# Industry Thoughts on Software Quality

*Unrestricted version.*

A range of people working in the software industry, many of them St Andrews graduates, were surveyed on their views on Software Quality.

This version contains comments from contributors who indicated that their responses could be freely circulated. Their views do not necessarily represent those of their current or previous employers. They retain copyright in their contributions.

*Text presented in this style has been paraphrased using ChatGPT.*

This project is covered by St Andrews ethics approval code CS17781.

*Graham Kirby*
*25 February 2025*

graham.kirby@st-andrews.ac.uk
https://www.st-andrews.ac.uk/computer-science/people/gnck

**List of Contributors**

- Victor Andrei, financial sector
- Robert Bain, Addepar
- Dimitrios Bairaktaris, business sector
- Ben Birt, Google
- Nathan Blades, financial sector
- Tage Borg, Scrive
- Jonathan Childers, tech sector
- Théophile Choutri, tech sector
- Charlotte Clapp, automotive sector
- Adam Copp, tech sector
- David Craddock
- Masih H. Derkani
- Fraser Dunlop
- Tomas Dzetkulic, tech sector
- Ade Fabola, consulting sector
- Alex Farkas
- Craig Gilchrist, financial sector
- Henry Hargreaves, financial sector
- Alice Herbison, HE sector
- Kobe Housen, tech sector
- John, tech sector
- Jonathan Jouty, tech sector
- Rafnas Arathan Kandi, tech sector
- Ali Khajeh-Hosseini, Infracost
- Samuel Koch, tech sector
- Kevin Lau
- Rhona McCracken, tech sector
- Angus Macdonald, Google
- Iain McDonald, tech sector
- Ross Nicoll, tech sector
- Paul Normington, Udemy
- Adrian O'Lenskie, tech sector
- Folarin Omotoriogun, Google
- Nathan Owens, tech sector
- Daniel Patterson, tech sector
- Douglas Pearson, online gaming sector
- Indika Perera, HE sector
- Mikhail Piankov, tech sector
- Yulia Samoylova, consulting sector
- Lakshitha de Silva, gaming sector
- Neil Skilling, tech sector
- Ashley Sole, tech sector
- Thomas Strang, cyber security sector
- Daniel Swarup, Amazon
- Bence Szabó
- Valentin Tunev, tech sector
- Robbie Wallace, tech sector
- Neil Wells
- Titus Winters, tech sector
- Evangelos Zirintsis

## Victor Andrei

Senior software engineer in financial sector

Graduated BSc (Hons) Computer Science, St Andrews, 2016

**What do you think are the most important aspects of software quality? How would you define what it is?**

*Software quality is a complex, multi-dimensional concept. It includes measurable traits like functionality, efficiency, and durability, as well as softer aspects like readability and branding. Quality software effectively meets user needs and remains manageable for developers.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*Knowledge essential for a software engineer to develop high-quality products hinges on a blend of technical expertise and interpersonal abilities. The foundational technical skills encompass algorithmic knowledge, proficiency in data structures, understanding of complexity, efficiency, security, hardware considerations, formal logic, and networking principles. These are universally taught in computer science and software engineering curricula worldwide and are categorized as 'hard skills'.*

*In addition to these technical skills, there are 'soft skills' that are equally crucial in the software industry. These include effective communication, adept time management, a collaborative approach, openness to diverse perspectives, delegation capabilities, and proficient people management. Unlike hard skills, these soft skills are typically honed through real-world experience rather than formal education alone.*

*Overall, a balanced mastery of both hard and soft skills is essential for software engineers aiming to thrive in the dynamic and collaborative environment of software development.*

## Robert Bain

Software engineering manager at Addepar

linkedin.com/in/robert-bain-272b8564/

Graduated BSc (Hons) Internet Computer Science, St Andrews, 2006

**What do you think are the most important aspects of software quality? How would you define what it is?**

*Software quality involves doing many things right across different areas:*

- ***User-facing aspects**: ensuring the software meets requirements, functions correctly, is bug-free, responsive, and provides a positive user experience.*
- ***Developer-facing aspects**: implementing continuous integration and delivery (CI/CD), code reviews, comprehensive testing, quick build times, automated security checks, static analysis, and reliable, rapid release processes.*
- ***Dependencies**: keeping dependencies minimal, up-to-date, and well-chosen, with appropriate licensing.*
- ***Code quality**: following best coding practices, maintaining simplicity, extensibility, minimal boilerplate, DRYness, and effective logging.*
- ***Miscellaneous**: keeping documentation and tech stack current, managing technical debt, and setting up monitoring and alerting systems.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*When hiring:*

- ***Junior/graduate roles**: emphasize potential, looking for a solid foundation in computer science and problem-solving skills. Specific knowledge can be developed on the job.*
- ***Senior roles**: seek experience and dedication to the quality aspects mentioned earlier.*

*In education:*

- ***Degree focus**: distinguish between computer science (theoretical) and software engineering (practical).*
- ***Teaching recommendations**: introduce unit testing and Test-Driven Development (TDD) from the start, making it a staple in practical exercises to foster better coding practices and confidence in refactoring.*
- ***Project delivery**: teach students to produce buildable projects (e.g. jars) rather than just code, prepping them for real-world deployment.*
- ***Logging**: instil a clear understanding of logging practices and what sensitive information to avoid logging.*
- ***Early code/PR reviews**: implement code and pull request reviews early to enhance skills, especially for students needing more support.*

*Overall, prioritize essential skills that equip students for successful careers, while recognizing that some practical insights are best gained through experience.*

## Dimitrios Bairaktaris

Director in business sector

Graduated PhD Computer Science, St Andrews, 1993

**What do you think are the most important aspects of software quality? How would you define what it is?**

Total cost of ownership over the lifecycle of the product including supportability, extensibility and reliability, information security including end-user and cloud security. Future proof technology stack to ensure longevity and improve maintainability.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

A thorough understanding of the product/software lifecycle including support and maintenance is essential. It is important that the theoretical basis for this is included in the curriculum. A good knowledge and practical experience of software quality tools and techniques is also useful. An overview of software quality tools will be a valuable addition to the curriculum. Ultimately, practical experience, learning through mistakes and hands-on coaching/shadowing, will be required, especially where complex project/software is concerned.

**Further comments**

In addition to the standard teaching material around coding standards and testing strategies, an understanding of mission critical applications and how to architect resilient systems will be most useful to creating brilliant solutions.

## Ben Birt

Staff software engineer at Google

linkedin.com/in/ben-birt-0064b314/

Graduated BSc (Hons) Computer Science, St Andrews, 2010

**What do you think are the most important aspects of software quality? How would you define what it is?**

Comprehensibility is the most important aspect of software quality. I'm defining it separately from 'readability', which is of course important, but causes people to dig into arguments about line length, verbosity, etc etc; and I think those arguments miss the point. Ideally, any code reader should have the easiest time possible in comprehending the code's intention. This might be achieved through clean and clear naming, well-designed abstractions, the use of common design patterns and/or common infrastructure/libraries, etc.

Robustness to change is also critically important. It must be possible to extend the software to introduce new features or satisfy new requirements without unexpectedly breaking the software. This obviously overlaps with the above (good abstractions) but also includes the ability to quickly and cheaply detect regressions (e.g. with unit tests).

General robustness is critical too: I'd much rather use software with fewer features but which works all the time, every time; as opposed to some multi-functional all-singing-all-dancing software which breaks.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- Design patterns.
- Ability to design clean, extensible abstractions.
- Ability to gauge when and where is the best place to test logic.

I think all of these should be taught in a degree, through practical projects, preferably working alongside experienced software engineers (e.g. professors?) from whom students can learn these skills. Of course any of these will be improved—in some cases dramatically—by exposure to day-to-day work as a software engineer, but I think with some creative thinking, university degrees could be significantly improved with respect to learning these skills.

## Nathan Blades

Software engineer in financial sector

Graduated BSc (Hons) Computer Science, St Andrews, 2016

**What do you think are the most important aspects of software quality? How would you define what it is?**

High quality software can be extended and replaced with confidence. When working with high quality software, developers are able to understand if a change is safe to make, without having to reason about many different places in the code.

High quality software can normally be understood immediately at first glance, and is never misleading.

High quality software has unit tests which cover all dynamic parts of its functionality. If it is difficult to write unit tests for a piece of code, that code is probably not high quality.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

I prefer to hire developers that are good at reviewing and critiquing other people's code. This skill can only be gained through practice and experience, but it is not a given—many senior developers never learn this skill, because you have to intentionally make time for code review. I think a degree is the perfect time to start practicing this.

I prefer to work with developers that are disciplined enough to unit test their code, but I find it difficult to test for this in an interview. Writing unit tests is easy, and regularly writing unit tests will force the rest of your code to be better. But since professional developers are almost never forced to write unit tests (outside of specific industries), it's entirely up to the individual's discipline / diligence. I don't think you can teach that in a degree because it is a personality trait.

**Further comments**

Books like *The Pragmatic Programmer*, *Clean Code* (e-book) and *Code Complete* (e-book) should be mandatory reading for someone that is learning to write high quality code.

I always recommend Sandi Metz to any new developers who actually care about the quality of their code. Her talks are incredible.

## Tage Borg

CTO at Scrive

linkedin.com/in/tageborg

**What do you think are the most important aspects of software quality? How would you define what it is?**

Availability and usability. If either fails, the software is not usable to the end user, which is paramount. A road with a bunch of potholes and too many red lights is better than one that ends in the middle of nowhere or leads to the wrong city.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

It's not possible to teach "experience", but anecdotal evidence of failures (Ariane 5 failure, imperial-or-metric units for Mars landing, etc) helps. What can be taught is the importance of integration and system tests. And the fact that God will always win in the race of "better programs" vs "users".

## Francesco Burato

Software development engineer at Adobe

linkedin.com/in/francesco-burato-61563830/

**What do you think are the most important aspects of software quality? How would you define what it is?**

I define software quality as a loose set of properties of the source code, the build infrastructure, and the deployment pipeline of a software project that make it easier for developers to deliver value at low cost without damaging existing workflows. Some instances of such properties include:

- Testability: all code paths should be covered by a reliable test suite, which is designed to verify the behaviour rather than the implementation. The test suite must be easy to read and update, it must run quickly and frequently.
- High cohesiveness/low coupling: the abstract components that make up the software project should follow the single responsibility principle, i.e. there should be only one reason and one reason only that would require to change the component.
- Repeatable build: it should take no more than one command for a developer to produce a build given a certain source reference (in git, that would be a commit SHA) which is functionally identical to a running instance.
- Local first: there should be scaffolding in place within the project that allows the developer to start the software project in their development environment with one or two commands, reducing the feedback loop for manual testing.
- Documented: every piece of information that is required to understand the reason for the existence of the project, how it is maintained, and why it is developed in a certain way should be easily reachable and if possible context relevant (e.g. comments).

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- Unit testing/TDD: a candidate should be able to retrofit good tests for a given implementation and should be able to build a test suite before implementing any feature. This can be taught within a degree as laboratory exercises.
- SOLID design: a candidate should be able to reason about the high level design of a project and should be able to produce one that follows the SOLID principle. More importantly, they should be able to articulate why such designs are "good". This can be taught in principle, but the subtlety of why they are important is unlikely to be grasped without experience.

**Further comments**

I believe it is difficult to teach software quality. The importance of tests, documentation, build replicability, etc. becomes painfully clear when your are called in the middle of the night because of an outage that is costing thousands of pounds per minute. Before then, it's difficult to appreciate how important it is and aspiring developers are likely to consider it the "right thing to do" that can be skipped on occasion rather than a fundamental aspect of software development. Many things can be covered in a course, but it's only with experience that the student will have the necessary epiphanies that will make them realise how important it is.

## Jonathan Childers

Software engineer in tech sector

**What do you think are the most important aspects of software quality? How would you define what it is?**

Testing: is the software testable? Are the constituent parts that make up the software decoupled enough to allow unit testing in isolation? Do the unit tests for each module meaningfully test the APIs that the module exposes to the rest of the system? Can you build integration tests that can test high level use cases of the system? Can all of this testing be automated? Can this testing be used to detect and reject changes that will break the system? When new bugs are identified, can the test suites be expanded to ensure that there will not be a regression in the future that re-introduces the bug?

Architecture: is the system designed in a scalable way? Are the components tightly or loosely coupled? How complicated is it to add new features? How complicated is it to fix bugs?

Measurement: how will we measure the effectiveness of the system? What metrics are useful to the specific problem domain? Performance? Memory usage? Power consumption? Binary size? Network throughput? Daily active users? Ad clicks? Bill of materials cost? Number of zero day vulnerabilities discovered? Which of these properties that you are interested in can be scientifically measured to determine whether your system is improving or regressing?

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Questions that I ask myself when interviewing candidates:

1. Can the candidate come up with a workable high level solution to a problem? Can they articulate trade-offs in their design with respect to performance, scalability, complexity, ease of testing, etc? Can they clearly communicate their ideas to me? Can they translate their ideas into code that I can understand and agree solves the problem? How do they validate that the code does what they said it does?
2. If hiring for a more senior position: does the candidate have subject matter expertise relevant to my team's problem domain? Can the candidate create solutions for complex systems that require high level architecture? Does the candidate have experience mentoring junior engineers or leading a team?
3. Since my work is security focused: does the candidate have the "Evil Bit" i.e. the ability to think like an attacker? When presented with a possible vulnerability, does their mind go towards "what is the worst thing that an attacker could do with this capability", or does it go towards rationalizing why this isn't a real problem?

Aside from hiring, I think that a lot of students go into software engineering expecting to spend most of their time in their professional career coding, and that their performance and advancement prospects will be determined by how good they are at coding. My experience has been that although you have to be good at coding to get a job, if you want to advance you have to be good at working with other people. That means writing design docs, having stakeholder meetings, discussing requirements with other teams, meeting with product managers to understand business needs, mentoring junior engineers, providing good code review feedback, organizing and planning your team's work, and a lot of other soft skills that are not heavily focused on in a computer science curriculum. I think that a lot of students would be well served by taking at least one or two classes on non-CS skills like project management, leadership, technical/business writing, communication, etc.

## Théophile Choutri

Backend engineer in tech sector

**What do you think are the most important aspects of software quality? How would you define what it is?**

Software quality is the property of a software system to fulfil its purpose and be easily adapted when the requirements inevitably evolve.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Humility, being collaborative, experience with advanced testing techniques like property testing, habit of documenting systems.

**Further comments**

Languages like Haskell help with software quality, as they are able to express high-level concepts in a way that makes the code reviewable by domain experts. Using types as a form of domain checking is a very powerful technique, which removes a lot of mental burden from the programmer.

## Charlotte Clapp

Software engineer in automotive sector

Graduated BSc (Hons) Computer Science, St Andrews, 2017

**What do you think are the most important aspects of software quality? How would you define what it is?**

Easily testable, easily understood (someone else should be able to work in it quickly without spending time trying to understand what's going on) and easily extendable (no tech debt—code should be at a high enough quality that if you want to add features there is no additional time spent redoing or reworking existing code to support this).

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- Attention to detail / thorough: I've seen smart software engineers try to finish code super quickly just to meet a deadline; as expected this always backfires and just results in time needed to rework the tech debt later. Think this is a skill that can be taught and emphasised at school (I think it was emphasised heavily at St Andrews).
- Experience extending code to support additional features (unfortunately think this is only learned through work experience).

## Simone Conte

Software Engineer at Adobe

linkedin.com/in/simoneivanconte/

Graduated PhD Computer Science, St Andrews, 2019

**What do you think are the most important aspects of software quality? How would you define what it is?**

*To me, these factors are key:*

- ***Reliability**: can your code run smoothly 24/7, or are you frequently getting woken up in the middle of the night to fix issues?*
- ***Reproducibility**: does your code behave predictably and consistently? Do your tests effectively catch errors when changes are made?*
- ***Understandability**: are you using appropriate abstractions? Is the code easy to read and modify?*
- ***Measurability**: are you tracking enough metrics (such as logs, telemetry, traces)? How much insight do you have into your software's behaviour?*
- ***Changeability**: introducing changes should be straightforward and easy.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*We primarily look for three key qualities in candidates:*

- *humility*
- *a strong work ethic*
- *intelligence*

*While these traits may sound broad, they allow us to hire regardless of specific expertise in tools or programming languages.*

*When evaluating technical knowledge or skills, I believe it's essential to focus on a deep understanding of fundamentals, as these skills transfer well across different tools and projects. Some of these core competencies (in no particular order) include:*

- *The ability to write code across different levels of abstraction (e.g., applying principles like single responsibility, dependency inversion, etc.)*
- *A solid grasp of concepts like encapsulation, composition, and polymorphism*
- *Strong familiarity with functional programming patterns, which are increasingly being adopted by many tools and languages*
- *Knowledge of how to make code testable, and how to write meaningful and comprehensive tests*
- *An understanding of various types of tests (unit, functional, integration, smoke, performance, etc.)*
- *A strong foundation in statistics*
- *The ability to read and review code effectively—this involves not just critiquing, but also suggesting alternative solutions, thinking about edge cases, and understanding other people's code*
- *Strong presentation and writing skills—this job isn't as solitary as it's often portrayed in films*

*There are numerous skills that are developed through experience. In fact, one might say that any skill (whether mentioned here or not) evolves with time. Experience reinforces understanding and helps form more solid opinions on what works and what doesn't.*

*Skills will also vary based on the work. For example, someone focused on UI development should understand design principles and how accessibility impacts web components, whereas someone working on backend development might need a completely different set of skills.*

*That said, over the years, I've found that having a strong foundation in core principles (as outlined above) and maintaining a customer-centric mindset (whether you're building for users interacting with your UI or for engineers using your APIs) will take you far.*

**Further comments**

*Here are some great reads I recommend:*

- *Clean Code (though take some of its advice with a grain of salt)*
- *Clean Architecture*
- *A Philosophy of Software Design (excellent, especially if you have some coding experience)*
- *The Mythical Man-Month*
- *Re-engineering Legacy Software*
- *Statistics Done Wrong*

*I also highly recommend the talk "How to Speak" by Professor Winston from MIT.*

## Adam Copp

Senior software engineer in tech sector

Graduated BSc (Hons) Computer Science, St Andrews, 2012

**What do you think are the most important aspects of software quality? How would you define what it is?**

There are so many things we could consider "software quality". I think there's two main perspectives from a software (as opposed to product) perspective: some quality is reflected directly in the product, and some is reflected indirectly, in the effectiveness of the team building it.

*Directly expressed quality*

This kind of quality is fairly easy to talk about and to understand, because you can look at existing software products and see good and bad examples:

- Apps which are so big you have to delete other stuff on your device.
- Web Apps which take so long to load you go elsewhere.
- Games with bugs which break the experience.
- Software which crashes regularly and loses you data.

These are largely directly measurable / observable by automated tooling or analytics. Making this measurability and observability possible is an important part of building software of high quality for the user.

*Indirectly expressed quality*

This kind of quality is harder to see, because you must have worked in teams which builds software to see why bad practice comes about and the effects first-hand.

Much of this is learning good software design, which is a balancing game between competing concerns, "increased complexity" being an oft-encountered and oft-underrated concern. For example:

- Do you build this kind of configurability/extensibility into a system in anticipation of a certain kind of change, even though that extensibility has an ongoing complexity cost which slows the team down?
- Do you use Framework X, which promises increased performance but which has a lack of observability tooling which might prevent you from noticing production issues?
- Is it worth wrapping the network stack in case we need to change to a different stack in future and then we only change the wrapper? Even though it means playing API treadmill with the underlying API and maintaining the wrapper?
- How do you handle legacy in a large system—it's inevitable that all systems have legacy parts at some point, so do you aggressively rewrite? Invest in education and knowledge sharing? Invest in isolation of components (even at the cost of complexity)?

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Directly expressed quality is much easier to teach, and I'd expect the average interview candidate to be able to respond to "can you talk to me about what you think are important aspects of software quality" with answers regarding crash rates, latencies, robustness, etc.

I'd be impressed with a candidate who could talk to me about how they might make sure those things are at a reasonable level, but I don't care about specific tech/tools. If they mentioned observability tooling, analytics tooling, performance testing, release gating etc. I specifically don't care about how Jenkins works or how to configure Github CI etc.

I'd be even more impressed with a graduate candidate who was able to talk about design tradeoffs and the cost of complexity, but I think this is something that can't be truly understood without experience working in larger systems for an extended period of time.

## David Craddock

Test manager, senior automation engineer

**What do you think are the most important aspects of software quality? How would you define what it is?**

Working, effective, software that meets requirements and is bug and defect-free.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

It would be useful to have the equivalent of the ISTQB in a degree format. In terms of standard education I highly recommend checking out the RST—Rapid Software Testing—courses by James Bach in terms of adopting a 'tester mindset'—that approach really is the strongest form of taught testing theory I have found.

**Further comments**

A module on philosophy and critical thinking would be very useful so you can analyse things from a logical perspective.

## Masih H. Derkani

CTO, principal architect

Graduated PhD Computer Science, St Andrews, 2015

**What do you think are the most important aspects of software quality? How would you define what it is?**

*Two crucial aspects of software quality include its adaptability to meet unexpected requirements at minimal expense and the quantifiable assurance of fulfilling those requirements reliably.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*I believe open-source contributions and working openly via technical blog posts are crucial. Problem-solving skills are essential when faced with ambiguity, requiring the ability to discern necessary actions. Understanding fundamental algorithms and their complexities is foundational; while algorithms are taught in degrees, complexity awareness develops with experience. Integrating open-source contributions into university assignments, particularly in senior years, would greatly enhance practical skills alongside traditional coursework.*

**Further comments**

*Balancing between engineering software to achieve the highest quality and promptly delivering revenue-generating products creates a conflict of interest. The cost of achieving absolute quality can reduce profits, which are the primary goal for most companies: to generate more revenue than expenses. Consequently, the concept of software quality shifts to making optimal trade-offs among stakeholders. This approach focuses on evolving software affordably to adapt to unexpected needs while ensuring it reliably delivers on its intended functionality.*

## Fraser Dunlop

Software engineer

Graduated MMath (Hons) Mathematics, St Andrews, 2013

**What do you think are the most important aspects of software quality? How would you define what it is?**

Software is a creative art form as much as it is an engineering discipline so assessing quality is always going to be relative to some frame of reference. A nice example of this is the indie video game title 'Undertale' which is well renowned as a work of great artistic quality; it is also well renowned for containing a 1000+ case long switch statement.

From the developer's perspective quality can also be relative to the culture surrounding a language or project. If the goal of a tool is to optimize for speed or algorithmic efficiency at all costs then quality will be measured quite differently compared to a project in which other aspects (such as ease of extension or addition of new features) are more important.

From my perspective there is an "I know it when I see it" aspect to quality. When code is packaged as a module or library that doesn't seem to need anything added or taken away then that's a pretty good sign of quality. If you are looking at a very long file and wondering why on earth they didn't abstract out any functions then you are probably on the other end of the quality spectrum.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Functions are our primary means of creating abstraction in software so having an appreciation for a functional language would be something I would be looking for though it is not essential. An eye for abstraction is essential however and I would look for those skilled in; building algorithms compositionally using functions, deconstructing and refactoring code into a more readable state, creating elegant designs that are modular, exposing interfaces while hiding implementation details.

A computer science degree needs to cover a lot of ground and many of the skills required to create quality software will come with experience. That said I think exposing students to functional programming principles and concepts early can help them to create better software. Many inexperienced programmers (even some experienced ones) seem to forget that they can use functions to structure their code in a more readable way or perhaps don't appreciate that doing so might help them to create code of better quality. It's important to have balance however and even for experienced developers the path to quality software can involve iterating on software that's somewhat provisional until you understand what quality looks like for your particular application.

## Tomas Dzetkulic

**What do you think are the most important aspects of software quality? How would you define what it is?**

- Correctness: software does what it promises to do.
- Robustness: software gracefully degrades under unexpected circumstances.
- Performance: software is fast.
- Reliability/testability: changes can be made with confidence that things won't break.
- Mutability: software is reasonably easy to change under new requirements.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- Algorithms and data structures for performance.
- Engineering best practices + engineering experience for the rest.

## Ade Fabola

User experience research consultant in consulting sector

Graduated PhD Computer Science, St Andrews, 2018

**What do you think are the most important aspects of software quality? How would you define what it is?**

- *Efficiency refers to the software's resource demands for development and operation, including costs and time, as well as the computing power needed for optimal performance on users' devices.*
- *Usability indicates how intuitive and user-friendly the software is.*
- *Security and privacy assess the safety of data and vulnerability to malicious attacks.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*Skills essential for software development include writing efficient code that optimizes resource usage, adhering to defined parameters and constraints, collaborating effectively across different teams and disciplines such as data scientists, product/project managers, and user experience professionals. It's crucial for students to learn these skills during their degree programs to prepare them for working in software development teams. They should also gain knowledge of typical organizational practices like agile methodologies (e.g. sprints, scrum), equipping them with the necessary techniques to thrive in their careers.*

## Alex Farkas

Principal security engineer

**What do you think are the most important aspects of software quality? How would you define what it is?**

*In my profession, several principles are crucial, adapted here for a software context:*

1. ***Provenance**: understanding who authored the software, their qualifications, and whether they have legitimate intentions. Assessing third-party dependencies for reliability and legitimacy.*
2. ***Repeatability**: ensuring open-source software can be rebuilt from its source code with transparency and verification. Avoiding dependency on opaque binary releases that obscure how they were created.*
3. ***Stability**: prioritizing software reliability and consistency for end users, emphasizing usability over technical details like programming languages or frameworks. Minimizing disruptive changes without user consultation.*
4. ***Ethics and business models**: examining how software influences consumer behavior and business models, highlighting ethical concerns such as forced feature changes or migrations. Emphasizing transparency and consumer choice in software development practices.*

*These principles underscore the importance of software quality beyond technical functionality, addressing broader implications for users and businesses.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*Critical thinking in Computer Science is crucial, emphasizing its scientific foundation. Many developers lack understanding of fundamental mathematics like Turing completeness, complexity theory, strong typing, and numerical analysis. Some adhere unquestioningly to Microsoft Windows as a universal standard, while others blindly follow industry buzzwords without empirical support. I've encountered students who inquire about generative AI's role in gateway technology, highlighting the need for deeper critical inquiry. While understanding technical details such as strong typing and execution methods is important, true excellence in development comes from critically evaluating the scope, purpose, strengths, and weaknesses of one's work.*

**Further comments**

*Some of my comments might come across as cynical or even overly cautious, but they echo sentiments shared by many in both cyber security and traditional software engineering communities. While vendors have the freedom to innovate, there's a concern that traditional engineering principles like rigorous requirements analysis, user involvement, configuration management, and thorough testing have been overlooked in software development.*

*Consider a scenario where software updates solely address defects without adding new features (and potential bugs), altering existing functions, or necessitating a full download of a large distribution image. Imagine the simplicity and reliability it could bring to users' experiences.*

## Craig Gilchrist

Software engineer in financial services sector

Graduated BSc, St Andrews, 2004

**What do you think are the most important aspects of software quality? How would you define what it is?**

I'm a keen proponent of the 'developer experience' and consider that a core part of software quality. That quality is driven by the readability, maintainability and testability/reliability of the code.

The caveat I would give is that the last decade of my work has been in web development, where raw performance tends to be less critical that in some industries. In previous roles I've needed to comply with MISRA (Motor Industry Software Reliability Association), or fit narrow constraints for embedded systems, which add further parameters to what constitutes 'quality' software in terms of compliance or performance/size, but even then the building blocks of software quality are that the engineers building and maintaining it can understand it, trust its robustness, and quickly iterate/improve it.

Given that I've tended to work in systems that lean towards object oriented design, I like the SOLID principles and similar patterns that favour keeping units of code with a singular purpose, concise and free of unnecessary complexity, and easy to test.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

While I have mixed feelings about my personal experiences of whiteboarding in interviews, one of the 'soft(er)' skills I try to identify in candidates is the ability to explain code—whether it's something they have written or something they are unfamiliar with—and to examine it critically to identify ways to simplify or abstract the component logical behaviours, as well as think about edge cases, input sanitization, and so forth.

That feeds into the role a team and its individual members play in ensuring quality. Being able to work collaboratively on breaking down requirements, identifying key behaviours and components during the planning stage, through to (empathetic!) code reviews, which I think are a necessary part of ensuring quality is maintained through peer review and feedback.

Perhaps one that would be very difficult to teach in a university environment is contributing to a large scale software project, where there are (potentially a very large number of) opinions on what constitutes quality, what are the most important facets, etc, and where standards, code styles, etc have been agreed long before your arrival.

**Further comments**

It's been a long time since I went to university, and I certainly wasn't the most attentive student, but it was quickly apparent in my first job that I lacked experience **really** working with others on software. Not just splitting the work into separate pieces and going away and working on them independently, but actively working day-in-day-out with other engineers towards a common goal, with continuous feedback, discussions of ideas, using source control and code reviews. I've learned so much from pairing with other engineers, and through code reviews, that I wouldn't/couldn't have learned working by myself in a purely solo environment. Incidentally, this is one of the biggest shortcomings I have seen with self-taught developers, in that they tend to focus on building their own portfolio projects rather than contributing to open source or working in larger groups where they are beholden to others' standards of what is an acceptable level of quality.

## Henry Hargreaves

Full stack engineer in financial services sector

Graduated BSc (Hons) Computer Science, St Andrews, 2019

**What do you think are the most important aspects of software quality? How would you define what it is?**

Clean code (concise, atomic functions, simple and easy to read/follow).

Unit tests.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Good understanding of algorithms and data structures (mostly leetcode style questions).

Problem solving skills including how they approach a problem.

**Further comments**

Systems design, including new technologies and where/when they could be used and their strengths/weaknesses.

## Alice Herbison

Front-end developer in HE sector

Graduated MSc Human Computer Interaction, St Andrews, 2014

**What do you think are the most important aspects of software quality? How would you define what it is?**

I think quality can be grouped in general into two categories following the traditional "build the thing right" and "build the right thing" concepts—developer-facing quality and user-facing quality.

User-facing quality involves a product that meets the intended user needs plus I would also encourage a little "sparkle" in terms of nice-to-have aspects. The smoothest apps and software involve clean and well-tested layouts, features borne from user research and feedback and smooth user journeys.

Secondly, I believe developer-facing quality is equally important. You can't maintain a wonderful user-facing app if the code itself is not DRY (do-not-repeat yourself), organised and clearly commented. Poor quality code leads to unmaintainable technical debt. A balance needs maintained between easy-to-read code and succinct, functional techniques—helped by peer code review. To onboard developers easily and make frequent personnel changes smooth means that in some regards the code should self-document. Additionally, company developer documentation should absolutely be part of a developer's role. Bus factors and gate-keeping are awful to deal with as a team. Bugs and problems in the software are also poor quality indicators showing lack of testing and forethought/planning (for some—sometimes they're inevitable!).

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

If the role application involved a software test we looked for code that basically was the antithesis of spaghetti and used the correct, relevant external packages in the right way.

Things that could be included in software teaching:

- Peer review experience.
- Quality assurance techniques and exposure.
- Refactoring practice of taking non-DRY code and improving it.
- Some understanding of user needs research.

Skills gained through experience would include general improvements in coding techniques and interpersonal skills from giving and receiving peer reviews.

## Kobe Housen

Site reliability engineer in tech sector

**What do you think are the most important aspects of software quality? How would you define what it is?**

- Using efficient algorithms to not create unneeded bottlenecks.
- Robust against changes in the code i.e. a developer can't easily break the code by making an uninformed choice.
- Well documented, such that developers can easily comprehend how to use a library or the existing code and modify it with understanding.
- Complicated logic is tested. Don't have change detector tests, but make sure that complex code has been tested so you have confidence changes don't introduce bugs.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

The person needs to be informed about the points I gave above, and then the skill would mainly be around how to structure APIs in a comprehensible manner.

## Jonathan Jouty

Software engineer in tech sector

Graduated BSc (Hons) Computer Science, St Andrews, 2012

**What do you think are the most important aspects of software quality? How would you define what it is?**

I think this is highly dependent on the domain: in some it is correctness (e.g. banking, medical), in others it could be speed, availability, and latency (e.g. non-critical web applications, video games). Furthermore, all software has failures, a large part of quality also plays out in how this failure is recovered from and/or communicated to the user.

Either way, I think a user-centric view of software quality is essential. High quality software serves its intended purpose well, and so a general definition is evasive without knowing what the purpose of the software is.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Look for when hiring:

- Being self-critical of the quality of their own code, and thus keen to use tools and techniques to measure and improve software quality.
- Appreciating the value of code review as a strong quality control mechanism and not a formality.
- As I work primarily using strongly typed functional languages, making the best use of the language features to avoid "foot guns" is a key skill.
- Keeping things simple (KISS), particularly for a large and long-lived codebase that will have many different engineers over the lifetime of the project.
- Always keeping the user in mind.

Taught within a degree:

- Key engineering practices and "gotchas" that can be applied broadly (some examples that come to mind: KISS, leaky abstractions, boolean blindness).
- Testing and monitoring practices (testing pyramid, application monitoring and alerting).
- Conducting good and thorough code reviews.
- Property-based testing and fuzz testing.
- The less code you have, the better!

Gained through experience:

- An intuition for code that will cause problems later on.
- Tooling used at particular companies.
- Quality metrics that are important in your domain.

## Rafnas Arathan Kandi

Lead engineer in tech sector

Graduated MSc Advanced Computer Science, St Andrews, 2012

**What do you think are the most important aspects of software quality? How would you define what it is?**

*Software quality ensures that a product fulfills both its functional and non-functional requirements, thereby fostering customer confidence. High-quality software incorporates robust quality processes from the early stages of development.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*I believe a solid foundation in the basics is crucial, including software version control, unit and integration testing, and familiarity with agile processes. Equally important are strong interpersonal and communication skills. While there's always on-the-job learning, mastering these fundamentals makes it significantly easier for engineers to build successful careers.*

*Programming skills are vital, but how engineers collaborate in a team is equally impactful on the quality of the software produced. I understand that university degree programs often include team projects, but it would be beneficial to clearly define primary roles and responsibilities for each team member, such as designer, developer, tester, lead, manager, and integrator. Rotating these roles based on individual preferences and expertise per project can provide valuable early exposure to industry work environments.*

*Integrating agile practices into these team projects is also important. It helps engineers focus on delivering even small, manageable tasks, fostering a sense of ownership and leading to higher quality software products.*

## Ali Khajeh-Hosseini

Co-founder at Infracost Inc.

linkedin.com/in/alikhajeh1

Graduated PhD Computer Science, St Andrews, 2012

**What do you think are the most important aspects of software quality? How would you define what it is?**

*Over the past decade, my perspective on software quality has significantly expanded. Ten years ago, I primarily focused on the functionality and correctness of the software, emphasizing aspects like automated testing and continuous integration. Today, I believe software quality encompasses much more than just correct code and logic. It now includes:*

- ***Reliability**: with the rise of roles such as site reliability engineer, ensuring reliability has become essential. Software often operates in complex cloud environments where unforeseen interactions between components can lead to performance issues or downtime. Users interpret these problems as indicators of low-quality software.*
- ***Usability**: the importance of roles like product designers has grown, extending their responsibilities beyond user interface design to include the overall user experience, particularly in intricate workflows. Users may find the software confusing or frustrating, leading to an inability to complete tasks and thus perceiving the software as low quality, despite the correctness of the code. In the past decade, user expectations have risen, possibly due to shortened attention spans or the high standards set by companies like Apple for user experience.*
- ***Cost**: historically, engineers were limited by system resources such as CPU, memory, disk, and network, necessitating careful consideration of performance. However, the cloud's on-demand provisioning model has introduced a new dynamic, allowing engineers to scale resources as needed, with cost being the primary consideration. While resource constraints have eased, managing costs effectively has become a critical aspect of software quality.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*We now expect software engineers to test their own code and be proficient with automated testing tools and continuous integration. As these tools have become more advanced and user-friendly, the necessity for dedicated software testers or QA engineers, especially in startups, has diminished.*

*The advancements in deployment platforms and tools have also led to a new norm where software is released hundreds or even thousands of times per year. This shift reduces the need for separate teams to spend weeks testing software before each release.*

*Modern software engineers must now consider deployment (DevOps), security (SecOps), and increasingly, cost management (FinOps). The tools and processes for these areas have matured or are maturing, integrating them into the regular engineering workflow.*

*Regarding education, it's essential for degree programs to cover the entire software development lifecycle and how various specialized teams (DevOps, SecOps, FinOps) provide tools and processes that support the engineering teams in creating high-quality software. Case studies could be beneficial, as well as discussions on the industry challenges that led to the emergence of these roles.*

## Samuel Koch

CTO & software consultant at various UK startups: thezensory.com, viasprout.com, youvouch.com

Graduated MSci (Hons) Computer Science, St Andrews, 2014

**What do you think are the most important aspects of software quality? How would you define what it is?**

For me, the base level aspect is correctness, i.e. the code should do what it was required to do.

Making the code clean, scalable and easy to understand bolster the quality of software.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

For production quality code, I believe that teaching attention to detail is very important. Being able to take a brief, understand the requirements and deliver on them, while also understanding what you do not understand and working out how to fill those gaps either by researching or asking fellow team members.

Teaching clean code that is built to scale, that fellow engineers can understand should definitely be taught as much as possible.

However, having worked with startups and scaleups, there was a steep learning curve between the two. Early-stage startups want MVP-style code, i.e. written faster, so they can ship quicker. Often that meant skipping unit testing and writing code that does not scale and is not easy to understand.

I have seen startups fail because they wanted to transition directly from MVP code to scaling code instead of starting again.

So one thing I picked up with experience that is very difficult to teach is how to accommodate business needs with software quality. How fast a business needs to move, when their requirements will change, predicting what bottlenecks they will encounter with non-scalable software. Every business is so different.

## Kevin Lau

Software engineer

**What do you think are the most important aspects of software quality? How would you define what it is?**

1. Correctness: the code should actually do what it was intended to, especially in edge cases.
2. Easy to read and lack of sharp edges: the less time reading code and documentation before you can understand what something is doing, the better. This aspect is a massive bucket: any concerns related to modularity, mutability, and abstractions/naming go here.
3. Easy to write further high-quality code: being able to introduce new code or modify existing code without compromising correctness or readability. This is highly related to code being easy to read, but they're a bit separate since you can have very readable code that has its correctness/readability compromised as soon as someone with less rigor/background knowledge touches it.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- Attention and care for detail—this is a cliché but it does help to have a level of rules lawyership, because small problems compound on each other and make a mess in a surprisingly short amount of time. This is inherent as a personality trait, but having more general expertise from both a degree and job experience help you spot things that are wrong.
- Ability to abstract smaller details into a bigger picture—having a feeling for what smaller things aren't important to communicate either in code or in actual conversation, while not losing track of them. Writing high quality code is often about ensuring that everything is accounted for without needing to check every line of code yourself. This is mostly through job experience—you can get a better idea of what you can hand-wave over time.
- General logic and programming skills—kind of obvious but you do need technical expertise. Being generically faster at reading and writing code lets you spend more time on higher-level concepts and making code better in less obvious ways. If you're not proficient at a language, your mental bandwidth is going to be occupied with getting basic things correct as opposed to readability or maintainability. Having background knowledge about algorithms, programming styles, and design patterns gives you the ability to pattern match and pick better approaches to avoid reinventing the wheel. This is likely best taught through degrees since that's where you can get exposure to a larger variety of concepts, though job experience would help too.

**Further comments**

I'd speculate that apprenticeship is actually a better model for teaching software engineering than the typical classroom or solo project→tech job pipelines. The difficulty in classroom learning and solo projects is often on getting the thing working, with software quality being relatively secondary—once it works, you're basically done.

In larger projects with more than a handful of people involved, getting the thing working is just step 1 and is likely not most of the time or effort spent on it. At the same time, there's very little inherent feedback that code you've written is low-quality, because the thing probably still works regardless of what horrific monstrosity you've made—knowing what's better or worse really requires either experience or, more likely, someone to just tell you. Code reviewing along with vetted shared resources are the most effective ways to get feedback across about code quality.

## Rhona McCracken

Software engineer in tech sector

Graduated MSci (Hons) Computer Science, St Andrews, 2023

**What do you think are the most important aspects of software quality? How would you define what it is?**

*I recently came across the term "software craftsmanship", and I believe it perfectly encapsulates the mindset engineers should adopt for producing high-quality software. Unlike factory workers who perform repetitive tasks, software engineers should focus on quality at both the granular and system-wide levels.*

*Quality in software isn't just about immediate correctness; it's about the system's resilience to various changes—whether in traffic and load, requirements, or team members. Engineers must ensure that the system is easy to test, reliable under scaling and fault conditions, comprehensible for new developers, and adaptable to evolving requirements. Balancing these factors becomes increasingly complex as software grows.*

*To maintain high software quality, two critical strategies stand out. First, it's essential to allocate time within daily workflows to address technical health and manage technical debt. Second, establishing and adhering to best practices and standards within the team or organization is crucial. This includes approaches to testing, language feature usage, and architectural patterns. These measures foster a culture of excellence and sustainability in software development.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*Certain aspects of software engineering are best learned through experience, such as understanding your company's quality standards, the reasons behind them, and knowing when to propose updates to these standards. However, other practices should be ingrained from the very beginning of a software engineer's career. This includes consistently covering code changes with unit tests (and revising the code if testing is too challenging), considering other testing types like integration tests for larger systems, load testing, and code style checks. It's also important to understand classic coding design patterns and architectural patterns and to always think about scalability and fault tolerance in system design.*

*Another often overlooked aspect in university courses is the importance of metrics for monitoring system behavior. Learning how to create effective alerts and graphs to track system errors can be invaluable. These concepts can be challenging to teach practically in a classroom setting. It would be beneficial if students could work with a large codebase that employs CI/CD, allowing them to experience maintaining a big service firsthand. This could involve adding new features, testing them, and observing an architecture in action.*

*Interacting with such a project would also highlight the importance of version control, making small and meaningful pull requests for peer review, and deploying services while monitoring metrics to detect and address any issues.*

**Further comments**

*While I'm uncertain if it's feasible, I've heard that some other courses offer brief industrial placements as part of their masters programmes. It would be highly beneficial for students to apply their learning in a real-world setting through such a placement, perhaps lasting about six weeks. This hands-on experience would not only solidify their skills but also enhance their CVs.*

## Angus Macdonald

Staff software engineer at Google

linkedin.com/in/angusdmacdonald/

Graduated PhD Computer Science, St Andrews, 2011

**What do you think are the most important aspects of software quality? How would you define what it is?**

I've now worked as a software engineer for over a decade and though the scope of what I've been working on has gradually scaled up, from functions and classes to inter-connected complex systems, my definition of quality has remained roughly the same.

I'd summarize it as predictability and understandability, or more generally how easy it is to understand what's happening and what might be going wrong.

You know a quality piece of software when you're confident in what it's doing, and able to quickly identify and fix bugs.

At the micro level most people have a pretty good sense of what this means. Functions with limited side-effects and clearly delineated logic make it easy to read code that you might not have written (or remember writing).

The same remains largely true of more complex systems. There are services you can be confident in maintaining, and those that you're scared to touch in case it causes some unexpected issue that wouldn't be discovered until a downstream system notices a problem.

Along similar lines, there are those that have enough logging and tooling to quickly get to the root of the problem, and there are those that require asking the person with 10 years of experience and a set of obscure SQL scripts.

Ultimately each of these examples boils down to the same observation: quality is measured in how easy it is to peer inside and understand what's happening.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

When hiring it's typically easier to look for proxy metrics on the more function-level aspects of quality. I think getting an insight into how someone refactors code can give a sense of what they see and how they view quality. Some undergrads have an intuitive understanding of this, but it's not often necessary because the code you write rarely lasts longer than the next practical grading.

Complex systems are harder, both for hiring and undergraduate experience. This typically comes with time, and often painful learned experiences (e.g. that system that always woke you up at 3am on an on-call rotation), but there are lots of good books out there nowadays (e.g. SRE books) that give some background motivation.

I think having a good understanding of function-level quality is a good proxy for understanding system level quality, but there are limits to how much that can truly be internalized and there are differences (e.g. system level tooling is intrinsically different).

By far the most useful experience of my undergrad degree were the team and individual software projects, which gave by far my closest experience of dealing with "real world" quality issues.

## Iain McDonald

Director of engineering in tech sector

Graduated PhD Computer Science, St Andrews, 2004

**What do you think are the most important aspects of software quality? How would you define what it is?**

*Understanding the needs of software users and the infrastructure required to handle their traffic patterns is crucial. For startups introducing novel ideas, often managed by small teams of one or two engineers, users may accept some level of imperfection or incompleteness. However, in B2B platforms serving top-tier enterprises or competitive B2C markets with numerous alternatives, expectations are higher due to the investment or alternative options available.*

*Software engineers must grasp user expectations to gauge the necessary quality level for implementation. This becomes pivotal in gaining market traction, where speed of delivery itself constitutes a key quality metric. Teams that overly prioritize reliability or optimization prematurely risk being surpassed.*

*Scalability is intertwined with user expectations. While a system might perfectly meet the needs of a single user, it may falter under the load of a million concurrent users.*

*To define success in meeting user expectations at scale, both product and operational metrics are essential. Product metrics such as acquisition, activation, revenue, and churn rates gauge user engagement and satisfaction. Operational metrics such as service level objectives and indicators verify whether the system can sustainably handle the required scale.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*A strong foundation in core software engineering skills—such as programming proficiency, knowledge of algorithmic complexity and data structures, web fundamentals, and effective communication—is typically expected of computer science graduates. However, these skills alone do not encompass important personal attributes like enthusiasm for the field and a readiness to learn.*

*Many critical software engineering competencies are developed through collaborative team efforts to solve large-scale user problems. This creates a challenge for degree programs, which primarily assess individuals rather than teams. Consequently, these programs have limited time for extensive project work, team organization, and exposure to real-world applications.*

*While essential software engineering skills require practical experience, a four-year degree provides ample opportunity to gain proficiency in programming and data structures. During interviews, we evaluate candidates based on these expected skills, but any additional experience is viewed as a valuable asset.*

*For instance, due to the constraints of evaluating individual students, I would be pleasantly surprised to encounter experience in areas such as delivering user-validated software using product analytics, effectively resolving interpersonal challenges within team dynamics, demonstrating proficiency in delivery automation (e.g., infrastructure as code, automated testing in deployment pipelines), designing distributed cloud systems with reliability, security, and scalability guarantees, and implementing comprehensive operational observability (e.g., automated alerting, cloud telemetry, service metrics, logs).*

## Paul Normington

Software engineer at Udemy

linkedin.com/in/paul-normington/

Graduated BSc (Hons) Computational Science, St Andrews, 1988

**What do you think are the most important aspects of software quality? How would you define what it is?**

The most important aspect is ensuring the software behaves as expected, and being able to verify former behaviours still hold true after modifications have been made.

As we move to a world of AI-assisted software creation and maintenance, this becomes even more important, as the quality metrics of legibility, complexity or avoidance of duplication are probably less concerning to AI. Humans will spend less time looking at code, but more time defining behaviour.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Testing ... at both unit and acceptance testing levels. Looking to the future of software development, acceptance testing (ensuring the software behaves as a system in the ways intended and not in the ways not intended) becomes the mainstay of testing.

Understanding how to represent, implement and execute tests, and how to integrate that testing into the continuous integration steps are really important.

I like people to have an opinion on code coverage, and be able to argue their opinion ... but I would be concerned if anyone said that 100% code coverage was necessary or significantly more effective than 90%, say.

The measurability of quality is interesting

- code coverage
- defect discovery rate
- mean time to resolve
- vulnerability counts

although the right target values depend on the purpose of the software, the regulatory landscape as well as the practical cost of repair (or not being able to) when a defect is found in production.

**Further comments**

Too early to see if Gen-AI will change the approach to software quality in the long run, but I think it will eventually cause us to look at the minutiae of code far less .. and that has been where quality initiatives started.

But the bigger picture of quality will be around system behaviour, and our prompts to AI coding bots will be all around requirements. Those prompts will be both the definition of the behaviour of the system as well as the rules that will be executed to test the system.

## Adrian O'Lenskie

Director of engineering in tech sector

**What do you think are the most important aspects of software quality? How would you define what it is?**

I'll break this down into quality of the software code and the quality of the software product.

For the former, idioms around coding style, design patterns, lint rules, unit test requirements etc., all look to maintain a certain degree of quality in the code. Maintaining a quality level in the code is critically important for maintainability (without it, the quality of the future versions of the software product is likely to diminish).

I view the quality of the software product along two dimensions. Does the software application conform to specification and does the software application feel well-designed/produced? For the first of these, we tend to look at defining functional and non-functional requirements (and even if they are not always defined, we can expect them to be implied). For example, the functional requirement might be to display your current bank balance. There can be an implied non-functional requirement that this completes within a short time window before you as a customer get frustrated.

The second dimension is around how well the application received by user stakeholders. When Trello was initially released, I was delighted by the somewhat gratuitous ability to re-order through drag and drop (not really the feature itself, but the *feel* of the feature). It added a different dimension to the quality feel of the product. Another important aspect of the quality of a software product is intrinsically related to design. An article that better describes this: https://www.spectator.co.uk/article/why-the-greatest-innovations-do-only-one-thing-but-do-it-well/

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

I would look for individuals who can talk intelligently about the distinction between unit testing, integration testing, load/stress testing, fuzz testing, chaos testing, end-to-end testing, A/B testing, canary testing, gradient switch-overs, tool-chain testing and so on. They each have a particular purpose/scope and each of these domains come with supporting infrastructure and tools (that change daily). Automation Automation Automation. This is critical.

I think it would be useful if (in a degree program) a student would gain an intuition around why testing is critically important during the development of a large scale application, and why it is equally important in the evolution of software applications where the engineering contributors change over time.

I do not think we spend enough time (as an industry) working on the quality of our test designs. For example, test coverage is a universal measure but we do not seem to focus enough on test efficacy (how many bugs are prevented through failing tests over time). Another example would be the overlap between unit tests and different testing scopes. If your end-to-end tests are replicating the tests you are running in integration tests, you are not gaining anything beyond false comfort (and every test **costs**).

Finally, I would also suggest that it is important to track trends over time. For example, I might be asked to deliver a feature that shows your bank account when you log in. It might be perfectly specified, the accurate balance must be displayed to you within a particular widget in 30ms. I achieve all that, and QE sign off on it. However, in achieving this, I need to pull in another library, I load that DLL at the ingress point for a particular path—I've just changed the performance envelope for all calls along that path. Even if the requirements across all the other functionality haven't been breached, it is important to be able to identify this degradation in aggregate. Your feature operates as a shared tenant in the overall application context.

## Folarin Omotoriogun

Engineer at Google

linkedin.com/in/folarinomotoriogun

Graduated MSc Software Engineering, St Andrews, 2014

**What do you think are the most important aspects of software quality? How would you define what it is?**

The most important parts of software quality in the context of programming and software engineering are a balance of the following:

1. It is objectively readable by others: is the code self-documenting? Is the software written in such a way that other engineers can easily understand the code? If only a few individuals can understand the code, then it becomes unmaintainable at scale and this is considered bad code. Following consistent best practices, good naming conventions and in-line comments are great ways to solve for this.
2. Does the software meet performance requirements: does the software meet business or user performance requirements? This one is obvious, but sometimes these issues do crop up at scale.
3. Does the software include guardrails that prevent misuse: there are a host of important factors one can throw in this bucket and it will vary from industry to industry. In a financial application, adequate security guardrails will be an important measure of quality, while for a peer to peer photo sharing app, privacy will be probably more important.
4. Is it reviewed: enforcing the above requires the use of tools to assess the code quality but also having a human reviewer assess the code for quality issues. Use of version control and static analysis tools are a few ways to do this.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

We will generally look for candidates that:

5. Use industry wide code best practices for target languages: companies will always have their own best practices but it is always great if a candidate displays a good knowledge of broadly known best practices. This can be evaluated by their code quality and their participating in open source projects. This should be taught and assessed in school in the context of collaborative software development and peer reviews.
6. Awareness for when technical debt is acceptable and how to deal with it (ensuring it doesn't linger forever) should also be taught in school. For example, using TODO code comments in code as a way to signal to oneself and others limitations of an implementation is great. This skill would come through practical experience by nature.
7. Software drives a majority of the world's processes. An ability to assess the non-functional requirements of software such as what level of security is acceptable for the software is a clear signal of a robust candidate that understands software quality in real world context.

## Nathan Owens

Software engineer in tech sector

Graduated BSc (Hons) Computer Science, St Andrews, 2015

**What do you think are the most important aspects of software quality? How would you define what it is?**

- As simple as possible but not more simple.
- Easy to read, avoid being clever.
- Eventually running tests for critical sections.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- Working with existing large code bases.
- Contributing to open source code.
- Knowing your way around a Linux CLI.
- Knowing how to use Git.
- Debugging skills.

## Daniel Patterson

Software engineer in tech sector

Graduated BSc (Hons) Computer Science, St Andrews, 2014

**What do you think are the most important aspects of software quality? How would you define what it is?**

- How easy it is to change existing software to add new functionality without breaking existing functionality—i.e. maintaining high reliability without sacrificing velocity.
- How long it takes to make a change to a software system, measured from when the requirements are first given to the development team until the feature is generally available to users.
- How afraid developers are of making changes to a particular system.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- Strong knowledge of automated testing, with an understanding of the trade offs between test coverage, development time, and CI speed/reliability.
- The ability to define good Service Level Objectives and Agreements, and how to use them to determine reasonable error budgets (these could be taught in a degree).
- The importance of Continuous Deployment and how to use canary releases, experiments, and feature flags to maintain both high velocity and high reliability.
- Ability to debug from multiple different angles: reading logs, testing in production, writing automated tests, querying real-world data, adding and analysing metrics in running systems (many of these are much easier to learn through experience).

**Further comments**

I think that software quality is a property of the system as a whole—both the software and the organisation building that software. I strongly believe software development is a symmathesy, meaning a learning system where each of the parts is learning from the other (see https://medium.com/@jessitron/symmathecist-n-c728957ce71f and https://medium.com/the-composition/the-origins-of-opera-and-the-future-of-programming-bcdaf8fbe960).

I think software quality is heavily related to the culture of the organisation and the perception of the people involved. If most developers are afraid of making a change, the cost of changes becomes high, they make the smallest change possible, and likely lose knowledge on how to test their change.

For example, imagine a software system that had been developed by one team, then handed over to another team for them to improve. The new team has no access to the old team. At the point of switchover of ownership, despite the fact that the code has not changed, I would say that the software quality has significantly decreased as the organisation as a whole cannot make changes as quickly or reliably as it previously could. I have seen this time and again in my career, as projects are shifted between teams or handed over from contractors.

## Douglas Pearson

Co-founder and CTO in online gaming sector

Graduated BSc (Hons) Computational Science & Mathematics, St Andrews, 1988

**What do you think are the most important aspects of software quality? How would you define what it is?**

Over the years I've come to believe that the primary skill for "good software quality" is actually "good software design". Good designers build good software systems.

And by "design" here I'm including everything from software system design (overall architecture) down to module design (class structure) and individual class design (methods and fields). Each needs to be "correctly designed" to my mind.

OK, so then what's good design?

I would say there are some critical elements in good software design:

a) The design should be clearly understandable

b) The design should make it easy to modify the software in ways we want to allow modification

c) The design should make it hard to modify the software in ways we don't want to allow

The first two are pretty obvious and what most people consider in good design, although they may forget how important (a) is—that your design (your software) needs to be something other people can easily understand. This "understandability" concept naturally leads to more specific choices —like keeping things simple. It's not because simple is really fundamentally better (maybe it's slower to write or execute)—it's that simple is easier to understand.

The third is more controversial, but actually I think is a key element. E.g. if we look at statically typed languages (e.g. Java, C++) the compiler is actually implementing exactly this concept—that we don't want to allow the wrong data type to be passed to a method. This constraint is actually a positive and a reason to (sometimes) prefer these languages over dynamically typed ones (e.g. Javascript). But what's true at the language level is also true in the software design itself. E.g. we don't want to allow all types of data modification (immutable types often being preferred) or maybe we don't want to allow certain types of changes (post office sales records come to mind as an example...)

Anyway, I think those are solid principles for good design and good design leads naturally to high software quality. Poorly designed systems cannot produce high quality software—no matter the skill of the programmer.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

I often ask people to explain a complex piece of software to me that they have built before. As you can imagine, I'm going after the design—did they understand the overall design, can they explain it, did it make sense?

Then when it's time to look at some code, the skills there are mostly around how the code looks on the page. If you look at a class, is it clear what the class is intended to do, are the methods small and focused (e.g. each method doing one thing without additional side effects).

From a teaching perspective—this is focusing very much on the "how" something is built and less on "what" is being built. You can build an operating system as a great sample project at a university. But I'm less concerned that you know how an operating system works (something I can probably teach you in a few days if you need to build a part of it), but if you don't know how to implement those ideas "well", I can't teach you that. I need you to already have learned that in college.

So a strong focus on looking at the details of what choices are being made in development. Not just in passing a set of tests and meeting criteria. That sort of knowledge is much easier to learn on the job. The key is learning about these good "core values" for how to design and build high quality software systems.

**Further comments**

When explaining this sort of thing to my non-techie friends, I always find the analogy of writing quite useful.

By 10 years old, children in US and Britain can all read and write English perfectly.

But the skills needed to write a good novel, no 10 year old has those. And it's that skill that we are really looking for in a "good software engineer". The ability to write a great novel. The senior architects are maybe laying out the chapters and the junior engineers are writing paragraphs, but it all needs to be part of a complete and unified single story. Otherwise it'll be a pile of junk that doesn't work right and nobody wants to maintain.

## Indika Perera

Professor in computer science and engineering

Graduated PhD Computer Science, St Andrews, 2013

**What do you think are the most important aspects of software quality? How would you define what it is?**

- ***Testability**: software should be testable with confidence to enhance its quality.*
- ***Observability**: software needs to expose its internals adequately for testing.*
- ***Cost-effectiveness**.*
- ***Relevance**: ensuring that the quality assessment aligns with the software's objectives.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- *Creating models and abstractions to develop appropriate quality frameworks for specific scenarios.*
- *Skills in automated testing (including programming and tool utilization).*
- *Skills in evaluating tests (which can also be acquired through hands-on experience).*

**Further comments**

*Modern software quality considerations are swiftly adapting to changes across the software lifecycle. For instance, contemporary systems largely rely on no-code or glue-code approaches, shifting much of the quality and testing efforts towards black-box modules. Additionally, the adoption of CI/CD pipelines and DevOps has transformed testing and quality assurance into regression-based processes rather than traditional testing methods. Service-oriented utility models also necessitate novel approaches to evaluating service quality. Furthermore, the integration of AI introduces entirely new methodologies for ensuring quality, such as validating ML/DL models.*

## Mikhail Piankov

Software engineer in tech sector

**What do you think are the most important aspects of software quality? How would you define what it is?**

It's an interesting question and it is slightly different in different contexts. From a developer who supports it perspective, the most important should be "how easy to support", but from developer who uses it (library or service)—probably the best thing is lack of the bugs and clearness of interfaces.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Ability to write clear code; it includes defining the interfaces; naming of the methods; keep methods small and simple; and following code style.

## Yulia Samoylova

Solution consultant in consulting sector

Graduated Erasmus Mundus MSc Dependable Software Systems, St Andrews, 2014

**What do you think are the most important aspects of software quality? How would you define what it is?**

Reliability, easy maintainability, good performance, easy to debug, structured logs.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

From university: logic mindset, scripting fundamentals, modern stack technology understanding, understanding bigger picture of the product.

From work experience: deploying and maintaining software, understanding industry.

## Lakshitha de Silva

Staff software engineer in online gaming sector

Software architect in various industry domains

Graduated PhD Computer Science, St Andrews, 2014

**What do you think are the most important aspects of software quality? How would you define what it is?**

1. The software needs to conforms to its user requirements. The functionality that the software provides should match what the end-user has specified. This means that software's design should also cater to the user requirements.
2. The software should be reliable and usable even when its operating environment changes within reasonable limits. In the event the software is unable to operate, it should shutdown gracefully and without damaging the integrity of its stored data.
3. The software should behave in a predictable and consistent manner when it encounters different inputs or when its operating environment changes.
4. The software should be easy to use to different users. The user should be able to interact and use the software without have to go through an unreasonably steep learning curve.
5. The design or the code level implementation of the software should not be more complex than it should be. Any complex areas of software should be sufficiently documented and isolated from other parts where possible.
6. The software should be testable and amenable to software testing tools.
7. The software should have the means to be monitored in production environments, and to be debugged easily when problems occur.
8. The software should lend itself to be maintainable, extensible and evolvable with small localised changes.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

1. A strong understanding of core software engineering principles such as SOLID, OOP, reuse, etc, rather than delving too much into the latest technology.
2. A strong understanding of: how to write simple, clean code, keeping complexity to minimum both at design and code level, and writing testable code.
3. Able to see the big picture around the software they are building and why, instead of just the small areas assigned to them, so they have a good understanding of the user requirements.

## Neil Skilling

Consultant in enterprise tech sector

linkedin.com/in/neilskilling/

**What do you think are the most important aspects of software quality? How would you define what it is?**

The usual metric that I have used is the number of known defects in production software. That metric can be used throughout the development lifecycle so that defects are being flushed as early as possible.

My view has probably changed over the years and that the importance of the traceability of a feature from its use case, through deliverable and test cases to delivered software is something that is closely correlated with the number of defects.

My experience has also told me that investment in test driven development, automated test coverage as close to 100% as possible and dev ops practices that enforce those gates in all development stages really improves the software quality.

Commits should not be allowed if test coverage has dropped or existing tests are failing and should be enforced with source code repository hooks.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Test driven development was one of the core skills that I looked for and the way that these can be mapped from the user stories and the importance of that to the finished product quality.

I think that a little more emphasis could be applied to test case generation. Often when complex data models are critical to the correct operation of the system the creation of realistic data scenarios is often overlooked. Test cases are just copied and obfuscated from production and not generated in a systematic fashion that helps with coverage testing.

## Ashley Sole

Director of engineering in tech sector

Graduated BSc (Hons) Computer Science, St Andrews, 2008

**What do you think are the most important aspects of software quality? How would you define what it is?**

For me, it's all about the end-user experience. Does the software do what the user expects? Tests should then be written in such a way that validates the user expectations.

As an example, software is not necessarily good because the tests all pass. It's also the case that just because tests fail, doesn't mean software is bad. In order for tests to be effective, they must accurately check how the user interacts with it.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- Ability to write tests, and think of different test scenarios.
- Ability to take a real-world case and translate this into the different aspects that need to be tested.

I'd say, straight out of university, many new employees lack the understanding that we are dealing with real customers, that have real bugs that cost the business money, or reputational damage. I think one thing that can be taught at university is very much thinking about the end user and translating their needs into checks that assess the quality.

As an example, let's take a website that takes 10 seconds to load. Perhaps functionally everything works, and all the tests pass. But this is a poor user experience and not a sign of good quality software.

**Further comments**

A lot of what we do in industry relates more to automation. Things like, testing in production, synthetics, blue green metrics based deployments, and so on. The way I see the industry going more and more is reducing flaky test environments, removing flaky integration tests, and just getting new code out in front of small subsets of customers so they can validate the functionality through real customer behaviour.

## Thomas Strang

Senior software engineer in cyber security sector

Graduated BSc (Hons) Computer Science, St Andrews, 2018

**What do you think are the most important aspects of software quality? How would you define what it is?**

I think there are lots of things that good quality software should be, but I consider the most important aspect of good quality software to be readability. Code that is readable but has bugs, can be fixed. Code that is readable, but is inefficient, can be improved. Code that is readable, but isn't required, can be easily deleted. Non readable code actually acts like an infection: later developers will struggle to understand it and so will be more likely to make poor decisions when extending or modifying it, making the code even more unreadable. Unreadable code is hard to deprecate as you need to first understand its side effects before deleting it.

But what makes code readable? Code that is easy to read and understand is really just a byproduct of other aspects of quality software:

- **Modular**: modular code is split up so that you can reuse an individual component without having to reuse the rest. This is the no.1 ingredient to readable code. If each individual component has a single non overlapping task / responsibility / domain, this allows a new developer to understand an individual component without having to first understand the entire system.
- **Extendable**: extendable code is written with consideration as to how it might be extended later on, without having to be refactored. This can often require adding additional complexity initially, but over time, as other people extend the code, they will be able to do so in a fashion that minimises complexity creep.
- **Minimal**: if there is a simple way to do something, and the performance cost to doing it the simple way will never be relevant, then it should be done the simple way. An example of this might be choosing a simple, $N^2$ complexity solution over an unintuitive Nlog(N) complexity solution, if you will never run your code on a large input. It can sometimes be a balancing act between writing extendable and minimal code.
- **Standard**: adhering to some form of code standard is usually quite easy, and has no drawbacks. A notable example might be consistent naming: if an object of type *ConnectionDistributor* is sometimes referred to as *conn_distr*, and sometimes referred to as *connection_distributor*, developers might not immediately realise that they are the same type. Calling a function *getRecordDB* might be a poor name if the function creates the record if it doesn't already exist in the database.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

When we perform system design interviews the intention is to see people's ability to determine / gather requirements from a vague problem statement, and split up problems into components. I would consider that an excellent addition to a degree.

When we perform coding challenges with candidates the intention is obviously to see people's abilities to solve problems in a given language, but also their ability to write code that is readable.

I believe that knowing when to make a minimal change, or write a more complex but possibly more extendable solution, does require experience. It is always tempting to gravitate towards the most extendable solution, but sometimes the minimal one is best, in particular when a deadline is due!

## Daniel Swarup

Software development engineer at Amazon

linkedin.com/in/daniel-swarup/

**What do you think are the most important aspects of software quality? How would you define what it is?**

*The cornerstone of software quality lies in testing and maintaining clean code. Testing guarantees functionality and reliability, while clean code—such as adhering to DRY principles—enhances readability and ease of maintenance. Ultimately, high-quality software exhibits functionality, reliability, usability, performance, maintainability, and security.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*When recruiting, my focus would be on candidates who demonstrate expertise in clean code practices, understanding of performance and security factors, and strong problem-solving capabilities. While these skills are teachable in academic programs, hands-on experience is invaluable.*

*Real-world projects, internships, and collaborative development settings offer opportunities for candidates to acquire and refine these skills. Moreover, staying current with industry advancements and continuously expanding knowledge of new technologies and methodologies are essential for thriving in software development positions.*

## Bence Szabó

Engineering manager

Graduated BSc (Hons) Computer Science, St Andrews, 2016

**What do you think are the most important aspects of software quality? How would you define what it is?**

*I consider maintainability the most critical aspect of software quality. This involves adhering to clean code principles, as detailed in Robert C. Martin's Clean Code (e-book), and fundamental principles of object-oriented programming.*

*I prioritize simplicity and code readability over adopting the latest technologies. For instance, in Java, excessive use of streams can compromise code readability and understandability.*

*I believe that well-written code should be self-explanatory and easily testable. I strongly recommend developers follow SOLID principles, as they typically lead to clearer code.*

*However, ensuring that the code performs its intended function is paramount. Achieving this requires robust unit testing, thorough specifications, and effective communication with business stakeholders.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*Experience is crucial. Finding the right equilibrium between coding and thoughtful consideration comes with exposure to diverse codebases.*

*Teachable skills include applying SOLID principles and adhering to clean code practices. It's crucial for students to grasp a fundamental truth in coding: Investing in quality code upfront saves resources in the long run.*

*Additionally, I would emphasize the skill of comprehending and modifying legacy code. This can be cultivated by assigning tasks where students work with old codebases to identify bugs, using provided bug reports. Simulating such real-world scenarios prepares them effectively for roles in the industry.*

## Valentin Tunev

DevOps engineer in tech sector

Graduated BSc (Hons) Computer Science, St Andrews, 2015

**What do you think are the most important aspects of software quality? How would you define what it is?**

*In no specific sequence: performance, reliability, security, and user experience. From my perspective, compromising any of these elements impacts the overall software quality. Defining quality is challenging for me because it heavily depends on context. A software piece might be deemed excellent in one context but inadequate in another. For instance, comparing performance expectations between a web application and aviation software illustrates this variability.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*Firstly, understanding the concept of software quality is crucial. Currently, I observe a lack of formal definition, with software quality often being subjective rather than based on established principles and practices.*

*Secondly, ensuring and maintaining software quality requires proficiency in testing, validation, observability, logging, and other essential practices.*

*Lastly, like all aspects of software development, there is a philosophical component involving the balance of key factors mentioned earlier. A common industry challenge I notice is either over-engineering systems, creating unnecessary complexity, or under-engineering, resulting in insufficiently robust solutions. For example, some web applications are built to scale as if they were serving thousands of users like Amazon or Google, despite only having a small user base. Conversely, minimal viable products are sometimes rushed into production without adequate testing.*

*I advocate for formalizing and teaching the foundational principles of software quality, including establishing a clear definition, best practices, and recommended patterns. Practical experience should then complement this theoretical knowledge. Achieving a nuanced understanding of balancing software quality aspects typically comes with experience, but it can be introduced and discussed in formal education to raise awareness among students.*

**Further comments**

*I think there's a critical need for formal education focused on software quality in the industry. Software development has started to resemble fast fashion, where there's rapid production but often at the expense of quality. It's surprising to see advancements in hardware and software development, yet overall software quality seems to be declining steadily.*

*In my view, this decline may partly stem from trends like resumé-driven development, the proliferation of coding bootcamps, and the increasing ease and accessibility of writing and deploying code. With these advancements comes the responsibility to ensure that software maintains high standards of quality and reliability.*

## Robbie Wallace

Software engineer in tech sector

Graduated MSci (Hons) Computer Science, St Andrews, 2023

**What do you think are the most important aspects of software quality? How would you define what it is?**

- Readability: as a dev we spend far more time reading existing code, trying to determine what caused a given issue. The more readable code is the easier it is to do this.
- One thing I think is hugely important in code is the layout. Striving for the locality of logic is important. Having to move between many files and directions makes it a lot harder to figure out what is going on.
- Minimise mutability, when opening a function if it is immediately obvious what variables are mutable and what are not, it's a lot easier to see what the function is trying to achieve.
- Logging: following on from this logging is hugely important in debugging issues and following the execution through code.
- Documentation: though not much code is documented in my experience, documenting APIs / interfaces is extremely useful. Documenting individual internal functions is less so. For documenting code not interfaces it is only necessary to explain why something is being done if it is not immediately obvious.
- Runnability: if a service can only be run in UAT or PROD it makes it significantly harder to debug and implement new features. Every dev should be able to spin up an instance of the prod environment or something akin to it, on their local machine.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

- Openness to learn new ideas and a natural curiosity about how software works.
- There are many good practices which are far better learnt than taught. It's hard to understand why good code is good if you haven't experienced the pain of bad code.
- Dependency injection and composition over inheritance. I do think I was taught a very OO approach to software engineering. OO while great can be quite dangerous when overused (in that it leads to hard to understand code).
- An understanding that at the end of the day, the code runs on a real machine. Whilst abstractions can be great, hiding behind them can lead to poor performant code.

**Further comments**

One thing I have heard at work which I think is important is "strong beliefs, but weakly held". In software engineering, there are several choices available and at the end of the day, you need to pick one. Having strong beliefs makes that easy. However, when presented with empirical evidence, I think one should be open to radically changing the way they write software.

## Neil Wells

Graduated BSc (Hons) Computer Science, St Andrews, 2015

**What do you think are the most important aspects of software quality? How would you define what it is?**

Readability/debuggability. You can have the fastest code in the world, but if I can't fix it when it breaks then I can't support it.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

An ability to deliver not just software that meets the specifications, but software that is (in no particular order):

- Well tested (unit and integration). I don't care so much about coverage numbers, but all complex algorithms and interactions between different parts of the software must be tested.
- Fault-tolerant. Faults happen. You can do what C++ does and crash the whole program with an unreadable error message. Or you can do what every other language does and return a nice error while progressing with unaffected work. Not that I'm biased...
- The right level of complexity. This is a hard one. Code should be as simple as possible while also having enough structure and abstractions to make it understandable and updateable.
- Readable. Unreadable code is un-debuggable and un-extendable and therefore un-useful.
- Amenable to introspection. Logging, metrics (yay Prometheus), and tracing are very valuable for debugging in production.
- Reconfigurable. At my company's scale building, testing, and pushing a new binary to fix an issue takes a long time. The ability to change behaviour by flipping a flag or changing config is invaluable. For example, if there's an issue with a new feature I want to be able to turn the feature off.

These all need to be internalised through experience, but I think it's definitely possible to design a course that gives some of this experience through practical exercises.

**Further comments**

There's nothing better than practical experience. I wonder if you could get students to try adding a feature to a reasonably complex piece of code that they wrote 6+ months ago. This would teach them that the code they thought was good when they wrote it isn't as readable and extensible as they thought.

## Titus Winters

Senior principal scientist in tech sector

**What do you think are the most important aspects of software quality? How would you define what it is?**

It's a little bit meta, but for me the most important form of software quality is "sustainability"—for the expected lifespan of our software, we need the capability of responding to changing technical or business requirements. If we have that in place, a lot of other qualities can potentially be improved (security, privacy, efficiency, etc). And there are a lot of potential ways to achieve sustainability, although in general it comes down to good overall engineering rigor/practice—consistent coding style, consistent use of infrastructure and utilities, good focus on automated testing, documentation, etc.

Interestingly, in a single-shot development environment, the Iron Triangle of project management applies, and product managers (especially) may view software quality as being in tension with meeting feature delivery deadlines and other customer obligations. But in a multi-round development, there is no evidence that software quality is in tension with product quality—specifically because a malleable and high-quality codebase is easier to shape to provide new features over time. (There is a comparison to be seen here in game theory: One-shot Prisoner's Dilemma has a different optimal strategy than Iterated Prisoner's Dilemma. Many of our prioritizations in tech still pretend the old shrink-wrapped single-delivery process applies.)

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

I think it's really hard to authentically convey the lessons of software quality and rigor in the context of a classroom. A lot of things (automated testing) have substantially better payoff over time, and in the face of refactoring/rearchitecting or updates to infrastructure (compiler version, language version, etc). Those activities just don't happen in an undergraduate experience—it's hard to build an authentic scenario where success or failure hinges on a long-term investment, especially when course durations are only a couple months.

I think a lot of these things should be introduced, conceptually. And in a few cases, we should give students some ability to practice with these tools/techniques. But again, it needs to be done carefully—even things like "teamwork" in the context of an undergraduate experience are somewhat fraught. No real team is made up of people with nearly identical backgrounds and similar experience levels—throwing a bunch of undergrads together and asking them to work together runs the risk of giving the wrong impression of what "teamwork" is like.

In a perfect world we'd be using internships and coops for the hands-on experience and focus classrooms on explaining the theory behind engineering practice. In the US, most programs have at most 1 (and often 0) such activity, so instructors have the challenge of trying to introduce everything, even when labs and lecture aren't a terribly authentic/effective approach.

I do think we can do better at some of the very basic practices that lead to quality: unit testing, code review, deeper understanding of the difference between static analysis and dynamic analysis, use of IDEs and debuggers, etc. For code review in particular, I've been advocating for assignments around code review being treated like an English essay: we are grading you on your communication skills as well as your attention to detail. Spotting bugs is valuable (but not the main point). Spotting quality issues is critical, but loses value if a reviewer cannot highlight the issue without upsetting the code author.

## Evangelos Zirintsis

Graduated PhD Computer Science, St Andrews, 2001

**What do you think are the most important aspects of software quality? How would you define what it is?**

*Software quality pertains to how effectively a software product aligns with its requirements and seamlessly fulfills the users' needs. This entails the software meeting customer expectations precisely and being free of bugs, ensuring consistent functionality.*

*Thus, achieving a quality software product necessitates thorough analysis of requirements and rigorous testing of the final deliverable.*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

1. *A solid grasp of the business rationale behind the software development.*
2. *Analytical abilities to translate requirements into well-structured modules.*
3. *Proficient design capabilities.*
4. *Methodical coding practices.*

*While the latter three skills can be effectively taught in a course (and further refined through work experience), the first skill necessitates on-the-job experience to comprehend customer requirements and develop business-oriented software effectively.*

**Further comments**

*Similar to any course, it should include both theoretical knowledge and practical application. I advocate for emphasizing real-life projects to demonstrate the benefits of methodologies and the consequences of not adhering to them. This approach will underscore the essential nature of software quality.*

## Name Withheld

**What do you think are the most important aspects of software quality? How would you define what it is?**

I see it as a min(correctness, user-friendliness, developer friendliness), or maybe weighted average of the three.

Correctness: bugs are inevitable, but quality software fails less often, with less severe consequences.

User-friendliness: quality software is responsive, easy to use (intuitive), most used features can be accessed quickly, etc.

Developer friendliness: quality software is easy to maintain and extend. This means code is easy to navigate, it's easy to get a high-level understanding (not necessarily low-level), easy to make modifications without fear of breaking it (code either trivial or well covered with tests).

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Correctness: this is already being taught within a degree? Exams are pretty strict, I think. User-friendliness: can be taught. I guess most of it falls under design, but there are some basic principles that can be memorized and applied. I wish I was taught this, or at least been made aware of its importance.

Developer friendliness: the best way to learn this is to teach the theory first, but then have a project, develop anything really that can be done in a month, should be big but not difficult. Maybe work in groups. Then extend the requirements. Then extend them a few more times. At some point shuffle the code ownership between groups. In my experience just simply learning the code design principles for an exam means almost nothing until you actually need to maintain 1) your own code and 2) someone else's. Even though I didn't forget the principles, I still failed to apply them correctly when I started working, meaning I didn't truly understand them, they were too abstract.

## Name Withheld

**What do you think are the most important aspects of software quality? How would you define what it is?**

Software quality is a combination of characteristics that a piece of software exhibits including but not exclusively extensibility, maintainability, documentation, research & design documentation and any requirements-related documentation.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

I would look for any evidence where a candidate has not only thought about writing code to meet certain requirements but also thought about the wider context that that code is being used in.

Skills like documenting design decisions in a suitable format, thinking about prioritising requirements vs the amount of resource available and balancing this against both the business requirements/impact as well as the effect these decisions have on the development team itself. Being able to work with product managers who might not understand the technical detail of the code but work with other stakeholders to prioritise requirements. Some of these things cannot be ‘taught’ in a degree and can only be experienced but I think it is possible to mock a situation where this occurs.

## Name Withheld

**What do you think are the most important aspects of software quality? How would you define what it is?**

Readability: ability for others to quickly understand what the intent and semantics of the code base are, so they can make changes or debug problems.

Friction: code is structured to allow future software engineers to make changes easily and quickly. A subset of tech debt inhibits rapid iteration by causing friction, for example: tight coupling between two unrelated bits of functionality may cause an engineer to first spend days disentangling the two pieces before making changes to the underlying functionality.

Picking good abstractions (and/or picking one at all): many code bases grow organically with new functionality being tacked on over time. Well-chosen abstractions can massively simplify the complexity of a piece of software, and this happens at all levels from simple function signature choices to entire workflows.

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

Ranked from simple to complicated for software engineering. More senior roles will commonly require a good grasp of how to work towards business objectives and handle customer relationships as well.

1. Ability to express thought in code. Example: write 10 files worth of code with nice interface boundaries between them. I've found many grads lack good taste in this, even from many elite universities, because they haven't written a lot of complicated code.
2. Firm grasp of the option space of medium sized design. Example: knowing when to choose sets vs maps. This one is hammered into people, but I interviewed and worked with people who never quite internalized that you sometimes need to consciously walk through the available data structures, so they choose convoluted solutions because they haven't really thought about it.
3. Being able to do research. This is probably the most academically relevant yet immediately-abandoned skill by many software engineers: writing complicated software often involved a trade-off between building it yourself and leaning on mature-but-complicated open source solutions. A skill I look for in engineers is the ability to consciously do a few hours of digging to really understand not just what open source solutions exist, but **why** they chose specific trade-offs. The degree to which "the person who read an intro to SQL and really tried to understand it" gains serious edge in various organizations cannot be overstated—and this very much applies to higher level knowledge.

Regarding teachability in a university setting: the last one seems like a great opportunity in a university setting. Understanding why different major projects chose e.g. architectures, API patterns, deployment mechanisms etc is both very valuable because it can inform internal efforts to build similar products.

I expect the first two to be teachable, but to become **very** good at them, students would likely need high quality code review which may be impractical. I found that the CS department at St Andrews does this to a higher standard than most—I still remember Dr Nederhof's (correctly) critical code review, which helped me a lot to improve!

## Name Withheld

**What do you think are the most important aspects of software quality? How would you define what it is?**

*Being able to comprehensively anticipate the outcomes of errors is crucial. This means clearly understanding the likely failure scenarios of the software, how it behaves in each scenario, and the steps it takes to recover.*

*For instance, consider a microservice reliant on another microservice. It's important to know the response if the dependency fails. Does it attempt retries? What happens if it fails to succeed after 5 minutes? Does it become stuck, or is there a recovery mechanism in place?*

**What knowledge or skills related to software quality would you look for when hiring for software development roles? Which of these do you think should be taught within a degree? Are there any that can only be gained through experience?**

*I seek candidates who demonstrate patience and a methodical approach, prioritizing thoughtful solutions over rushing to conclusions.*

*I believe strongly that every skill can be taught, even if it involves practical, hands-on learning. To foster these qualities, I recommend workshops and practical exercises centered on software failure modes. For example, using scenarios with inadequate exception handling, students can learn to debug errors effectively and think critically about improving software reliability.*