Software Quality Assurance

QA in Status Reports

"We are on-track. We haven't done any QA because we haven't started coding."

Which tells me …

- Our wireframes are untested and of suspect quality
- Our design may be severely flawed
- We don't know if we are even programming anything useful or suitable
- QA is an idea we know little about because we think it means "testing"
- We are thinking like coders, not developers.

What Is SQA?

Software Quality Assurance: a process which assures that all development activities, processes, methods, and work items are monitored against and comply with the suitable standards.

SQA incorporates all software development processes starting from defining requirements to coding until release.

Its prime goal is to ensure quality.

Impact of Software Quality

Customer adoption and retention rates

Product reviews and customer satisfaction

Company reputation and income

Development time and costs

Company survival

Top 6 Reasons Mobile Applications Crash

(https://techbeacon.com/app-dev-testing/top-6-reasons-mobile-apps-crash-how-best-avoid-murphy)

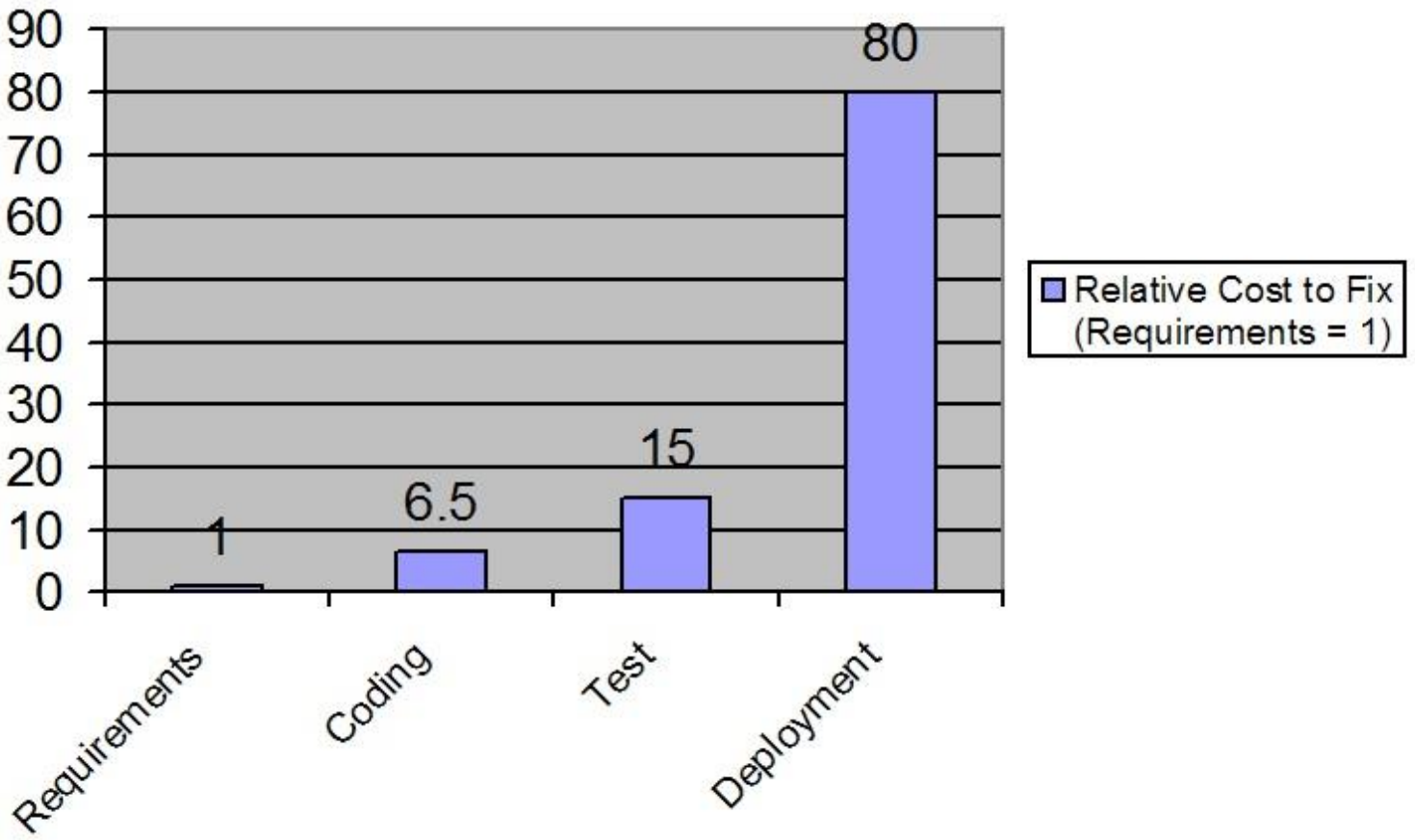
As we have seen:

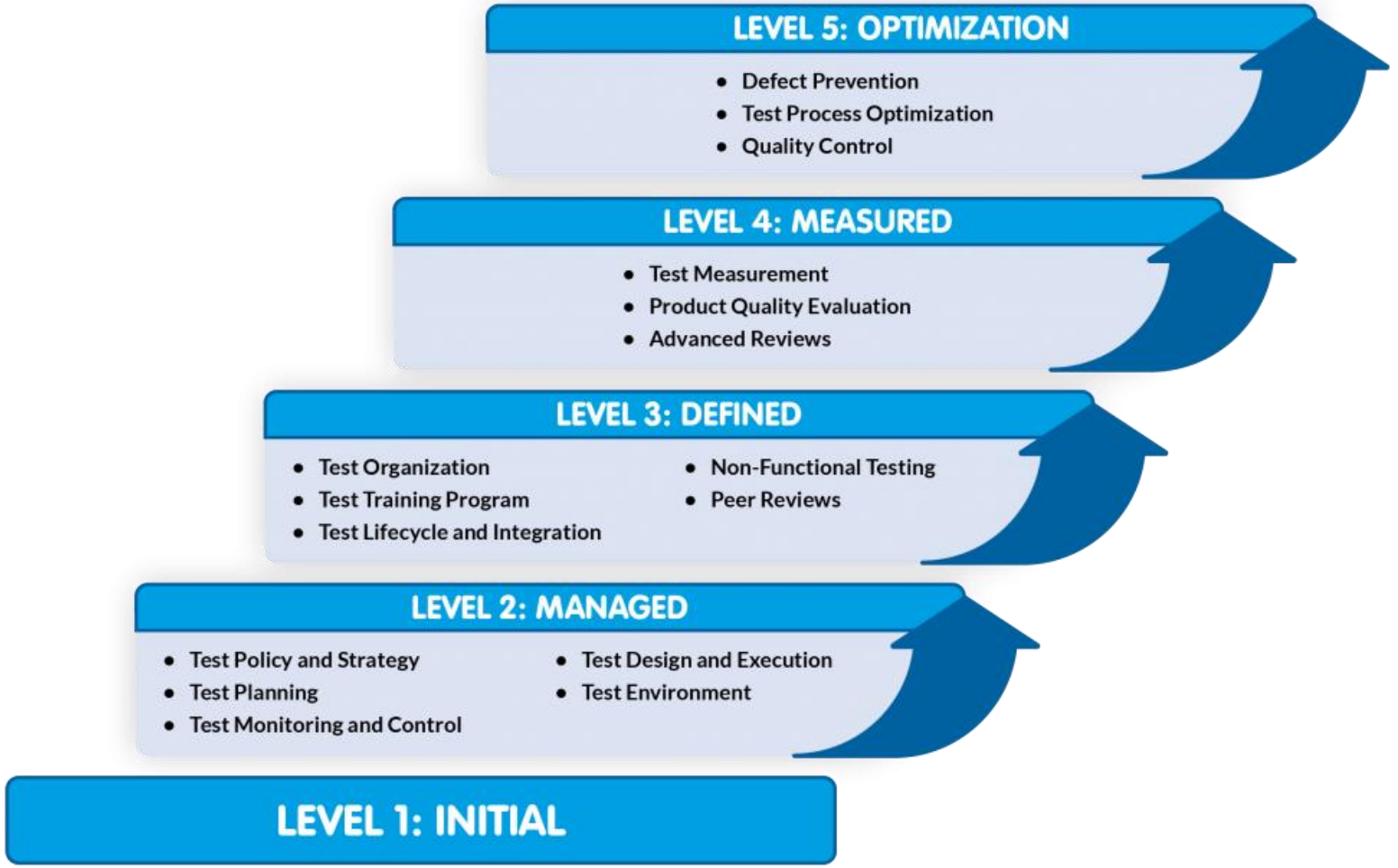
- 61% of users expect mobile apps to start < 4 seconds
- 49% want responses to inputs in < 2 seconds
- 53% of users uninstall an app that crashes, freezes, or has errors

These issues are most often caused by:

1. Memory management (surprisingly, much is caused by the SDK)
2. Software Lifecycle: Weak control over OS, APIs, libraries, frequent releases
3. Inadequate Testing (particularly when dev. uses emulators)
4. Poor network management (switching networks, lost data, …)
5. Failure to handle errors
6. Too much code

The C0$T of Faults





Levels of SQA

https://www.tmmi.org/

What is software quality?

- **Correctness**—the extent to which a program satisfies its specification
- **Reliability**—the extent to which software completes its intended task without failure in a specified environment (e.g., uptime)
- **Efficiency**—the hardware, software, and programming resources required by a program to perform its function
- **Security/Integrity**—the extent to which access to data and software by unauthorized persons is controlled
- **Usability/Accessibility**—the effort required to learn, use, enter data, and interpret program output.
- **Maintainability/Understandability**—the effort required to locate and fix program defects
- **Flexibility**—effort required to modify an existing software system
- **Testability**—effort required to create the test cases and procedures necessary to ensure software performs its required functions.
- **Portability/Interoperability**—effort required to transfer a program from one computing platform or operating environment to another
- **Reusability**—extent to which software components can be used in new applications
- **Responsiveness/Performance**—speed and perceived speed

Elements of SQA

There are 10 essential elements of SQA:

1. Standards (coding, legal, accessibility, etc.)
2. Technical reviews and audits
3. Software Testing for quality control
4. Error collection and analysis
5. Change management
6. Educational programs
7. Vendor management
8. Security management
9. Safety evaluation (users, h/w, privacy, etc.)
10. Risk management

What Needs Quality?

- Code (of course)
- Deployed system
- Hardware (as applicable)
- Tools, libraries, APIs, DBs
- Documentation (both internal and external)
- Design
- Requirements, use-cases, and functional specifications
- Training materials and tutorials, demonstrations

That is, EVERYTHING you create or use!

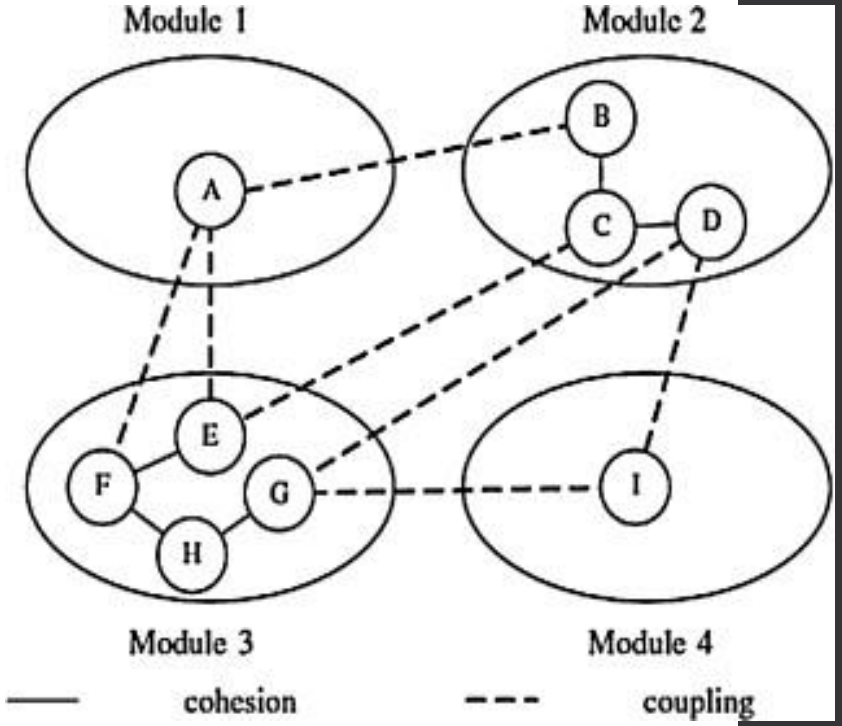
Techniques: Everything

- **Auditing**: inspection of the work products and its related information to determine if processes were followed or not.
- **Reviewing**: a meeting in which the software product is examined by both the internal and external stakeholders to seek their comments and approval.
- **Simulation**: a tool that models the real-life situation in order to virtually examine the behavior of the system under study.
- **Standardization**: comparing the software against applicable standards to ensure compliance (including Legal requirements)
- **Quantitative Evaluation**: evaluation of a product using metrics
- **Walkthroughs**: Product walkthrough is a kind of peer review where the developer guides the members of the development team to go through the product and raise queries, suggest alternatives, make comments regarding possible errors, standard violations or any other issues.
- **Prototyping**: trial version to explore correctness, suitability, and expose faults earlier

Design Inspection

Design inspection is done using a checklist that inspects:

- General requirements and design
- Functional and Interface specifications
- Conventions
- Requirement traceability
- Structures and interfaces
- Logic
- Performance
- Error handling and recovery
- Testability, extensibility
- Coupling and cohesion
- SOLID



Techniques: Code

- **Code Inspection**: a formal review that does static testing to find bugs and avoid defect growth in the later stages. It is done by a trained mediator/peer and is based on rules, checklist, entry and exit criteria. The reviewer should not be the author of the code.
- **Functional Testing**: verifies what the system does without considering how it does it. This type of black box testing mainly focuses on testing the system specifications or features.
- **Static Analysis**: It is a software analysis that is done by an automated tool without actually executing the program. This technique is highly used for quality assurance in mission critical software.
- **Walkthroughs**: Code walkthrough is a kind of peer review to detect faults, bugs, improvements, risks, and other issues
- **Path Testing**: It is a white box testing technique where the complete branch coverage is ensured by executing each independent path at least once.
- **Stress Testing**: This type of testing is done to check how robust a system is by testing it under heavy load i.e. beyond normal conditions. Similar to volume testing (e.g., with large data sets).

Risks

- A risk is an unwanted **event or situation** that has a probability of occurring and can cause some form of damage to a system.
- The severity of a risk is a measure of the amount of damaged caused.
- Risks can be reduced (lower the probability of occurring) and/or mitigated (lower the severity or amount of damaged caused).
- If the probability can be reduced to zero, the risk can be avoided.

Risk Management

Quality can be improved by:

Avoiding risks: Not doing things that could cause quality to be jeopardised	**Mitigating risks**: Doing things to reduce damage that harms quality

But we can't do these if we don't know what the risks are, thus requiring:

Risk identification: Finding the risks	**Risk monitori**ng: Determining if/when they are occurring


Excellent
Above Average
Average
Below Average
Poor
Mobile App Quality Takes More than Just Software Testing Automation