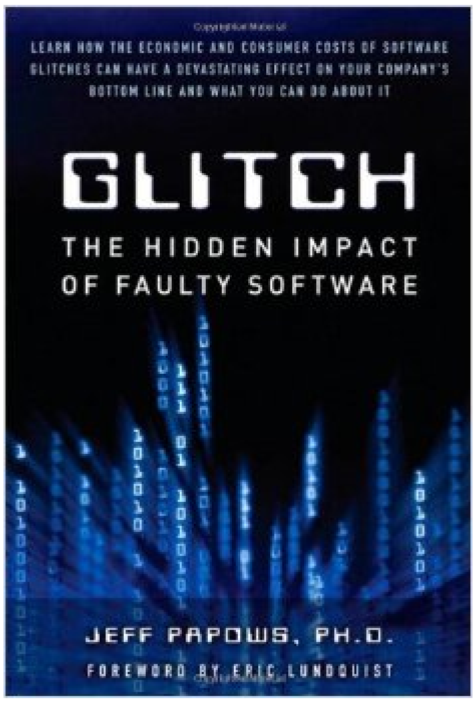
Glitch: The Hidden Impact Of Faulty Software


LEARN HOW THE ECONOMIC AND CONSUMER COSTS OF SOFTWARE GLITCHES CAN HAVE A DEVASTATING EFFECT ON YOUR COMPANY'S BOTTOM LINE AND WHAT YOU CAN DO ABOUT IT
GLITCH
THE HIDDEN IMPACT OF FAULTY SOFTWARE
JEFF PAPOWS, PH.D.
FOREWORD BY ERIC LUNDQUIST

Synopsis

Donâ ™t Let Software Failures Destroy Your Business   The growing impact of software failures on brands, customers, and business performance   How to govern software more effectively, prepare for glitches, and mitigate their impact   By Jeff Papows, Ph.D., one of the worldâ ™s most experienced software executives   Your software systems are the heart of your businessâ “and they may be vulnerable. In Glitch, industry leader Jeff Papows, Ph.D., outlines the three converging forces that are leading to the proliferation of glitches. Papows explains why and how these glitches are affecting businesses, government agencies, and consumers and provides recommendations as to what we can do about them. Â Written for senior decision makers, Papows explains why the risks of software failure are growing worse, not betterâ “and shows how software glitches can destroy both your profitability and your reputation. He also introduces proven governance techniques that can help you systematically find and fix weaknesses in the way you build, buy, and manage software. Â Donâ ™t fall victim to the next business software disaster. Read Glitchâ “and learn about the cultural, technical, and business issues behind software glitches, so you can proactively prevent them. Â

Book Information

Hardcover: 208 pages

Publisher: Prentice Hall; 1 edition (September 4, 2010)

Language: English

ISBN-10: 0132160633

ISBN-13: 978-0132160636

Product Dimensions: 6.4 x 0.7 x 9.3 inches

Shipping Weight: 14.4 ounces

Average Customer Review: 4.0 out of 5 starsÂ Â See all reviewsÂ (29 customer reviews)

Best Sellers Rank: #3,319,935 in Books (See Top 100 in Books)   #73 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Quality Control   #932 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Testing   #3734 inÂ Books > Textbooks > Computer Science > Software Design & Engineering

Customer Reviews

Given my (and many others') increasing frustration with consequences of buggy software, I was really hoping for this book to be useful at identifying the issues and proposing some solutions. Not only did the book come up short on both of these, but much of the book has nothing to with glitches.

Further there were numerous errors in much of the material, and most of the "solutions" proposed amounted to suggesting increased regulation, as if the government can demand bug-free software (when that isn't even defined) or "certified" software engineers, again not showing what kind of certification (or even training) can reduce bugs.Some examples of wrong, if not downright dangerous, advice:He points to Bernie Madoff as an example of the need for more government regulation (in general), ignoring the fact that preventing situations like Bernie Madoff was EXACTLY what the SEC was set up to, and even after 75 years was unable to do so.He also aims to expand government's consumption of technology - "without ... Web 2.0 capabilities on government websites, e-government initiatives will deliver diminishing returns. ... when ... [Web 2.0 capabilities are] unavailable on government sites, citizens will not return." This completely misses several major points - most notably, that government does not have competitors (for true government functions), and that as long as an electronic solution is easier than the alternative, it will succeed. Will anyone really choose to go in to the DMV for a renewal that can be done on the web, just because the website doesn't offer flashy graphics or collaboration features?In suggesting not to understand, much less upgrade mission-critical systems- "Know when to leave well enough alone. Given fluctuations in staffing and long-term investments in technology, the reality is that you simply won't fully understand certain applications. ... you will not always know exactly what makes them tick ...If it's a business-critical application, ... you may not need to know every line of code ... In this case, the risk of compromising the infrastructure by opening the application far outweighs the need to sate curiosity." One would think that if a system were business-critical, understanding every line would be business-critical. Or else one glitch could bring down the company, because no one will understand enough to repair it in a timely manner. Also, even the most green developer knows that "opening the application" (i.e. reviewing source code) does not entail ANY risk. Ignorance, however, that's risk.In proposing more attention being paid to cloud applications - "Should faulty software practices make their way into a cloud, they might impact a wider audience than a more traditional on-premises model of software ownership. Therefore we need to be that much more diligent when it comes to developing [cloud applications]" This is particularly disturbing as he earlier pointed out software bugs that killed people in a (not widely used) radiology application. "Diligence" should be a function of potential impact, not where an application is hosted, or even how many users it has.Even his definition of a software problem is questionable - "When I buy a flat-panel TV and it takes me four hours to make it work with the components, that's a software lifecycle problem. When I wait for three hours for what was supposed to be a one-hour brake pad replacement, that's a software life cycle failure."Speaking of definitions, he continually talks about "transparency" and "governance" without ever clearly defining those terms.In summary, while the author may have been CEO of Lotus Development Corp, it sounds like he was never actually a developer or even a tester, and hence appears to be a PHB (pointy haired boss) right out of Dilbert, spouting off dictates with no understanding of the real problems, much less solutions. What a shame.

Glitch is an interesting book, but to be honest I didn’t really enjoy it. The author is certainly an expert in the field and provides plenty of useful information, but at times I found it difficult to distinguish the stories from the proven content from the opinion.Each chapter has a topic and conclusion but the coverage feels anecdotal and often a bit light, padded by the included stories. Maybe I was a little sensitive while reading it, but the book is fairly short to begin with and while the stories, figures and tables are important to the text it means there is much less content than the 208 official pages. To me it felt more like a collection of articles or presentations than a book.The topics are wide ranging and touch many aspects of IT in business and the value of this book is in this breadth. The coverage can be used as a check list or refresher for your own IT reviews or provide suggestions on areas to focus in your own organisation, but you could possibly distil this into a one or two page list and move on.In the end I just wasn’t convinced that the book was worthwhile. It was light, padded and occasionally disjointed. Many of the stories weren’t software related and I couldn’t see how they served the book. It will look nice on the bookshelf but it didn’t deliver the experience I was hoping for.

This is a meaningful book but was written just before the explosion in mobile applications and the onslaught of malicious crimes. As such, the issues presented use examples that predict some of the problems of late, but would be more meaningful if they could have been included. The book does bring up important issues that are drowned out by more dramatic current events, such as the aging of COBOL resources, but without the current examples the very valid warnings are too easily dismissed. Lastly, the book’s timing causes it to miss the impact of such things as the Apple App Store and Google Play on the increasing speed of development, combined with the lack of expertise in security practices that can create new category of glitches that magnify existing conditions that contribute to errors. The rise of a global criminal class that can exploit otherwise benign errors is not given sufficient weight. Programming errors in the past that would go unnoticed have now become vulnerabilities that can be exploited. The glitches discussed are like quality issues found in domestic cars of the 80s and 90s. But current glitches are often akin to how well built cars will behave if chased by pursuers: cornering, acceleration, braking distances, engine resilience would all be measured differently given the higher stakes in which the car is forced to operate. Modern software attached to the Internet needs greater resilience and more programmer knowledge of safe coding practices. These are newer sources of glitches, along with the need for rapid release of patches, the difficulty of patch distribution, and more extensive and new types of testing. These sources of glitches and their dire effects are not sufficiently included in this discussion. At the same time, many things included remain problems and the book does well to remind us that not all software issues are only those we hear about in the news. The lesson is that older problems remain and cannot be ignored as we become distracted by newer ones.