**More Features, Fewer Tests?**

Cameron had been working as a software developer for about eight months at a startup that specialized in fintech solutions. The company was creating software to streamline loan approvals, leveraging a broader dataset and machine learning to make faster decisions.

Starting out as a Junior developer, Cameron had quickly made a name for himself and was promoted to a mid-level role after just six months, thanks to his proactive attitude and knack for solving complex bugs. His lead engineer praised his problem-solving skills and noted that he had an exceptional understanding of the codebase and a strong grasp of software principles.

Like all developers on the team, Cameron was part of an on-call rotation, stepping in whenever issues arose in the company’s 24/7 system. He was required to be on-call once every 10 weeks, and while issues were rare, the team took on-call responsibilities seriously, preparing for the unexpected.

One night during his on-call shift, Cameron received an urgent notification around 11:30 p.m.—the system was down. Immediately, he jumped into action. After a brief investigation, he found that the problem stemmed from a minor bug in the new loan application interface that had just been rolled out. While digging deeper into the logs, he noticed something else—the notification system that was supposed to alert applicants of their loan status had been failing silently for the past two weeks.

Cameron quickly resolved the initial bug, pushed a fix, and updated the necessary channels on Slack. He decided to tackle the notification issue the following morning, assuming it wasn’t critical.

The next morning, Cameron woke up early, grabbed a coffee, and began investigating the notification problem. Upon reviewing the code, he realized the notification service had no tests. He wrote a set of unit tests and, unsurprisingly, three of them failed. As he delved deeper, he discovered a recent deployment had introduced a subtle error that blocked notifications from being sent.

During the team’s daily standup, Cameron presented his findings. Before he could finish explaining how he was addressing the problem, his team lead and the engineering manager grew visibly concerned. They interrupted, asking why he hadn’t escalated the issue immediately the night before. What Cameron didn’t know was that the company had a legal obligation to inform applicants about their loan status within 48 hours. Failing to do so could result in fines of $10,000 per applicant, for each day past the deadline.

Cameron felt a rush of anxiety and frustration, realizing he had inadvertently put the company at risk.

Within minutes, a “crisis room” was formed, led by the team lead, to urgently investigate the matter. Meanwhile, the engineering manager was already notifying company executives and the finance team. Cameron explained to the team that he had written new unit tests that pinpointed the problem, and they worked together to patch the code. In under an hour, they deployed the fix to production.

A few weeks later, during his quarterly review, Cameron received a disappointing “needs improvement” rating. His manager cited the handling of the notification bug as the primary reason.

How could Cameron have approached the situation differently?