Chapter 9
Conclusions

This book has sought to cover the most salient aspects of software quality assurance. In doing so we have discussed what software quality is, why it is important, and how it is defined. We have examined how the activities of quality assurance are closely linked to the choice of software development process. We have covered agile software development, testing, inspections, safey reviews, metrics, and cost estimation.

9.1 Topical and Emerging Quality Concerns

With the rapid emergence of new technology, and rapid changes in the way technology is used, the landscape of software quality assurance is constantly shifting. In the rest of this chapter, we look at what could be considered to be some of the key quality assurance challenges to have emerged in recent years.

9.1.1 Autonomy in Socio-Technical Systems

Let us consider two relatively recent incidents. In 2009, an Airbus A330 Air France flight 447 on its way from Rio de Janeiro to Paris crashed into the Atlantic, killing all 228 passengers on board. A subsequent crash investigation indicated the following sequence of events [116]:

- The plane, flying on autopilot, had encountered an adverse weather system, which had caused the Pitot tubes[1] to freeze.
- The autopilot disconnected and the control of the plane fell to manual control.
- The pilots were unprepared for the sudden disengagement, and were confused as to why this had happened.

[1] Small external tubes that are used to measure air-speed on aircraft.

N. Walkinshaw, *Software Quality Assurance*, Undergraduate
Topics in Computer Science, DOI 10.1007/978-3-319-64822-4_9

- In the mean time the plane had entered into a stall and descended rapidly (11,000 ft per minute) towards the ocean and crashed.

For the second incident, in May 2016 a Tesla Model S was driving along a highway at 74mph. The driver had engaged the "autopilot" mode in the car and was not concentrating on what was happening. A tractor-trailer crossed the path of the car, but its sensors failed to detect it. The car crashed under the side of the trailer, ripping off the roof of the car, and subsequently crashed into a pole at the side of the road, killing the driver.

Although the vehicles involved and the scales of the tragedies are different, both share two links. Firstly, and most obviously, the Airbus and the Tesla had been entirely *autonomous* (controlled by their own software without human intervention) in the run-up to the crash (in the case of the Airbus) and during the crash (in the case of the Tesla). Secondly, and perhaps more surprisingly, in both cases the software was *not* held to be responsible for the tragedies, which were ultimately blamed on human error.

Why *human* error? In the case of the Air France crash, the pilots were deemed to have lacked 'situational awareness '[46]; they should have immediately been able to determine why the autopilot had disengaged, and have thus reacted more speedily, in a more appropriate way. In the case of the Tesla, the driver was deemed again to lack situational awareness. For that car model, the system was only designed to keep the car in its lane and to avoid crashes with other cars (the scenario of trailer crossings was beyond the scope of the system).

This is understandable on the one level; the software systems when viewed in isolation performed exactly as they were designed to. However, when one takes a step back, and considers the broader context within which the systems were used, the culpability of the users is perhaps not so clear-cut. These systems are invariably complex, comprising a multitude of components and operators, where it is not necessarily possible for a single unit (human or technological) to maintain a coherent, macroscopic overview of the state of the system at any given time. In the case of the Air France disaster this problem has been demonstrated [116].

However, even in the case of the Tesla accident one could also argue that, though contrary to the system specifications and instructions, it is only to be expected that some drivers will be seduced by the idea of "driverless cars" and be willing to test its capabilities to the limit. Indeed, for typical road users, who are unaware of the detailed sensor configurations and limitations on the underlying control algorithms and Machine Learnt models, it can be argued that it is ultimately impossible for the driver to maintain a sufficient degree of "situational awareness" to be able to truly account for the behaviour of their car when it is under its own control.

So, although lack of situational awareness is to blame, the question of whether it is the human operator's *fault* is another question. The problems caused by the fuzzy boundaries between an inscrutably complex system (or system of systems) and a human operator are not new, and are not even specific to digital systems. In his book *"Normal Accidents: Living with High Risk Technologies"* [109], Perrow highlights how similar incidents – misuse of technology, rooted in misunderstanding and a lack of situational awareness, have contributed to some of the great disasters of our time, including the disasters at the Three-Mile Island and Chernobyl nuclear power plants.

Although the problem in its essence is not new, there is a strong argument to be made that the increased pervasiveness of software-driven technology is greatly exacerbating it. Software is taking over activities that have traditionally been entirely manual, and the rules by which people interact with technology are constantly being re-written. With these changes, users (or drivers or pilots) are bound to be uncertain about the boundary between their areas of responsibility, and the 'system's' set of responsibilities.

From a quality assurance perspective, this blurring of responsibilities between the operator and the system put a new spin on long-standing questions. What is the scope of a 'system'? What use and context should the system design take into account? What should be the 'contract' between the user and the system, and how should this be communicated to users, to prevent the sorts of misunderstandings that we have described above?

9.1.2 Data-Intensive, Untestable Systems

Machine Learning algorithms used to play a relatively confined role when it came to software systems, finding their uses for relatively 'niche' activities such as detecting junk emails or credit card fraud. However, as the prevalence of data (especially data that pertains to individuals) has grown, and Machine Learning algorithms have become more versatile and powerful, their role in the functionality of software has greatly increased. The driverless cars discussed above represent one particular area where Machine Learning has become a central component. However, it has become prominent in almost any area that involves large volumes of data, from detecting user web-browsing patterns, to intrusion detection in networks and detecting suitable trades in financial systems. These systems become problematic when the algorithms, which are developed and trained to react to a vast range of scenarios, encounter one example of a scenario that they have not been prepared for.

As an example we refer to another example of a driverless car crash, also in 2016, but this time one that did not cause any fatalities. In May 2016 in Mountain View, a Google driverless car pulled out from a parked position and crashed into the side of a bus that was overtaking it[2]. Here, the fault *did* lie with the software system. This was not an isolated event; according to a document filed by Google with the California Department for Motor Vehicles, its driverless car software experienced 272 failures and would have crashed 13 times had it not been for human intervention[3]. The various problems that are thrown up in driverless cars by Machine Learning

---

[2] `https://www.theguardian.com/technology/2016/feb/29/google-self-driving-car-accident-california`

[3] `https://www.theguardian.com/technology/2016/jan/12/google-self-driving-cars-mistakes-data-reports`

algorithms are discussed in more detail by Wagner and Koopman [130] (Koopman had also investigated the Toyota Unintended Acceleration fault - see Chapter 2).

Of course, driverless cars are far from the only area within which data-intensive algorithms have become increasingly prevalent. Another area is in the finance sector, and specifically in the form of *High Frequency Trading (HFT)* algorithms. HFT algorithms analyse data from a variety of sources; they monitor real-time stock-market data, often alongside large streams of news-information, and use this data to automatically trade on the stock market. They use this data to predict future stock values, often by the use of intricate statistical data processing algorithms, often using the results to execute thousands of trades per second.

When these systems malfunction, they can have potentially disastrous effects, not just on individual businesses, but entire economies. There are plenty of examples [84], and we pick out a few notable ones here. In August 2012, Knight Capital introduced a faulty trading algorithm to the market. As soon as they activated it, it lost approximately £6.4m per minute, ultimately losing £281m before it could be switched off. Aside from such individual failures, the more frightening problems arise when HFT algorithms interact with each other to produce "flash crashes". In 2010, such an event led to the loss of 998 points (approximately 9%) off the Dow Jones Industrial Average, only to rise to its previous level after about 15 minutes. More recently, in the aftermath of the Brexit vote, the British pound slumped 6% against the dollar in a similar flash crash, only to regain its value again after a couple of minutes.

What are the similarities that link HFT systems to driverless cars? They both involve the use of Machine Learning and other statistical data processing algorithms to process large volumes of data. Both are either safety or business critical; when the underlying software is faulty, the consequences can be disastrous.

Exercise: *Before reading on, think back to the chapter on software testing. Using testing terminology, what is the problem with statistical data processing / Machine Learning algorithms?*

Statistical data processing and Machine Learning algorithms pose interesting problems from a quality assurance perspective because their "correct" behaviour is difficult to anticipate and express. They exist to mine and discover things about data that are not necessary known or even knowable a-priori, which means that a lot of their outputs will by necessity be unpredictable.

As a result, such systems are intrinsically difficult to specify. Though straightforward in the abstract (a car should not collide with other cars, a trading algorithm should minimise losses and maximise gains) tying these requirements to implementation details and verification or validation activities is extremely difficult. "Other cars" could correspond to a vast range of possible sensor signal patterns and could, depending on the sensor configurations, vary according to light / road conditions, speed, etc. The question of whether a trading algorithm is successful at minimising losses and maximising gains depends on a host of external factors (the combination of current events, competing trading algorithms, current stock market movements, etc.), where it is practically impossible to determine “the” ideal behaviour.

The standard approaches to quality assurance and testing would struggle (and probably fail) to provide compelling answers to these questions. Although the problem is not new [133], the prevalence of such systems certainly makes the task of finding a compelling solution all the more urgent.

9.2 Concluding Remarks

Software development does not take place in a vacuum, and is therefore inevitably an involved, at times messy, process. Users are capricious and liable to change their minds about what they want. Technology is evolving at a fast pace, demanding continuous changes to the way in which software operates. The time and effort required for various activities can often be grossly underestimated (and occasionally overestimated).

Ultimately, the challenge of trying to produce a successful software system in the face of these various forces lies with a band of fallible human software developers. They probably have varying abilities, are subject to different time pressures, may be geographically distributed, perhaps don’t work under the auspices of the same organisation, and possibly don’t even know each other. Often it is these same developers that are also responsible for managing the quality assurance of the product.

It is unsurprising that even the most safety-critical or business-critical systems can end up with quality problems. There are plenty of techniques that can improve quality assurance, however, no technique is bullet proof. All approaches tend to require a degree of intuition and experience, and rely on a level of discipline, time, and effort that is rarely practical in a realistic software engineering context.

This is what makes quality assurance especially interesting: Far from being an after-thought to be ‘done’ if there is time, it is integral throughout – the most vital, and most interesting angle of software development. The dynamics of software development are relentless, subject to so many human and technological factors. And in the face of all of this, you are given only an imperfect armoury of tools, and very little time to ensure the quality of a product where failure to do so can have significant (potentially devastating) consequences.

As it stands, it is impossible to guarantee that a software system will be bug free, will be readily maintainable, and be delivered on schedule. It is even impossible to reliably *measure* the quality of a software system. The big challenges of software quality assurance have not yet been solved. As time passes, technology and the way in which it is used will continue to evolve at breakneck speed. Software systems will inevitably become increasingly complex, larger in scale, and increasingly safety- and business-critical.

One aim of this book has been to present an overview of the key problems, principles, and techniques within the remit of quality assurance. Given the breadth of the area, this has by necessity been selective. For any of the topics covered there exist enormous volumes of in-depth text books and research publications.

And this is where your pursuit of software quality assurance can begin in earnest! If you have found yourself looking up references, alternative approaches, or querying whether one of the techniques described in this book really is the most appropriate solution to a problem, then the book has fulfilled its ultimate goal – to pique your interest.

References

[1] (1982) DO-178B: Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics (RTCA)

[2] (1988) DOD-STD-2167A, Defense Systems Software Development. Department of Defence

[3] (2006) IEC 60880: Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions. International Electrotechnical Commission

[4] (2011) ISO 26262-1:2011: Road Vehicles - Functional Safety. International Standards Organisation

[5] (2011) ISO/IEC 25010:2011: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. International Standards Organisation

[6] (2015) The promise repository of empirical software engineering data

[7] Achimugu P, Selamat A, Ibrahim R, Mahrin MN (2014) A systematic literature review of software requirements prioritization research. Information and software technology 56(6):568–585

[8] Albrecht AJ (1979) Measuring application development productivity. In: Proceedings of the joint SHARE/GUIDE/IBM application development symposium, vol 10, pp 83–92

[9] Alexander C (1977) A pattern language: towns, buildings, construction. Oxford University Press

[10] Anderson J (2011) A million monkeys and shakespeare. Significance 8(4):190–192

[11] Arcuri A, Briand L (2011) Adaptive random testing: An illusion of effectiveness? In: Proceedings of the 2011 International Symposium on Software Testing and Analysis, ACM, pp 265–275

[12] Bacchelli A, Bird C (2013) Expectations, outcomes, and challenges of modern code review. In: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, pp 712–721

N. Walkinshaw, *Software Quality Assurance*, Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-64822-4

[13] Barr ET, Harman M, McMinn P, Shahbaz M, Yoo S (2015) The oracle problem in software testing: A survey. Software Engineering, IEEE Transactions on 41(5):507–525
[14] Barrett SR, Speth RL, Eastham SD, Dedoussi IC, Ashok A, Malina R, Keith DW (2015) Impact of the volkswagen emissions control defeat device on us public health. Environmental Research Letters 10(11):114,005
[15] Basili VR, Briand LC, Melo WL (1996) A validation of object-oriented design metrics as quality indicators. IEEE Transactions on software engineering 22(10):751–761
[16] Basili VR, Green S, Laitenberger O, Lanubile F, Shull F, Sørumgård S, Zelkowitz MV (1996) The empirical investigation of perspective-based reading. Empirical Software Engineering 1(2):133–164
[17] Beck F, Diehl S (2011) On the congruence of modularity and code coupling. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM, pp 354–364
[18] Beller M, Bacchelli A, Zaidman A, Juergens E (2014) Modern code reviews in open-source projects: which problems do they fix? In: Proceedings of the 11th working conference on mining software repositories, ACM, pp 202–211
[19] Bentley JL, McIlroy MD (1993) Engineering a sort function. Software: Practice and Experience 23(11):1249–1265
[20] Boehm B, Clark B, Horowitz E, Westland C, Madachy R, Selby R (1995) Cost models for future software life cycle processes: Cocomo 2.0. Annals of software engineering 1(1):57–94
[21] Boehm BW (1988) A spiral model of software development and enhancement. Computer 21(5):61–72
[22] Boehm BW, Brown JR, Kaspar H (1978) Characteristics of software quality
[23] Boehm BW, et al (1981) Software engineering economics, vol 197. Prentice-hall Englewood Cliffs (NJ)
[24] Borning A (1987) Computer system reliability and nuclear war. Communications of the ACM 30(2):112–131
[25] Brooks F (1975) The Mythical Man Month. Information Systems Programs, General Electric Company
[26] Buxton JN, Randell B (1970) Software engineering techniques: report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969. NATO Science Committee
[27] Challenger PCOSS, Rogers W (1986) Report of the presidential commission on the space shuttle challenger accident
[28] Chen TY, Leung H, Mak I (2004) Adaptive random testing. In: Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making, Springer, pp 320–329
[29] Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Transactions on software engineering 20(6):476–493
[30] Chow TS (1978) Testing software design modeled by finite-state machines. IEEE transactions on software engineering 4(3):178
[31] Chung L, Nixon BA, Yu E, Mylopoulos J (2012) Non-functional requirements in software engineering, vol 5. Springer Science & Business Media

[32] Clegg D, Barker R (1994) Case method fast-track: a RAD approach. Addison-Wesley Longman Publishing Co., Inc.

[33] Cohen D, Lindvall M, Costa P (2004) An introduction to agile methods. Advances in computers 62:1–66

[34] Committee ICSSES, Board ISS (1998) Ieee recommended practice for software requirements specifications. Institute of Electrical and Electronics Engineers

[35] of Commons Public Accounts Select Committee H, et al (2013) Universal credit: early progress. URL https://www.nao.org.uk/wp-content/uploads/2013/09/10132-001-Universal-credit.pdf

[36] Company GE, McCall JA, Richards PK, Walters GF (1977) Factors in software quality: Final report. Information Systems Programs, General Electric Company

[37] Comptroller General (1981) Norad's missile warning system: What went wrong? URL http://www.gao.gov/assets/140/133240.pdf

[38] Crosby PB (1980) Quality is free: The art of making quality certain. Signet

[39] Darimont R, Delor E, Massonet P, van Lamsweerde A (1997) Grail/kaos: an environment for goal-driven requirements engineering. In: Proceedings of the 19th international conference on Software engineering, ACM, pp 612–613

[40] De Neufville R (1994) The baggage system at denver: prospects and lessons. Journal of Air Transport Management 1(4):229–236

[41] Demeyer S, Ducasse S, Nierstrasz O (2002) Object-oriented reengineering patterns. Elsevier

[42] Dijkstra EW (1968) Letters to the editor: go to statement considered harmful. Communications of the ACM 11(3):147–148

[43] Dijkstra EW (1972) The humble programmer. Communications of the ACM 15(10):859–866

[44] Domke F, Lange D (2015) The exhaust emissions scandal ("dieselgate"). URL https://events.ccc.de/congress/2015/Fahrplan/system/event_attachments/attachments/000/002/812/original/32C3_-_Dieselgate_FINAL_slides.pdf

[45] Dybå T, Dingsøyr T (2008) Empirical studies of agile software development: A systematic review. Information and software technology 50(9):833–859

[46] Endsley MR (1995) Toward a theory of situation awareness in dynamic systems. Human Factors: The Journal of the Human Factors and Ergonomics Society 37(1):32–64

[47] Eypasch E, Lefering R, Kum C, Troidl H (1995) Probability of adverse events that have not yet occurred: a statistical reminder. BMJ: British Medical Journal 311(7005):619

[48] Fagan M (1976) Design and code inspections to reduce errors in program development. IBM Journal of Research and Development 15(3):182

[49] Federal Aviation Administration (2015) Docket no. faa-2015-0936; directorate identifier 2015-nm-058-ad; amendment 39-18153; ad 2015-09-07. URL https://s3.amazonaws.com/public-inspection.federalregister.gov/2015-10066.pdf

[50] Fenton N (1994) Software measurement: A necessary scientific basis. IEEE Transactions on software engineering 20(3):199–206

[51] Fenton NE, Whitty RW, Iizuka Y (1995) Software Quality Assurance and Measurement: A Worldwide Perspective. Itp-Media

[52] Flyvbjerg B, Budzier A (2011) Why your it project may be riskier than you think. Harvard Business Review 89(9):601–603

[53] Fowler M (2004) UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional

[54] Fowler M, Beck K (1999) Refactoring: improving the design of existing code. Addison-Wesley Professional

[55] Fowler M, Highsmith J (2001) The agile manifesto. Software Development 9(8):28–35

[56] Fraser G, Arcuri A (2011) EvoSuite: Automatic test suite generation for object-oriented software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ACM, New York, NY, USA, ESEC/FSE'11, pp 416–419

[57] Gamma E, Helm R, Johnson R, Vlissides J (1994) Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley

[58] Glinz M (2007) On non-functional requirements. In: Requirements Engineering Conference, 2007. RE'07. 15th IEEE International, IEEE, pp 21–26

[59] Goldberg DE (2006) Genetic algorithms. Pearson Education India

[60] Goodenough JB, Gerhart SL (1975) Toward a theory of test data selection. IEEE Transactions on Software Engineering 1(2):156–173

[61] Gotel OC, Finkelstein C (1994) An analysis of the requirements traceability problem. In: Requirements Engineering, 1994., Proceedings of the First International Conference on, IEEE, pp 94–101

[62] Gousios G, Pinzger M, Deursen Av (2014) An exploratory study of the pull-based software development model. In: Proceedings of the 36th International Conference on Software Engineering, ACM, pp 345–355

[63] Groce A, Holzmann G, Joshi R (2007) Randomized differential testing as a prelude to formal verification. In: 29th International Conference on Software Engineering (ICSE'07), IEEE, pp 621–631

[64] Guimarães ML, Silva AR (2012) Improving early detection of software merge conflicts. In: Software Engineering (ICSE), 2012 34th International Conference on, IEEE, pp 342–352

[65] Hall M, Walkinshaw N, McMinn P (2012) Supervised software modularisation. In: Software Maintenance (ICSM), 2012 28th IEEE International Conference on, IEEE, pp 472–481

[66] Halstead M (1977) Elements of Software Science. North-Holland

[67] Hamlet R (1994) Random testing. Encyclopedia of software Engineering

[68] Hannay JE, Dybå T, Arisholm E, Sjøberg DI (2009) The effectiveness of pair programming: A meta-analysis. Information and Software Technology 51(7):1110–1122

[69] Humphrey W (1989) Managing the Software Process. Addison Wesley

[70] Inozemtseva L, Holmes R (2014) Coverage is not strongly correlated with test suite effectiveness. In: Proceedings of the 36th International Conference on Software Engineering, ACM, pp 435–445

[71] Isaac R (2013) The pleasures of probability. Springer Science & Business Media

[72] Jacobson I (1992) Object-Oriented Software Engineering. A Use Case Driven Approach. Addison-Wesley

[73] Jacobson I, Booch G, Rumbaugh J, Rumbaugh J, Booch G (1999) The unified software development process, vol 1. Addison-Wesley Reading

[74] Jia Y, Harman M (2011) An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering 37(5):649–678

[75] Johnson PM, Tjahjono D (1998) Does every inspection really need a meeting? Empirical Software Engineering 3(1):9–35

[76] Johnson SC (1977) Lint, a C program checker. Bell Telephone Laboratories Murray Hill

[77] Jørgensen M (1999) Software quality measurement. Advances in engineering software 30(12):907–912

[78] Juran J (1970) Quality planning and analysis: from product development through usage

[79] Just R, Jalali D, Inozemtseva L, Ernst MD, Holmes R, Fraser G (2014) Are mutants a valid substitute for real faults in software testing? In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, pp 654–665

[80] Kan SH (2002) Metrics and models in software quality engineering. Addison-Wesley Longman Publishing Co., Inc.

[81] Kano N, Seraku N, Takahashi F, Tsuji S (1984) Attractive quality and must-be quality. Journal of the Japanese Society for Quality Control 14

[82] Kelly T, Weaver R (2004) The goal structuring notation–a safety argument notation. In: Proceedings of the dependable systems and networks 2004 workshop on assurance cases

[83] King JC (1976) Symbolic execution and program testing. Communications of the ACM 19(7):385–394

[84] Kirilenko AA, Lo AW (2013) Moore's law versus murphy's law: Algorithmic trading and its discontents. The Journal of Economic Perspectives 27(2):51–72

[85] Kirsch M (2014) Technical support to the national highway traffic safety administration (nhtsa) on the reported toyota motor corporation (tmc) unintended acceleration (ua) investigation. URL `http://www.nasa.gov/topics/nasalife/features/nesc-toyota-study.html`

[86] Koopman P (2014) A case study of toyota unintended acceleration and software safety. URL `betterembsw.blogspot.co.uk/2014/09/a-case-study-of-toyota-unintended.html`

[87] Kransner G, Pope S (1988) Cookbook for using the model-view-controller user interface paradigm. Object Oriented Programming pp 26–49

[88] Langner R (2011) Stuxnet: Dissecting a cyberwarfare weapon. IEEE Security & Privacy 9(3):49–51

[89] Larman C, Basili VR (2003) Iterative and incremental development: A brief history. Computer (6):47–56

[90] Lee D, Yannakakis M (1996) Principles and Methods of Testing Finite State Machines - A Survey. In: Proceedings of the IEEE, vol 84, pp 1090–1126

[91] Leroy X (2007) Formal verification of an optimizing compiler. Lecture Notes in Computer Science 4533:1

[92] Malcolm DG, Roseboom JH, Clark CE, Fazar W (1959) Application of a technique for research and development program evaluation. Operations research 7(5):646–669

[93] McCabe TJ (1976) A complexity measure. IEEE Transactions on software Engineering (4):308–320

[94] McMinn P (2004) Search-based software test data generation: A survey. Software Testing Verification and Reliability 14(2):105–156

[95] Mens T (2002) A state-of-the-art survey on software merging. IEEE transactions on software engineering 28(5):449–462

[96] Menzies T, Yang Y, Mathew G, Boehm B, Hihn J (2016) Negative results for software effort estimation. arXiv preprint arXiv:160905563

[97] Miguel JP, Mauricio D, Rodriguez G (2014) A review of software quality models for the evaluation of software products. CoRR abs/1412.2977

[98] Moløkken-Østvold K, Haugen NC, Benestad HC (2008) Using planning poker for combining expert estimates in software projects. Journal of Systems and Software 81(12):2106–2117

[99] Moore EF (1956) Gedanken–experiments on sequential machines. In: Shannon CE, McCarthy J (eds) Annals of Mathematics Studies (34), Automata Studies, Princeton University Press, Princeton, NJ, pp 129–153

[100] Nair S, De La Vara JL, Sabetzadeh M, Briand L (2014) An extended systematic literature review on provision of evidence for safety certification. Information and Software Technology 56(7):689–717

[101] Nerur S, Mahapatra R, Mangalaraj G (2005) Challenges of migrating to agile methodologies. Communications of the ACM 48(5):72–78

[102] Nuseibeh B, Easterbrook S (2000) Requirements engineering: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering, ACM, pp 35–46

[103] Oman P, Hagemeister J (1992) Metrics for assessing a software system's maintainability. In: Software Maintenance, 1992. Proceerdings., Conference on, IEEE, pp 337–344

[104] Ostrand TJ, Balcer MJ (1988) The category-partition method for specifying and generating fuctional tests. Communications of the ACM 31(6):676–686

[105] Parnas DL (1972) On the criteria to be used in decomposing systems into modules. Communications of the ACM 15(12):1053–1058

[106] Parnas DL (1985) Software aspects of strategic defense systems. Communications of the ACM 28(12):1326–1335

[107] Parnas DL, Clements PC (1986) A rational design process: How and why to fake it. Software Engineering, IEEE Transactions on (2):251–257
[108] Parnas DL, Weiss DM (1985) Active design reviews: principles and practices. In: Proceedings of the 8th international conference on Software engineering, IEEE Computer Society Press, pp 132–136
[109] Perrow C (1984) Normal Accidents: Living with High Risk Technologies. Princeton University Press
[110] Pezzè M, Young M (2007) Software testing and analysis - process, principles and techniques. Wiley
[111] Potvin R, Levenberg J (2016) Why google stores billions of lines of code in a single repository. Communications of the ACM 59(7):78–87
[112] Rigby PC, Bird C (2013) Convergent contemporary software peer review practices. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ACM, pp 202–212
[113] Rosenberg D, Stephens M (2003) Extreme programming refactored: the case against XP. Apress
[114] Royce WW (1970) Managing the development of large software systems. In: proceedings of IEEE WESCON, Los Angeles, vol 26, pp 1–9
[115] Rubin KS (2012) Essential Scrum: A practical guide to the most popular Agile process. Addison-Wesley
[116] Salmon PM, Walker GH, Stanton NA (2016) Pilot error versus sociotechnical systems failure: a distributed situation awareness analysis of air france 447. Theoretical Issues in Ergonomics Science 17(1):64–79
[117] Sen K, Marinov D, Agha G (2005) CUTE: a concolic unit testing engine for C, vol 30. ACM
[118] Series NWP (1980) If Japan Can, Why Can't We? URL `https://www.youtube.com/watch?v=vcG\_Pmt\_Ny4`
[119] Shaw M, Garlan D (1996) Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall
[120] Shepperd M (1988) A critique of cyclomatic complexity as a software metric. Software Engineering Journal 3(2):30–36
[121] Shewhart WA (1931) Economic control of quality of manufactured product. ASQ Quality Press
[122] Sindre G, Opdahl AL (2005) Eliciting security requirements with misuse cases. Requirements engineering 10(1):34–44
[123] Sjøberg DI, Anda B, Mockus A (2012) Questioning software maintenance metrics: a comparative case study. In: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement, ACM, pp 107–110
[124] Spinellis D (2003) Code reading: the open source perspective. Addison-Wesley Professional
[125] Staats M, Whalen MW, Heimdahl MP (2011) Programs, tests, and oracles: the foundations of testing revisited. In: Software Engineering (ICSE), 2011 33rd International Conference on, IEEE, pp 391–400

[126] Staples M, Niazi M, Jeffery R, Abrahams A, Byatt P, Murphy R (2007) An exploratory study of why organizations do not adopt cmmi. Journal of systems and software 80(6):883–895

[127] Team CP (2010) CMMI for Services Version 1.3. Lulu. com

[128] UK MOD (2004) Tornado safety case report. URL `https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/475938/20040616_Annex_C_-_Tornado_SC__v1_U_0__-_Baseline.pdf`

[129] Van Solingen R, Berghout E (1999) The Goal/Question/Metric Method: a practical guide for quality improvement of software development. McGraw-Hill

[130] Wagner M, Koopman P (2015) A philosophy for developing trust in self-driving cars. In: Road Vehicle Automation 2, Springer, pp 163–171

[131] Wagner S, Lochmann K, Heinemann L, Kläs M, Trendowicz A, Plösch R, Seidl A, Goeb A, Streit J (2012) The quamoco product quality modelling and assessment approach. In: Proceedings of the 34th international conference on software engineering, IEEE Press, pp 1133–1142

[132] Weyuker EJ (1979) Translatability and decidability questions for restricted classes of program schemas. SIAM Journal on Computing 8(4):587–598

[133] Weyuker EJ (1982) On testing non-testable programs. The Computer Journal 25(4):465–470

[134] Wilson JM (2003) Gantt charts: A centenary appreciation. European Journal of Operational Research 149(2):430–437

[135] Wood M, Roper M, Brooks A, Miller J (1997) Comparing and combining software defect detection techniques: a replicated empirical study. In: ACM SIGSOFT Software Engineering Notes, Springer-Verlag New York, Inc., vol 22, pp 262–277

[136] Woodcock J, Davies J (1996) Using Z: specification, refinement, and proof, vol 39. Prentice Hall Englewood Cliffs

[137] Wynne M, Hellesoy A (2012) The cucumber book: behaviour-driven development for testers and developers. Pragmatic Bookshelf

[138] Yuan D, Luo Y, Zhuang X, Rodrigues GR, Zhao X, Zhang Y, Jain PU, Stumm M (2014) Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pp 249–265

[139] Zhu H, Hall PA, May JH (1997) Software unit test coverage and adequacy. Acm computing surveys (csur) 29(4):366–427

[140] Zowghi D, Coulin C (2005) Requirements elicitation: A survey of techniques, approaches, and tools. In: Engineering and managing software requirements, Springer, pp 19–46

Index

N. Walkinshaw, *Software Quality Assurance*, Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-64822-4