Law of Minor Release
More Bugs $\implies$ Better Software Quality

Audris Mockus

Avaya Labs Research
211 Mt Airy Rd
Basking Ridge, NJ 07920
audris@avaya.com

Aug 20, 2013
Motivation

Are there fundamental time-lag relationships among software production factors (Laws of Software Evolution)?

Can they be harnessed to improve software development?
Studies of Software Evolution

Focus on long-term trends in, e.g., software size

- Such trends are caused by non-software factors
  - World economy
  - Business practices
  - Technology change

- Not clear how to use in practice
Studies of Software Evolution

Focus on long-term trends in, e.g., software size
  ▶ Such trends are caused by non-software factors
    ▶ World economy
    ▶ Business practices
    ▶ Technology change
  ▶ Not clear how to use in practice

Investigating SW evolution by observing only software

≡

divining reality from shadows on a cave wall
Proposed Solution

- Investigate short-term and repeating relationships with a clear mechanism originating from the way software is created and used
- Use information from outside software development cave
- Answer practical questions
  - Can we compare quality among releases to evaluate the effectiveness of QA practices?
  - Can quality be approximated with easy-to-obtain measures, e.g., defect density?
Approach

- Start from clear assumptions
- Observe fundamental relationships
- Validate
- Build more complex propositions using validated relationships

Define: *Bug*
A user-observed (and reported) program behavior (e.g., failure) that results in a code change.

Define: *Action Will Introduce a Bug*
Action will increase the chances of a Bug occurring in the future.
First Fundamental Law of Software Evolution

Formulation
Code change will introduce bugs

Mechanism

- New code has defects
- New code exercises existing code differently
- Program behavior changes

Note: platform changes cause code changes

Evidence

- New releases bring new bugs
- Model: a business-driven feature change

\[ N \sim \text{Poisson}(\lambda) \text{ fixes with delay } T \sim \text{Exp}(\mu) \] [4]
Model prediction for one release

Weekly MRs (Person Weeks)

Calendar Weeks

New feature MRs
Actual Repair MRs
Predicted Repair MRs
Model prediction for 11 releases (using earlier release)

![Graph showing weekly MRs and predicted repair MRs for 11 releases over calendar weeks from 1994 to 2002.]
Corollary 1: Need to Normalize by Change to Obtain Quality

How to normalize by change?

- divide by the number of pre-release MRs
- divide by the LOC added or changed

Hypothesis 1

Increase \( \uparrow \) in the number of customer-found defects per pre-release MR (a simple-to-obtain measure) affects users’ perception of software quality negatively \( \downarrow \)
Qualitative evidence: No

How to compare software releases?

“we tried to improve quality: get most experienced team members to test, code inspections, root cause analysis, ...”

“Did it work? I.e., is this release better than previous one?”

Everyone uses defect density (e.g., customer reported defects per 1000 changes or lines of code), but “it does not reflect the feedback from customers.”
Let’s Peek Outside the Software Development Cave
Does the increase in

the number of users and

the amount of usage

introduce bugs?
Second Fundamental Law of Software Evolution

Formulation
Deploying to more users will introduce bugs

Mechanism
- New use profiles
- Different environments

Evidence

Release with no users has no bugs
Third Fundamental Law of Software Evolution

Formulation
Longer (and heavier) use will introduce bugs

Mechanism

- New inputs and use cases are encountered over longer periods
- More extreme environmental conditions happen over longer periods

Evidence

- Bugs tend to be encountered even after year(s) of usage
- See Commandments below
Third Fundamental Law of Software Evolution

Formulation
Longer (and heavier) use will introduce bugs

Mechanism

- New inputs and use cases are encountered over longer periods
- More extreme environmental conditions happen over longer periods

Evidence

- Bugs tend to be encountered even after year(s) of usage
- See Commandments below

Does every user and every year of usage introduce the same number of bugs?
Commandment 1: Don’t Install Right After the Release Date

Formulation
Users who install close to the release date will introduce more bugs

Mechanism
- Later users get builds with patches
- Services team understands better how to install/configure properly
- Workarounds for many issues are discovered

Evidence

![Graph showing the fraction of customers observing SW issue vs. time between launch and deployment]

- Quality ↑ with time after the launch, and is an order of magnitude better one year later [5]
Commandment 2: Don’t Panic After Install/Upgrade

Formulation
A user will introduce more bugs close to their install/upgrade date

Mechanism
- Software is not hardware: parts do not wear off
- Misconfiguration or incompatibility with the environment

Evidence

- Two thirds of customer issues (leading to a software fix) are reported within three months of install
- Sample: 87 release/product combinations
Corollary 1: Customer Quality

Formulation
Software release quality from users perspective is the fraction of:

- The number of customers who report a bug shortly after the installation over
- The number of customers who install soon after the release date (e.g., within seven months)
Corollary 1: Customer Quality

Formulation
Software release quality from users perspective is the fraction of:

- The number of customers who report a bug shortly after the installation over
- The number of customers who install soon after the release date (e.g., within seven months)

“We live or die by this measure”

VP for quality
Testing Hypothesis 1: Defect Density Reflects Customer Quality

![Graph showing defect density per pre-GA MR](image)

- **M** Customer Defects Per Pre-GA MR

- **Axes:**
  - **Y-axis:** Quantity
  - **X-axis:** r1.1, r1.2, r1.3, r2.0, r2.1, r2.2
Testing Hypothesis 1: Defect Density Reflects Customer Quality

![Graph showing defect density over time]

- **Customer Defects Per Pre-GA MR**
- **% of customers with defect within 3m. of install**

The graph illustrates the relationship between the quantity of defects and the percentage of customers with defects within 3 months of installation. The data points are labeled with the years r1.1, r1.2, r1.3, r2.0, r2.1, and r2.2.
Testing Hypothesis 1: Defect Density Reflects Customer Quality

![Graph showing defect density and percentage of customers with defects within 3 months of installation.](image-url)
Testing Hypothesis 1: Defect Density Reflects Customer Quality

![Graph showing defect density and customer defect rates over time.]

- **Better**: Customer defect rates drop significantly in the first release (r1.1) and continue to improve in r1.2, suggesting improved quality.
- **Worse**: In r1.3, there is a noticeable increase in defect rates, indicating potential issues with the release.
- **Customer Defects Per Pre-GA MR**: The graph shows the cumulative defect rates over time.
- **% of customers with defect within 3m. of install**: The red dashed line indicates the percentage of customers experiencing defects within three months of installation, highlighting the impact of defects on customer satisfaction.

---

**Legend**:
- **Customer Defects Per Pre-GA MR**
- **% of customers with defect within 3m. of install**

---

**Axes**:
- **Quantity**: The y-axis represents the quantity of defects or customer defect rates.
- **Time**: The x-axis denotes time periods (r1.1, r1.2, r1.3, r2.0, r2.1, r2.2) indicating release cycles or time intervals.
Testing Hypothesis 1: Defect Density Reflects Customer Quality

![Graph showing defect density over time and customer defect percentages.]

- **Customer Defects Per Pre-GA MR**
- **C** - % of customers with defect within 3 months of install

**Legend:**
- Black line: Customer Defects Per Pre-GA MR
- Red dashed line: % of customers with defect within 3 months of install
Testing Hypothesis 1: Defect Density Reflects Customer Quality

![Graph showing defect density and customer satisfaction over releases](image)

- **Customer Defects Per Pre-GA MR**
- **% of customers with defect within 3m. of install**

**Legend:**
- Black line: Customer Defects Per Pre-GA MR
- Red line with circle: % of customers with defect within 3m. of install

**Axes:**
- **X-axis:** Release versions (r1.1, r1.2, r1.3, r2.0, r2.1, r2.2)
- **Y-axis:** Quantity (0.00, 0.05, 0.10, 0.15)

**Annotations:**
- Better: Increases in defect density correlate with better customer satisfaction.
- Worse: Increases in defect density correlate with worse customer satisfaction.

**Figure Description:**
- The graph illustrates the relationship between defect density and customer satisfaction across different release versions. The x-axis represents the release versions, and the y-axis represents the quantity of defects. The black line indicates the trend of customer defects per pre-GA MR, while the red line with circles shows the percentage of customers with defects within 3 months of installation. The graph suggests that as defect density increases, customer satisfaction tends to decrease (worse), and vice versa (better).
Testing Hypothesis 1: Defect Density Reflects Customer Quality

Perfect anti-correlation?!
Trying Another Product

Perfect anti-correlation again?!
Why customers like high defect density?
Why customers like high defect density?
Why customers like high defect density?

Customers don’t care about defect density

- Most customers try to avoid bugs
  - By not jumping to a major dot zero release
  - By not installing immediately when new release is available

Software salesmen don’t care about defect density

- They want their customers to avoid bugs
  - By warning about products that are likely to cause problems

Software support people don’t care about defect density

- They want their customers to report as few problems as possible
  - By delaying wide installation of new releases
Lemma 1: Major Releases Have Few Customers

Minor releases have two to five times more customers

Note: based on 38 major and 49 minor releases in 22 products
Commandment 3
Thou Shell Have a Constant Rate of Customer Issues

Mechanism

- The only thing customers like less than a Bug is
Commandment 3
Thou Shell Have a Constant Rate of Customer Issues

Mechanism

- The only thing customers like less than a Bug is
  - The bug that does not get fixed for a long time
Commandment 3
Thou Shell Have a Constant Rate of Customer Issues

Mechanism
- The only thing customers like less than a Bug is
  - The bug that does not get fixed for a long time
- Team handling customer issues can not expand and collapse instantaneously and has limited throughput

Evidence

Monthly number of new customer issues is relatively constant
# Law of Minor Release

## Formulation
Minor releases have high defect density but low chances that any given customer will observe a defect.

## Definition
Major Releases Have More Code Change

## Mechanism

<table>
<thead>
<tr>
<th>Numerator</th>
<th>Denominator</th>
<th>CQ</th>
<th>Defect Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affected systems</td>
<td>Systems installed within 7m of GA</td>
<td>Stay constant</td>
<td>Customer reported defects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Move in opposite directions</td>
<td>The size of the release Effort/Changes</td>
</tr>
</tbody>
</table>

A. Mockus (Avaya Labs Research)
Law of Minor Release

Formulation
Minor releases have high defect density but low chances that any given customer will observe a defect

Definition
Major Releases Have More Code Change

Mechanism

<table>
<thead>
<tr>
<th>Numerator</th>
<th>Denominator</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CQ</strong></td>
<td>Systems installed within 7m of GA</td>
</tr>
<tr>
<td><strong>Affected systems</strong></td>
<td>Stay constant</td>
</tr>
<tr>
<td><strong>Stay constant</strong></td>
<td>Move in opposite directions</td>
</tr>
<tr>
<td><strong>Defect Density</strong></td>
<td>Customer reported defects</td>
</tr>
<tr>
<td><strong>The size of the release Effort/Changes</strong></td>
<td></td>
</tr>
</tbody>
</table>
Law of Minor Release

Formulation
Minor releases have high defect density but low chances that any given customer will observe a defect.

Definition
Major Releases Have More Code Change

Mechanism

<table>
<thead>
<tr>
<th>Numerator</th>
<th>Denominator</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Affected systems</td>
<td>Systems installed within 7m of GA</td>
</tr>
<tr>
<td>Stay constant</td>
<td>Move in opposite directions</td>
</tr>
<tr>
<td>Customer reported defects</td>
<td>The size of the release Effort/Changes</td>
</tr>
<tr>
<td>Defect Density</td>
<td></td>
</tr>
</tbody>
</table>
Law of Minor Release

Formulation
Minor releases have high defect density but low chances that any given customer will observe a defect.

Definition
Major Releases Have More Code Change

Mechanism

<table>
<thead>
<tr>
<th>Numerator</th>
<th>CQ</th>
<th>Defect Density</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Customer reported defects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The size of the release Effort/Changes</td>
</tr>
</tbody>
</table>

Denominator

| Systems installed within 7m of GA | Stay constant | Move in opposite directions |

A. Mockus  (Avaya Labs Research)  Law of Minor Release  Aug 2013  24 / 38
Law of Minor Release

Formulation
Minor releases have high defect density but low chances that any given customer will observe a defect

Definition
Major Releases Have More Code Change

Mechanism

<table>
<thead>
<tr>
<th>Numerator</th>
<th>Denominator</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ</td>
<td></td>
</tr>
<tr>
<td>Affected systems</td>
<td>Systems installed within 7m of GA</td>
</tr>
<tr>
<td>Stay constant</td>
<td>Move in opposite directions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Defect Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer reported defects</td>
</tr>
<tr>
<td>The size of the release Effort/Changes</td>
</tr>
</tbody>
</table>
Law of Minor Release

Formulation
Minor releases have high defect density but low chances that any given customer will observe a defect.

Definition
Major Releases Have More Code Change

Mechanism

<table>
<thead>
<tr>
<th>CQ</th>
<th>Defect Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerator</td>
<td>Customer reported defects</td>
</tr>
<tr>
<td>Denominator</td>
<td>The size of the release Effort/Changes</td>
</tr>
</tbody>
</table>

- CQ: Affected systems
- Stay constant
- Move in opposite directions
- Systems installed within 7m of GA
# Law of Minor Release

## Formulation
Minor releases have high defect density but low chances that any given customer will observe a defect.

## Definition
Major Releases Have More Code Change

## Mechanism

<table>
<thead>
<tr>
<th>Numerator</th>
<th>CQ</th>
<th>Defect Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affected systems</td>
<td>Stay constant</td>
<td>Customer reported defects</td>
</tr>
<tr>
<td>Systems installed within 7m of GA</td>
<td>Move in opposite directions</td>
<td>The size of the release Effort/Changes</td>
</tr>
</tbody>
</table>
Discussion

- There exist Laws of Software Evolution, but
  - Focus on short-term, repeating relationships with a clear mechanism
  - Look outside SW cave to observe them
  - Chose practical questions

- Practice hints
  - Development process view does not represent customer views
  - Maintenance — the most important quality improvement activity
Audris Mockus.
Empirical estimates of software availability of deployed systems.

Audris Mockus.
Organizational volatility and its effects on software defects.
In *ACM SIGSOFT / FSE*, pages 117–126, Santa Fe, New Mexico, November 7–11 2010.

Audris Mockus and David Weiss.
Interval quality: Relating customer-perceived quality to process quality.

Audris Mockus, David M. Weiss, and Ping Zhang.
Understanding and predicting effort in software projects.

Audris Mockus, Ping Zhang, and Paul Li.
Drivers for customer perceived software quality.
Abstract

Traditionally software evolution models consider long-term trends, but at such scales the primary driving factors are extrinsic to software: the changes in technology landscape or business environment. However, shorter software cycles recur in a predictable manner and can be best explained by a customer-provider equilibrium that leads to an apparent paradox of software quality: the best quality releases have the most defects. In OSS and in commercial products customers/end users are critical contributors to software quality improvement: they discover and report defects that can not (or are too costly to) be discovered otherwise. As new functionality is delivered in major releases, reliability conscious customers typically stay on the sidelines until the second minor release with properly working features, bug fixes, and stability improvements arrive. The major releases, thus, are used by fewer customers and, consequently, have fewer customer-reported issues. Understanding such predictable software cycles and the mechanisms underlying them is essential for effective software quality improvement and customer satisfaction.
Audris Mockus
Avaya Labs Research
233 Mt. Airy Road
Basking Ridge, NJ 07920
ph: +1 908 696 5608, fax: +1 908 696 5402
http://mockus.org, mailto:audris@mockus.org

Audris Mockus is interested in quantifying, modeling, and improving software development. He designs data mining methods to summarize and augment software change data, interactive visualization techniques to inspect, present, and control the development process, and statistical models and optimization techniques to understand the relationships among people, organizations, and characteristics of a software product. Audris Mockus received B.S. and M.S. in Applied Mathematics from Moscow Institute of Physics and Technology in 1988. In 1991 he received M.S. and in 1994 he received Ph.D. in Statistics from Carnegie Mellon University. He works at Avaya Labs Research. Previously he worked in the Software Production Research Department of Bell Labs.