CS2212
Introduction to Software Engineering

Quality Concepts

Ask Questions Live
cs1.ca/ask

What is Quality?

Philosophical View

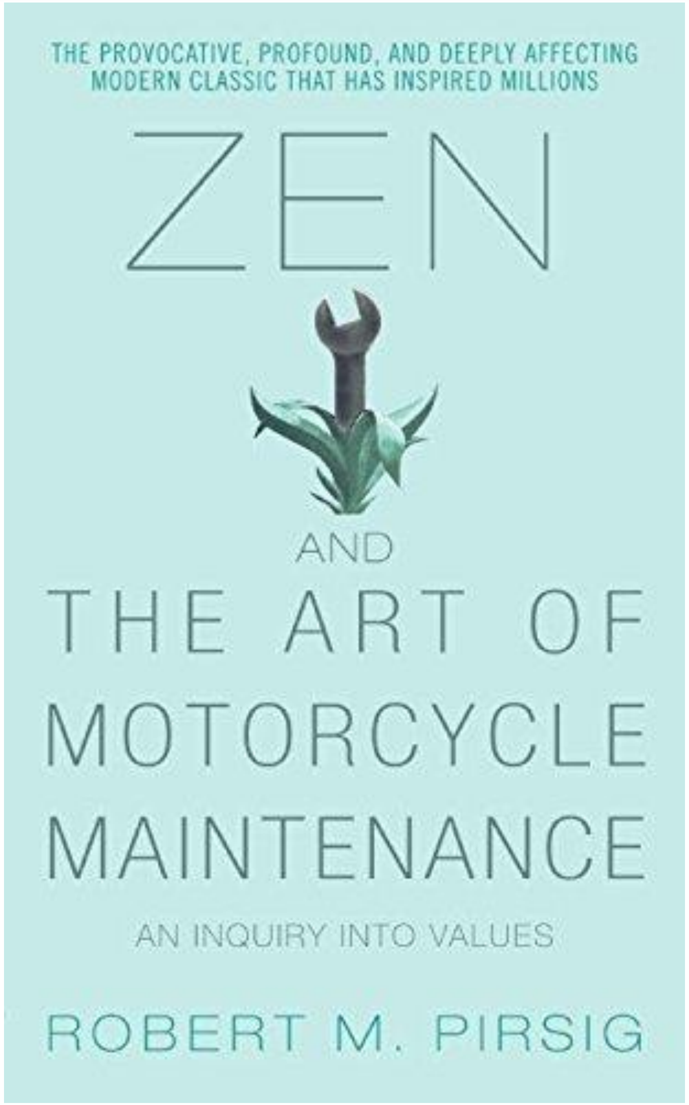
Quality . . . you know what it is, yet you don't know what it is. But that's self-contradictory.

But some things are better than others, that is, they have more quality. But when you try to say what the quality is, apart from the things that have it, it all goes poof! There's nothing to talk about.

But if you can't say what Quality is, how do you know what it is, or how do you know that it even exists? If no one knows what it is, then for all practical purposes it doesn't exist at all.

But for all practical purposes it really does exist. What else are the grades based on? Why else would people pay fortunes for some things and throw others in the trash pile.

What the hell is Quality? What is it?


THE PROVOCATIVE, PROFOUND, AND DEEPLY AFFECTING MODERN CLASSIC THAT HAS INSPIRED MILLIONS
ZEN
AND
THE ART OF MOTORCYCLE MAINTENANCE
AN INQUIRY INTO VALUES
ROBERT M. PIRSIG

What is Quality?

Pragmatic Views

- **Transcendental View:** argues (like Persig) that quality is something that you immediately recognize but cannot explicitly define.
- **User View:** sees product quality in terms of meeting the end-user's specific goals.
- **Manufacturer's View:** defines quality in terms of making sure a product conforms to its original specification.
- **Product View:** suggests that quality can be tied to inherent characteristics (for example: functions and features) of a product.
- **Value-Based View:** measures quality based on how much a customer is willing to pay for a product.

What is Quality?

Software Views

For software, three kinds of quality may be encountered:

- **Quality of Design:** encompasses **requirements**, **specifications**, and the **design** of the system.
- **Quality of Conformance:** is an issue focused primarily on **implementation**.
- **User Satisfaction** = **compliant product + good quality + delivery within budget and schedule**.

What is Quality?

Ultimately, **quality** encompasses all of these views and more.

Software Quality

Software quality can be defined as:

*An **effective software process** applied in a manner that creates a **useful product** that provides **measurable value** for those who produce it and those who use it.*

Effective Software Process

- Establishes infrastructure that supports building a high-quality software product.
- The **management aspects** of process **create the checks and balances** that help avoid project chaos.
- **Software engineering practices** allow the developer to **analyze the problem and design a solid solution.**
- **Umbrella activities** have as much to do with quality as any other part of software engineering practice.

Software Quality

Software quality can be defined as:

An ***effective software process*** *applied in a manner that creates a* ***useful product*** *that provides* ***measurable value*** *for those who produce it and those who use it.*

Useful Product

- A **useful product** delivers the content, functions, and features that the end-user desires.
- Must deliver these assets in a **reliable**, **error free way**.
- **Satisfies requirements** that have been explicitly stated by stakeholders.
- Also **satisfies implicit requirement** that are expected of all high-quality software (e.g. ease of use).

Software Quality

Software quality can be defined as:

*An **effective software process** applied in a manner that creates a **useful product** that provides **measurable value** for those who produce it and those who use it.*

Measurable Value

- By **adding value for both the producer and user of a software product**, high quality software provides **benefits for the software organization** and **the end-user community**.
- The **software organization** gains added value because high quality software requires less maintenance effort, fewer bug fixes, and reduced customer support.
- The **user community** gains added value because the application provides a useful capability in a way that expedites some business process.

Why Does Quality Matter?

- Countless stories of poor software quality and its impact on people and organizations around the world.
- CIO Magazine, *Let's Stop Wasting $78 Billion a Year*:
  - *"American **businesses spend billions** for software that doesn't do what it's supposed to do."*
- ComputerWorld once lamented that:
  - *"Bad software plagues nearly every organization that uses computers, causing **lost work hours** during computer downtime**, lost or corrupted data**, **missed sales opportunities**, **high IT support and maintenance costs**, and **low customer satisfaction**."*

Who is at Fault?

Software quality remains an issue, but who is to blame?

- **Customers blame developers**, arguing that sloppy practices lead to low-quality software.

- **Developers blame customers** (and other stakeholders), arguing that irrational delivery dates and a continuing stream of changes force them to deliver software before it has been fully validated.

Quality Factors

- How do we access and evaluate quality?

- Many software quality models and standards have been proposed in software engineering literature.

- **ISO 25010 standard** offers two models of quality:

  - **Quality in Use Model**: describes characteristics appropriate when considering using a product in a particular context (e.g. using the product on a specific platform by a human).

  - **Product Quality Model**: describes characteristics that focus on both the static and dynamic nature of computer systems.

ISO 25010: Quality in Use Model

- **Effectiveness:** Accuracy and completeness with which users achieve goals.
- **Efficiency:** Resources expended to achieve user goals completely with desired accuracy.
- **Satisfaction:** Usefulness, trust, pleasure, comfort
- **Freedom from Risk:** Mitigation of economic, health, safety, and environmental risks.
- **Context Coverage:** Completeness, flexibility.

ISO 25010: Product Quality Model

- **Functional Suitability:** Complete, correct, appropriate.
- **Performance Efficiency:** Timing, resource use, capacity.
- **Compatibility:** Coexistence, interoperability.
- **Usability:** Appropriateness, learnability, operability, error protection, aesthetics, accessibility.
- **Reliability:** Maturity, availability, fault tolerance, recoverability.
- **Security:** Confidentiality, integrity, authenticity, accountability.
- **Maintainability:** Reusability, modifiability, testability, modularity.
- **Portability:** Adaptability, installability, replaceability.

Qualitative Quality Assessment

- **Quality factors** focus on the complete software product and can be used as a generic indication of the quality of an application.
- These **quality factors** can be assessed **qualitatively** or **quantitatively**.
- **Example of qualitative assessment:**
  - Your team might decide to **create a user questionnaire** and a **set of structured tasks for users to perform** for each quality factor you want to assess.
  - You might **observe the users while they perform these tasks** and have them **complete the questionnaire when they finish**.
  - For some quality factors it **may be important to test the software in the wild** (or in the production environment).

Qualitative Quality Assessment

- When accessing **useability**, your team should create questions to evaluate each of the **ISO 25010 useability** subfactors *(Appropriateness, learnability, operability, error protection, aesthetics, accessibility)*.

- **Example Questions for Useability:**
  - How quickly can users determine whether the software can be used to complete their task or not? (appropriateness).
  - How long does it take users to learn how to use the system functions needed to complete their task? (learnability).
  - Is the user able to recall how to use system functions in subsequent testing sessions without having to relearn them? (learnability).
  - How long does it take users to complete tasks using the system? (operability).
  - Does the system try to prevent users from making errors? (error protection).
  - Does the system allow users to undo operations that may have resulted in errors? (error protection).
  - Do answers give favorable responses to questions about the appearance of the user interface? (aesthetics)
  - Does the interface conform to the expectations set forth by the UI design golden rules? (accessibility).
  - Does the user interface conform to the accessibility checklist items required for the intended users? (accessibility).

Quantitative Quality Assessment

- The software engineering community strives to develop precise measures for software quality.
- Internal code attributes can sometimes be described quantitatively using **software metrics**.
- If **software metric** values computed for a code fragment fall outside the range of acceptable values, it **may signal the existence of a quality problem**.
- **Metrics** represent indirect measures; we never really measure quality but rather some manifestation of quality.
- The complicating factor is the accuracy of the relationship between the variable that is measured and the quality of software.

Software Quality Dilemma

The Quality Dilemma:

1. If you produce a software system that has **terrible quality**, you **lose because** no one will want to buy it.

2. If you **spend infinite time**, **extremely large effort**, and **huge sums of money** to build a perfect piece of software, then it's going to take so long to complete and will be so expensive to produce that **you'll be out of business** as you either **missed the market window**, or you **exhausted all your resources**.

Need to find that magical middle ground where the product is **good enough** not to be rejected right away, but also not take too long or cost too much to complete.

Case Study: Animusic

Animusic was an animation company specializing in the 3D visualization of **MIDI-based music**.

In the early 2000s Animusic had tremendous succuss using their proprietary software that algorithmically synchronized animations to MIDI music.

Animusic used this software to create two DVD compilations:

- *Animusic: A Computer Animation Video Album*
- *Animusic 2: A New Computer Animation Video Album*

These products were well received and Animusic 3 was planned.

Case Study: Animusic

Case Study: Animusic

For Animusic 3, Wayne Lytle, the founder of Animusic wanted to remake their proprietary software from the ground up.

A Kickstarter was launched in 2012 to fund Animusic 3.

Kickstarter was successful, but Animusic has few updates of substance until August 2015 when Dave Crognale announces his departure from Animusic after 16 years.

Dave Crognale made up 50% of the Animusic team, leaving Wayne Lytle as the sole employee and responsible for the Art, Software, Music, and production.

Case Study: Animusic

Wayne Lytle pledged to keep working on Animusic 3 by himself stating that:

> *“If I’m living and breathing (barring some cataclysmic event)… YES, it will be finished.”*

This was 7 years ago, and there have been no updates since.

In August 2017, the Animusic headquarters was sold to RP Solutions, Inc.

The Animusic community considers the project to be dead, many citing Wayne’s perfectionism and lack of focus on the end product as being the cause of Animusic’s downfall.

Case Study: Animusic

What Went Wrong?

- In groups of 2 to 5 discuss the case of Animusic and try to brainstorm some reason why Animusic failed based on what we heard.
- If you have a computer, feel free to look up additional information about the company.
- Also brainstorm some ideas regarding how this could have been avoided. What steps could have Wayne Lytle taken to make sure Aniumusic 3 remained on track.

"Good Enough" Software

- One suggested solution to the **software quality dilemma** is "good enough" software.

- **What is "good enough" software?**
    - Delivers high quality **functions and features that end-users desire**, but at the same time it delivers other more obscure or **specialized functions and features that contain known bugs.**
    - The **hope is that the vast majority of end users either will miss the bugs** or will overlook them as they are otherwise very satisfied with what the software delivers.

“Good Enough” Software

- **Common occurrence in video game industry:**
  - Games are released in unfinished state, key features may be present, but others are missing or buggy.
  - Software patches and updates at a later date add back missing features, fix bugs, etc.
  - In some case, games are even sold as an empty box that only contains a download code (no DVD/Blu-ray).
  - Justification is to not miss critical release window (e.g. release in time for holiday season) and missing features can be developed and offered as an update or even as paid downloadable content (DLC).

“Good Enough” Software

- **Issues with “Good Enough” software development:**
  - “Good enough” may work in some application domains and for a few major software companies.
  - If you work for a small company and you deliver a “good enough” (buggy) product, you **risk permanent damage to your company’s reputation** and may lose customers.
  - You **may never get a chance to deliver version 2.0** because bad buzz may cause your sales to plummet and your company to fold.
  - If you work in certain application domains (e.g. real time embedded software) delivering “good enough” **may be considered negligent** and open your company to **expensive litigation**.

"Good Enough" Software

- No Man's Sky was an open world exploration and survival game developed and published by Hello Games.
- Hello Games made significant claims about the features of No Man's Sky before it's August 2016 release.
- Matt Kamen of Wired UK called No Man's Sky *"perhaps one of, if not the, most hyped indie titles in the history of gaming".*
- Hello Games took the "Good Enough" approach to software and launched the game with most hyped features missing including the highly anticipated multiplayer feature (but failed to inform users of this).
- By October 2016, the game had one of the worst user-based ratings on Steam, with an aggregate *"mostly negative"* average from more than 70,000 users.

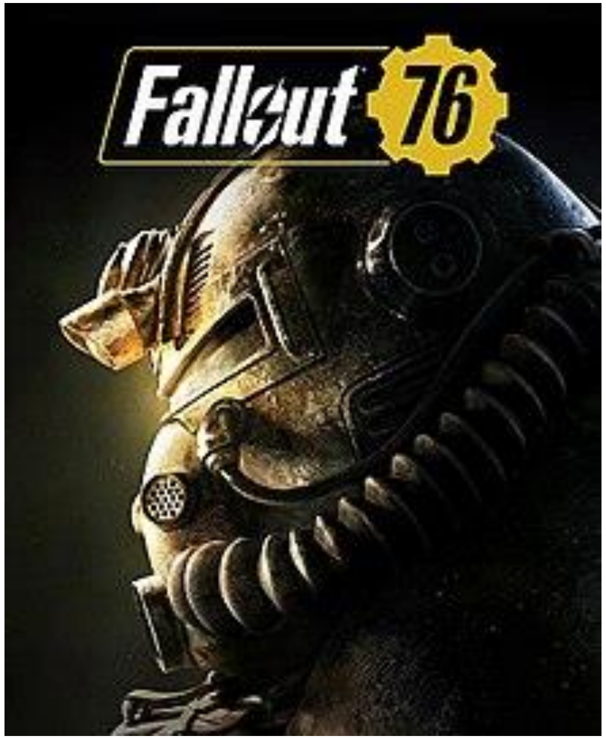
“Good Enough” Software

- In September 2016, the Advertising Standards Authority (ASA) of the United Kingdom, following on "several complaints", began an investigation into the promotion of No Man's Sky.
- ASA eventually ruled Hello Games was not in breach of their standards due to the misleading statements not technically being done via advertising (e.g. through interviews, trade shows, etc. rather than through official channels).
- Still had a profound impact on industry including causing changes to Steam and other storefronts refund and advertising policies.
- Hello Games eventually did update No Man’s Sky and added many of the missing features, but the impact of “Good Enough” design, likely cost them millions of dollars.

“Good Enough” Software

- **Additional Notable Examples:**

Cyberpunk 2077



Fallout 76

“Good Enough” Software

“A delayed game is eventually good, but a rushed game is forever bad.”

- Shigeru Miyamoto

The Cost of Quality

- **Delivering quality software has a cost**, in terms of time and money.
  - Fixating on perfection can cause these costs to lead to project failure as we saw with Animusic.
- **Lack of quality also has a cost**, not only to end users who must live with buggy software, but also to the software organization that has built and must maintain it.
  - Reputation damages may cost more than spending extra time creating a finished product.
  - Negligence and false advertising can also lead to potential legal liability.
  - We saw both in the case of Hello Game’s No Man’s Sky.

The Cost of Quality

Quality costs fall into one of the following categories:

- **Prevention Costs:** quality planning, formal technical reviews, test equipment, training.
- **Appraisal Costs:** conducting technical reviews, data collection and metrics evaluation, testing and debugging.
- **Internal Failure Costs:** rework, repair, failure mode analysis.
- **External Failure Costs:** complaint resolution, product return and replacement, help line support, warranty work

The Cost of Quality

Quality costs fall into one of the following categories:

- **Prevention Costs:** quality planning, formal technical reviews, test equipment, training.
- **Appraisal Costs:** conducting technical reviews, data collection and metrics evaluation, testing and debugging.
- **Internal Failure Costs:** rework, repair, failure mode analysis.
- **External Failure Costs:** complaint resolution, product return and replacement, help line support, warranty work

Prevention Costs

Prevention costs include:

- The cost of management activities required to plan and coordinate quality control and quality assurance
- The cost of added technical activities to develop complete requirements and design models
- Test planning costs
- The cost of all training associated with these activities

The Cost of Quality

Quality costs fall into one of the following categories:

- **Prevention Costs:** quality planning, formal technical reviews, test equipment, training.
- **Appraisal Costs:** conducting technical reviews, data collection and metrics evaluation, testing and debugging.
- **Internal Failure Costs:** rework, repair, failure mode analysis.
- **External Failure Costs:** complaint resolution, product return and replacement, help line support, warranty work

Appraisal Costs

Appraisal costs include activities that gain insight into various aspects of a software product's condition:

- The cost of conducting technical reviews for software engineering work products
- The cost of data collection and metrics evaluation
- The cost of testing and debugging

The Cost of Quality

Quality costs fall into one of the following categories:

- **Prevention Costs:** quality planning, formal technical reviews, test equipment, training.
- **Appraisal Costs:** conducting technical reviews, data collection and metrics evaluation, testing and debugging.
- **Internal Failure Costs:** rework, repair, failure mode analysis. (software)
- **External Failure Costs:** complaint resolution, product return and replacement, help line support, warranty work

Internal Failure Costs

Internal failure costs are incurred when you detect an error in a product prior to shipment; such costs include:

- The cost required to perform rework (repair) to correct an error
- The cost that occurs when rework inadvertently generates side effects that must be mitigated
- The costs associated with the collection of quality metrics that allow an organization to assess the modes of failure

The Cost of Quality

Quality costs fall into one of the following categories:

- **Prevention Costs:** quality planning, formal technical reviews, test equipment, training.
- **Appraisal Costs:** conducting technical reviews, data collection and metrics evaluation, testing and debugging.
- **Internal Failure Costs:** rework, repair, failure mode analysis.
- **External Failure Costs:** complaint resolution, product return and replacement, help line support, warranty work

customer services.

External Failure Costs

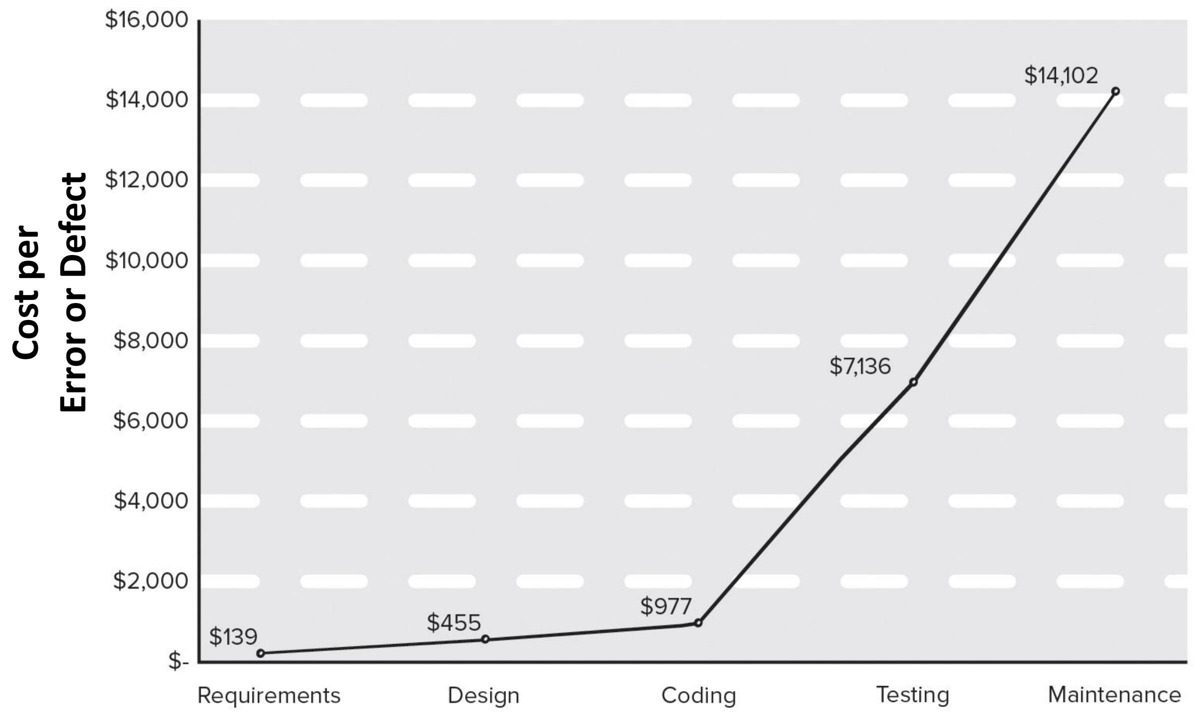
External failure costs are associated with defects found after the product has been shipped to the customer; examples include:

- Complaint resolution
- Product return and replacement
- Help line support
- Labor costs associated with warranty work

The Cost of Quality

- The costs to find and repair an error or defect
  increase dramatically as we go from prevention to
  appraisal to internal failure to external failure costs.
- This chart from our textbook (Figure 15.2) reflects real world data collected by Boehm and Basili.

Cost of Correcting Errors and Defects During Each Framework Activity



Source: Boehm, Barry and Basili, Victor R., "Software Defect Reduction Top 10 List," IEEE Computer, vol. 34, no. 1, January 2001.

Quality and Risk

“People bet their jobs, their comforts, their safety, their entertainment, their decisions, and their very lives on computer software. It better be right.”

Simply put, poor quality leads to risks, some of them very serious.

Quality and Risk

- **Low quality software leads to risk** for both the developer and the end user.
- Easy to see potential risk in application domains such as:
  - Self driving cars
  - Health and medicine
  - Aerospace
  - Embedded systems in factories/industry
  - Nuclear applications (health, power, weapons)
  - Military applications
- Significant risk also exists in more traditional domains in terms of liability.

Quality and Risk

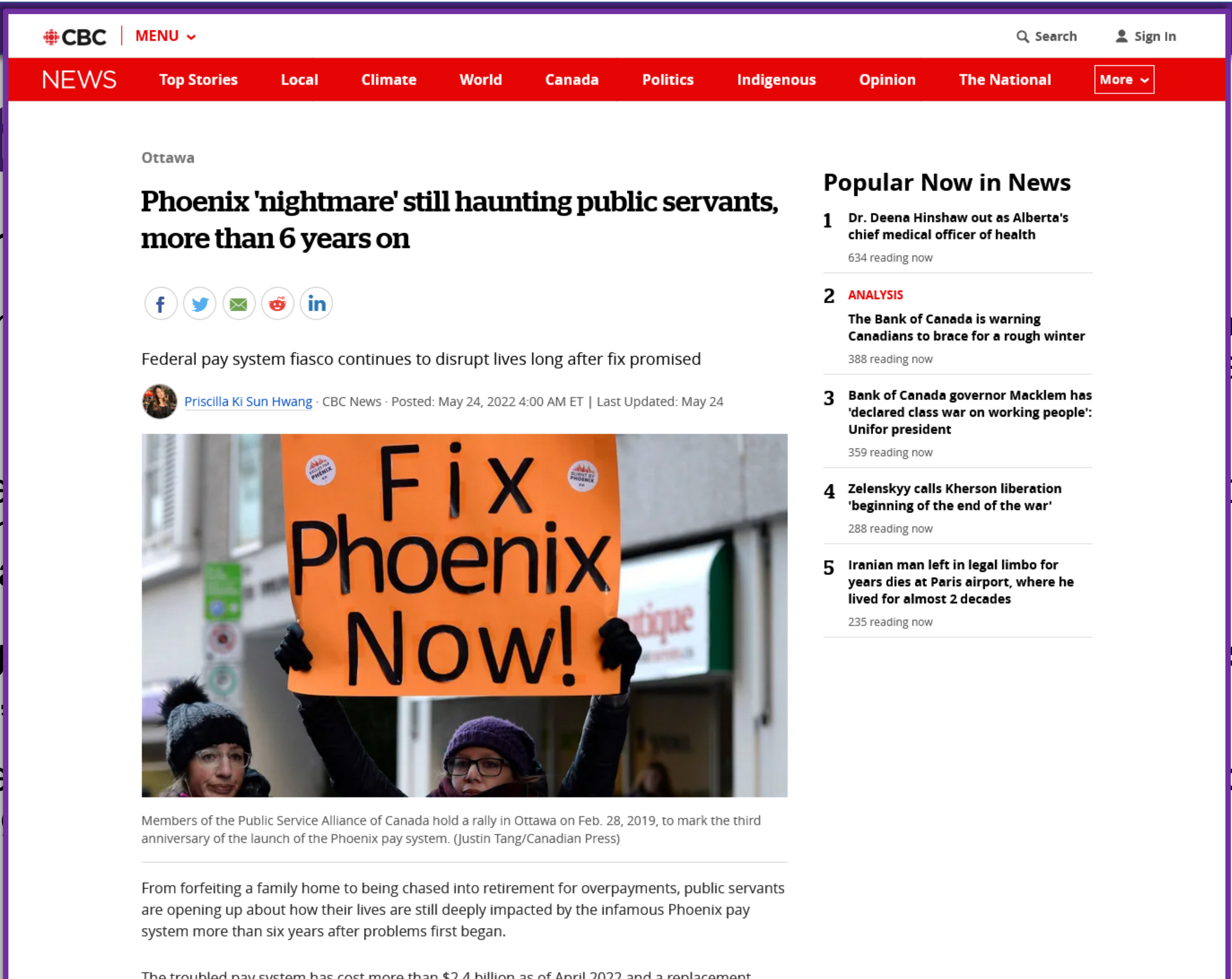
An all to common occurrence:

1. A governmental or corporate entity hires a major software developer or consulting company to analyze requirements and then design and construct a software-based "system".
2. The system might support a major corporate function (for example: pension management) or some governmental function (for example: healthcare administration or homeland security).
3. Work begins with the best of intentions on both sides, but by the time the system is delivered, things have gone bad.
4. The system is late, fails to deliver desired features and functions, error-prone, and does not get customer acceptance.
5. Litigation ensues.

Qualit

An all to co

1. A governn ... nsulting
   company ... are-based
   "system".
2. The syste ... on
   managem
   administra
3. Work beg ... system is
   delivered,
4. The syste ... one, and
   does not
5. Litigation



CBC | MENU   Search   Sign In

NEWS Top Stories Local Climate World Canada Politics Indigenous Opinion The National More

Ottawa

Phoenix 'nightmare' still haunting public servants, more than 6 years on

Federal pay system fiasco continues to disrupt lives long after fix promised

Priscilla Ki Sun Hwang · CBC News · Posted: May 24, 2022 4:00 AM ET | Last Updated: May 24

Members of the Public Service Alliance of Canada hold a rally in Ottawa on Feb. 28, 2019, to mark the third anniversary of the launch of the Phoenix pay system. (Justin Tang/Canadian Press)

From forfeiting a family home to being chased into retirement for overpayments, public servants are opening up about how their lives are still deeply impacted by the infamous Phoenix pay system more than six years after problems first began.

The troubled pay system has cost more than $2.4 billion as of April 2022 and a replacement

Popular Now in News

1 Dr. Deena Hinshaw out as Alberta's chief medical officer of health
634 reading now

2 ANALYSIS
The Bank of Canada is warning Canadians to brace for a rough winter
388 reading now

3 Bank of Canada governor Macklem has 'declared class war on working people': Unifor president
359 reading now

4 Zelenskyy calls Kherson liberation 'beginning of the end of the war'
288 reading now

5 Iranian man left in legal limbo for years dies at Paris airport, where he lived for almost 2 decades
235 reading now

Quality and Security

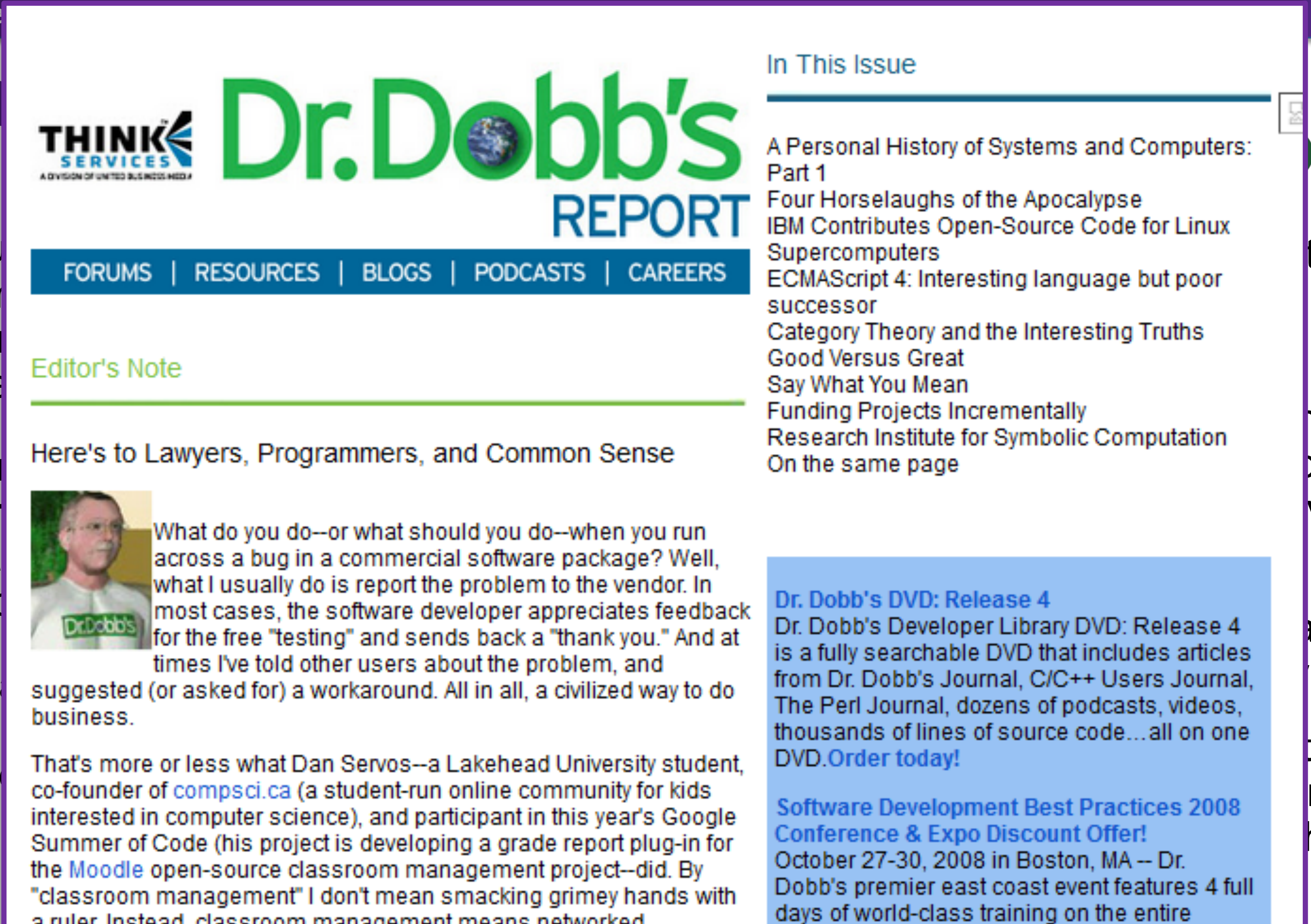
- **Low quality software is easier to hack** and can increase the security risks for the application once deployed.
- A **secure system cannot be built without focusing on quality** (security, reliability, dependability) during design.
- Low quality software is liable to contain architectural flaws as well as implementation problems (bugs).

Example: LANSCHOOL

- LanSchool is a classroom management software company that offers products for K to 12 that can control and monitor computer labs.
- In the early 2000s, LanSchool viewed security as an after thought rather central to their design and ignored most security best practices.
- During 2003 to 2008 vulnerabilities were discovered in LanSchool's protocol that allowed an attacker take over computers running the software and alter the audit log to cover up the hack.

Qua OL

- **Low qu** ted to **hack** ar matter risks for deploye
- A **secu** nst **withou** ons (securit via during c
- Low qu and contain are. impleme tation nt and h



THINK SERVICES A DIVISION OF UNITED BUSINESS MEDIA

Dr. Dobb's REPORT

FORUMS | RESOURCES | BLOGS | PODCASTS | CAREERS

In This Issue

A Personal History of Systems and Computers: Part 1
Four Horselaughs of the Apocalypse
IBM Contributes Open-Source Code for Linux Supercomputers
ECMAScript 4: Interesting language but poor successor
Category Theory and the Interesting Truths
Good Versus Great
Say What You Mean
Funding Projects Incrementally
Research Institute for Symbolic Computation
On the same page

Editor's Note

Here's to Lawyers, Programmers, and Common Sense

What do you do--or what should you do--when you run across a bug in a commercial software package? Well, what I usually do is report the problem to the vendor. In most cases, the software developer appreciates feedback for the free "testing" and sends back a "thank you." And at times I've told other users about the problem, and suggested (or asked for) a workaround. All in all, a civilized way to do business.

That's more or less what Dan Servos--a Lakehead University student, co-founder of compsci.ca (a student-run online community for kids interested in computer science), and participant in this year's Google Summer of Code (his project is developing a grade report plug-in for the Moodle open-source classroom management project--did. By "classroom management" I don't mean smacking grimey hands with a ruler. Instead, classroom management means networked

Impact of Management Decisions

Software **quality is often influenced as much by management decisions** as it is by technical decisions:

- **Estimation Decisions:** irrational delivery date estimates cause teams to take short-cuts that can lead to reduced product quality.
- **Scheduling Decisions:** failing to pay attention to task dependencies when creating the project schedule.
- **Risk-Oriented Decisions:** reacting to each crisis as it arises rather than building in mechanisms to monitor risks may result in products having reduced quality.

Achieving Software Quality

- Software quality doesn't just appear.
- Software quality is the result of good project management and solid engineering practice.
- This comes into play in four broad activities:
  - **Software engineering methods**
  - **Project management techniques**
  - **Quality control**
  - **Quality assurance**

Meskimen’s Law

“There’s never time to do it right, but always time to do it over again”

In practice, taking the time to do it right is almost never the wrong decision to make.