the world's attractions with an assortment of weapons from the grappling hook to the twin harpoon. The game is crammed with power-ups including a force field and time freeze; anything to give you the edge over those deadly bubbles!

**FEATURES**: 17 locations, covering 50 levels
An assortment of weaponry
Power-ups and prizes for points
World Tour or infinite Challenge to choose from
Leaderboards – beat your friends for the highest score
Customisable options suit your style

Download Pang NOW! Relive the great times of retro gaming.

Perfect for fans of Space Invaders, PacMan and Q*bert

Works only on iPhone / iPod touch 3.0 OS and above.

Upcoming updates: 1.0.1 to 2.2.1 OS and above support added.
1.0.2 – OpenFeint online leaderboards supported.

Join our Facebook page here: [http://Tinyurl.com/PANG-Mob-F8](http://Tinyurl.com/PANG-Mob-F8)

http://www.geardiary.com/2009/10/19/pang-for-iphonetouch-review/

**LANGUAGES:**

English
### Orange_Cloud

- **Algorithm Rating:** 2282
- **High School Rating:** not rated
- **Conceptualization Rating:** not rated
- **Specification Rating:** not rated
- **Architecture Rating:** not rated
- **Design Rating:** 776
- **Development Rating:** 1715
- **Assembly Rating:** 1216
- **Application Testing Rating:** not rated
- **Marathon Matches Rating:** 1204

- **Total Earnings:** $35,421.80
- **Member Since:** 07/24/2006
- **Country:** Russian Federation

#### Development Competitions

**Rating:** 1715

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Rank</th>
<th>Country Rank</th>
<th>Volatility</th>
<th>Competitions</th>
<th>Reliability</th>
<th>Maximum Rating</th>
<th>Minimum Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>87.5</td>
<td>12 of 96</td>
<td>1 of 2</td>
<td>53</td>
<td>66.67%</td>
<td>1864</td>
<td>1050</td>
<td></td>
</tr>
</tbody>
</table>

#### Submission Details

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>.NET</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inquiries</td>
<td>58</td>
<td>0</td>
<td>58</td>
</tr>
<tr>
<td>Submissions</td>
<td>33</td>
<td>n/a</td>
<td>33</td>
</tr>
<tr>
<td>Submission Rate</td>
<td>56.90%</td>
<td>n/a</td>
<td>56.90%</td>
</tr>
<tr>
<td>Passed Screening</td>
<td>33</td>
<td>n/a</td>
<td>33</td>
</tr>
<tr>
<td>Screening Success Rate</td>
<td>100.00%</td>
<td>n/a</td>
<td>100.00%</td>
</tr>
<tr>
<td>Passed Review</td>
<td>33</td>
<td>n/a</td>
<td>33</td>
</tr>
<tr>
<td>Review Success Rate</td>
<td>100.00%</td>
<td>n/a</td>
<td>100.00%</td>
</tr>
<tr>
<td>Appeals*</td>
<td>241</td>
<td>n/a</td>
<td>241</td>
</tr>
<tr>
<td>Appeal Success Rate*</td>
<td>47.72%</td>
<td>n/a</td>
<td>47.72%</td>
</tr>
<tr>
<td>Maximum Score</td>
<td>99.43</td>
<td>n/a</td>
<td>99.43</td>
</tr>
<tr>
<td>Minimum Score</td>
<td>78.46</td>
<td>n/a</td>
<td>78.46</td>
</tr>
<tr>
<td>Average Score</td>
<td>94.06</td>
<td>n/a</td>
<td>94.06</td>
</tr>
<tr>
<td>Average Placement</td>
<td>1.85</td>
<td>n/a</td>
<td>1.85</td>
</tr>
<tr>
<td>Wins</td>
<td>19</td>
<td>n/a</td>
<td>19</td>
</tr>
<tr>
<td>Win Percentage</td>
<td>57.59%</td>
<td>n/a</td>
<td>57.59%</td>
</tr>
</tbody>
</table>

* only includes appeals from projects posted on or after March 16, 2006 (TopCoder did not previously collect the relevant data)
Market-Based Software

- Only Possible Kind of Solution
- Empowers Users and Programmers
- Makes Problem Quantitative
Thanks.

Feedback?
Mechanism Design Problems

- Avoiding Freeloading
- Preventing Fraudulent “Fixed” Claims by Providers
- Preventing Fraudulent “Not Fixed” Claims by Consumers
- Lag in fix verification by Consumers
Lots of Uncertainty

- When are two crashes the “same bug”? Line number? Data set?
- When does a change “fix” a bug?
  - Partial fixes & incorrect fixes are not uncommon
  - One fix may improve or worsen another bug
- If multiple fixes submitted, which is best?
  - Band-aids versus Deep fixes
- Program analysis can help reduce uncertainty, but will never eliminate it
Next Steps

- Simplified market mechanism design with analytical equilibrium property
- Identify analysis and testing techniques that can be integrated into the system.
- Prototype market infrastructure
- Trial run (seed a market?)
**TopCoder**

- Handles “supply side” -- developers
- Highly differentiated stages of development
- Short, manageable tasks
- Competitive process
- Validation:
  - automated testing
  - competitive forces: challenges
iTunes App Store

- Micropayment system with broad acceptance
- Primarily supply side
  - but often compete for users on similar apps
- Monolithic -- but apps are fine-grained
- Developers responsive to user feedback
- Software Distribution Mechanism
Bug Auctions for Vulnerability Markets

- **Producer**
- **Testers**
- **Attackers**
- **Users**

*Purchase Price*
Bug Auctions for Vulnerability Markets
(Ozment’s redefinition of Schechter)

- Note: paying for bug reports ("user" activity)
- Bounty $R$ starts at $R_0$ increasing by $\partial$/day
- Open first-price ascending (reverse Dutch) auction
  - Open auction speeds discovery
- Non-security bugs receive $fR$, where $f << 1$
- $R$ acts as a "measure of security"
Bug Auctions for Vulnerability Markets (Ozment’s Enhancements)

- **Producer**
- **Trusted Third Party**
- **Testers**
- **Attackers**
- **Users**

Mathematical formula: $E = rt + \alpha R_0$

Purchase Price
Bug Auctions for Vulnerability Markets (Ozment’s Enhancements)

- Set initial reward (first $R$) high
- Include reputation reward
- Commit/escrow minimum payout $E = rt + \nu R_0$
- Reduce $R$ to $Rx$ ($x < 1$) if exploit precedes fix
- Don’t expose number of testers (unless small)
- Give reward for registered testers
- Use trusted third party to escrow reward fund
“Federal Funding” (Kannan & Telang)

Federal Government

“Infomediary” (CERT)

Testers

Attackers

Users

$p_b$

leak

$p_p$
A Comprehensive Market for Software Evolution
Specifications and Proofs of Correctness
- Limited to ~1000 line programs

Model Checking
- Limited to problems with small state spaces

Big, real-world programs often have no precise “spec”
- ...or it’s too complex to verify or test exhaustively

Dijkstra Turing Award prediction failed to happen
But Why Differentiate?

BUG

FEATURE

$
Aside: Mechanism Design

- What information is revealed has a big impact
### Top 3 Fatal Bugs
**Mozilla Firefox**

<table>
<thead>
<tr>
<th>COMPONENT</th>
<th>DESCRIPTION</th>
<th>BOUNTY HUNTERS</th>
<th>USERS</th>
<th>PER-USER BOUNTY</th>
<th>TOTAL BOUNTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Widget: Cocoa</td>
<td>firefox hangs if cookie ask permission to set whilst save target as dialog is open (image)</td>
<td>3</td>
<td>1521</td>
<td>$2.27</td>
<td>$3457.98</td>
</tr>
<tr>
<td>Places</td>
<td>Live bookmarks load way too aggressively (lock up/hang/freeze browser)</td>
<td>1</td>
<td>162</td>
<td>$9.12</td>
<td>$1477.44</td>
</tr>
<tr>
<td>XUL</td>
<td>UI freezes if alert/dialog comes up while dragging (Modal dialog during drag causes hang)</td>
<td>0</td>
<td>3818</td>
<td>$0.34</td>
<td>$1298.12</td>
</tr>
</tbody>
</table>
Since Specs Are Fallible...

- Forget formal specification

- The spec is what the market says it ought to be
And While We’re At It…
Broaden the Market

- Documentation
- “Help Desk” Support (0-line aka RTFM fixes)
- Installation
Empowering the Tail: Consumer Bug Bounties

- Security bugs
- Reputation cost to producer = repair cost to producer
- Bug value to consumers = repair cost to programmer

Select an amount to offer as a bounty for fixing this bug.

Your bounty will be held in escrow until the bug is fixed or the time limit expires. The default time limit is 6 months.

Currently, 875 users have offered a total of $2298.45 for fixing this bug. You have been affected by this bug 7 times.

Select: $0.99, Avg. $2.63, Max. $50, Other

Bugs sorted by value by Consumers
SOFTWARE IMPROVEMENT
Complex Structure
Problem only with Uncertainty(?)

Multiple Aggregated “Consumers”

Multiple Competing “Providers”
Social Utility Issues

Open source: avoid “crowding out” altruistic providers

Closed source: drive collaboration and profit-sharing

- Would companies allow their programmers to collect bounties?
Generalized Application

- Security bugs
- Functional bugs
- Non-fatal bugs
- Feature requests

How are these reported and aggregated??
Assume Away Uncertainty?

- Design market assuming we can
  - precisely classify bugs
  - precisely identify fixes
Attack Uncertainty Separately

- Program analysis
- Program slicing
- Statistical clustering techniques

User Observation

- Change in bug frequency
- Rating of Producers (for fixes) and Consumers (for acceptance tests)
"App Store" Model

* $N$ Consumers, but only 1 Producer