Software Engineering Timeline: major areas of interest and multidisciplinary trends

Isabel M. del Águila, José del Sagrado and Joaquín Cañadas
Department of Informatics, University of Almería,
Almería 04120, Spain

December 2019

Abstract

Society today cannot run without software and by extension, without Software Engineering. Since this discipline emerged in 1968, practitioners have learned valuable lessons that have contributed to current practices. Some have become outdated but many are still relevant and widely used. From the personal and incomplete perspective of the authors, this paper not only reviews the major milestones and areas of interest in the Software Engineering timeline helping software engineers to appreciate the state of things, but also tries to give some insights into the trends that this complex engineering will see in the near future.

Keywords: History of computing, Software Evolution, Software Methodologies

1 Introduction

Computer systems have progressed extraordinarily over the last half century along with one of their core components - the software. This progress has been mirrored by people's ability to embrace it; all of us use a computer on a day-to-day basis, whether directly or indirectly. Software Engineering (SE) is tasked with fostering software development; it oversees all aspects of software production, from the early stages of system specification through to system maintenance until it comes into use [39].

Since the late 1970s, this knowledge area has been a subtle yet fundamental part of our daily life given that software underpins countless everyday human activities. Nevertheless, none of us are aware of its presence, nor its complexity - until, of course, it fails or crashes [9].

The worldwide software industry generates a huge amount of money in revenue annually and continues to expand in scope and revenue volume [32]. As with most human disciplines, SE matured out of the necessity to deal with the various challenges encountered since its inception 50 years ago. This has created

the widest variation of tools, methods and languages of any engineering field in human history. Based on these challenges, we have constructed an SE timeline that uncovers our personal multidisciplinary trend proposal that software engineers might face in the near future.

Establishing a simile with the Oedipus's answer to the riddle of the sphinx, SE has been evolving through several ages from childhood to senescence, passing through adulthood (see Figure 1), although senescence has not been reached yet. The transition from childhood to adulthood can be dated back to the early nineties when software development established itself as a worldwide industry. Software applications expanded to multiple domains such as telecommunications, the military, industrial processes, and entertainment, becoming in an adult discipline. The Second Age is still with us; today, we cannot know whether the advances to come, will drive SE towards a new stage, driven by multidisciplinarity and knowledge, being these the last events drawn on the proposed timeline.

We have divided the proposed ages into several eras, as it had done in others SE history related works [12, 15], whose boundaries are a bit blurred. Each era is defined by a prevailing idea (see table 1) about the main challenges characterized within it. These challenges generated new SE methods and techniques to take another step in SE evolution. The milestones that have been selected represent either a unifying moment or a bifurcation leading to new approaches (e.g. the first one marks the birth of SE as a discipline).

Table 1: Prevalent idea behind each era

Age	Era	Period	This era is mastered by
	Mastering the Machine	1956-1967	Hardware resources defined software
1	Mastering the Process	1968-1982	Methodologies guide the software development
	Mastering the Complexity	1983-1992	Domains and complexity raise force enhanced methods and tools
	Mastering Communications	1993-2001	Distributed environments change the processes
2	Mastering the Productivity	2001-2010	Software factories manage the rules
	Mastering the Diversity	2010-2017	Devices, platforms or approaches variety expands the used methods
	Mastering the Knowledge	2018- ... new era?	SE Knowledge should be managed

2 The First Age

At the beginning, the main purpose of any software was to optimize the exploitation of the limited hardware resources available. It was in 1956 when General Motors produced the first operating system (i.e. GM-NAA I/O for IBM704), becoming this year our starting point for our SE timeline, even when the term SE had not been coined yet. Thus, **Mastering the Machine (1956-1967)** is the first era we have identified as the start of this Age. It was characterized by

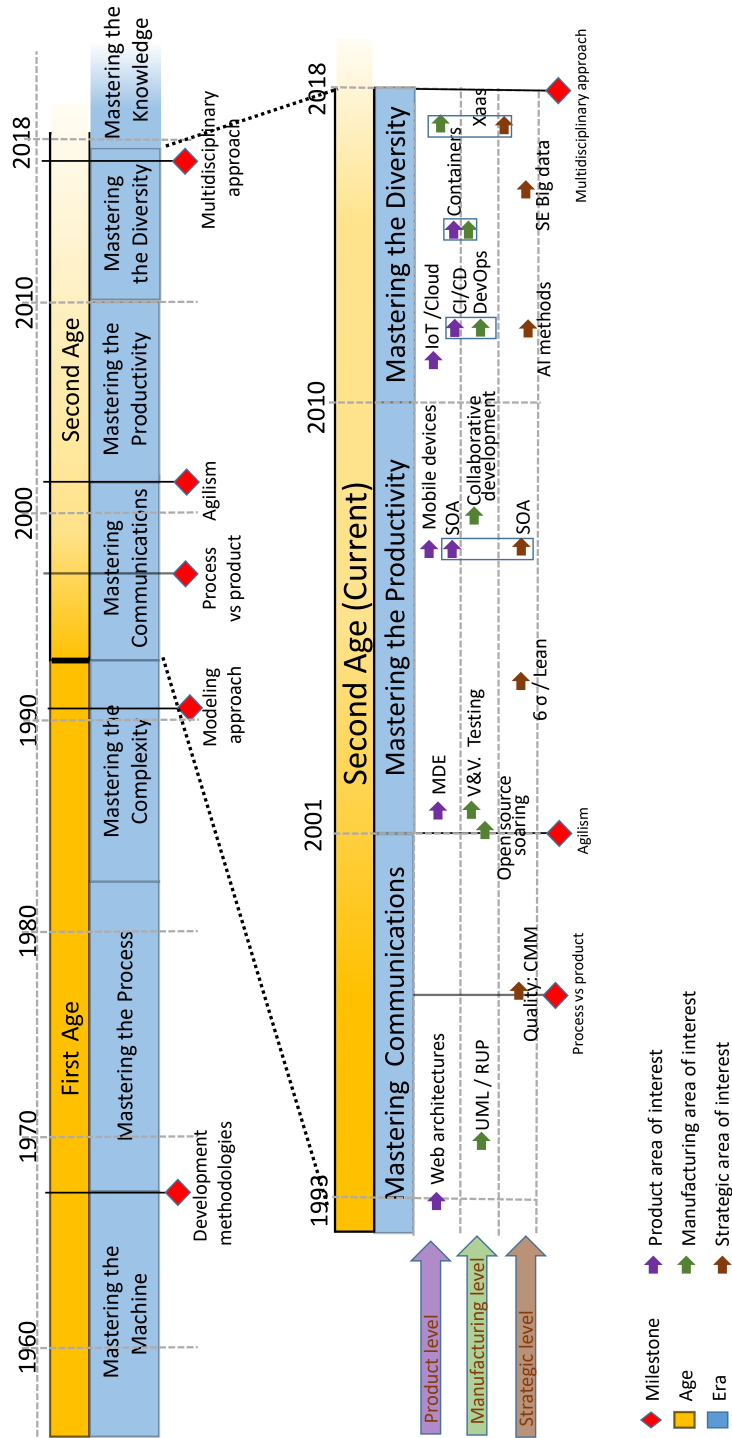

Figure 1: SE Timeline

the lack of software development methods, which led to the origin of the term Software Engineering in NATO Science Committee, Garmisch, Germany, 1968 [30] this was the *development methodologies* milestone.

The second era, **Mastering the Process (1968-1982)**, was driven by the infamous software crisis, or maybe software chronic disease [33], which forced developers to focus on the stages of software specification and maintenance to deal with software aging [13]. A number of structured methods arose, such as Software Requirement Engineering Methodology (SREM) [37], or Structured Analysis and Design Technique (SADT) [1], allowing the development of specification documents for business management software. These methodologies extended the concepts of modularization and information hiding, previously applied in structured programming [23, 14], from design to specification phase. The rise of software engineering standard is also a major accomplishment dated in this age. The community as a whole were starting to focus on standards as a means of achieving the goals [40]

Later, in the **Mastering the Complexity era (1983-1992)**, the predominance of hardware over software came to an end, and application complexity increased exponentially. Computer Aided Software Engineering (CASE) tools governed this SE period, as they gave support to engineers of this emerging discipline. Even though the main modeling approaches - data modeling and function modeling - still followed separate paths, they converged in object-oriented methods (OO) [29]; such was the case early on with structured methodologies, which were first introduced into coding and design but eventually made their way into specification and analysis. This OO approach enabled efficient software reuse and thus improved productivity in the process for building software [29]. A second milestone, at the end of the first SE age, was the need to evolve to a *modeling approach* that encouraged models to support software construction, which translated into the natural evolution of OO methods [8]. Nowadays, this is a fundamental pillar in software development.

3 The Second Age

The second Age began after about 25 years after the first milestone, when SE was sufficiently resilient to reach maturity [27]. Several changes forced SE to grow up. On the one hand, software development started to be considered as an industrial business, and on the other, globalization had a deep impact on development processes, customer' feedbacks, and competitiveness. What is more, even research community provided nurturing advances in a variety of ways to SE practitioners, although research impact was not fully felt until at least 10 years later [32]. A good example of this can be found in the work of the software engineering coordinating committee, which began in 1997, to define the "generally accepted" knowledge about software engineering as a profession. This project, based on consensus, released the first trial version of the *Guide to the Software Engineering Body of Knowledge (SWEBOK)* in 2001, which third released in 2014 edition is the newest one [11].

One can delineate this second Age by describing several areas of interest that join in the timeline (see the bottom of Figure 1).Some of them tend to set up a string of connected links evolving around similar ideas; such as the case of the thread comprising model-driven, service-oriented, containers and everything as a service areas, which exploits the notion of building software by assembling components or pieces. Figure 1 describes these areas considering three connected points of view (or levels). The product level includes those topics that view software as an artifact, as well as those that are physically closer to software, such as infrastructure or hardware related issues. The manufacturing level involves the processes and methodologies employed in building software applications. Finally, the strategic level focuses on the business perspective, dealing with the high-level decisions and organizational tasks for the entire software development business, also called umbrella areas [35].

3.1 Mastering Communication

Software companies became factories in the **Mastering Communications (1993-2001)** era. The intertwining of commercial and research networks by the early 1990s marks the birth of the Internet as a global accessible network. On October 24, 1995, the Federal Networking Council unanimously passed a resolution defining the term Internet [26]. The emergence of the World Wide Web brought with it a new software concept, which causes SE methods to encompass distributed system development. *Web applications and client/server architectures* appeared as a new area of interest at the product level [6]. Object-oriented technology evolved into software reuse through the design of reusable patterns [19] and components-based software development [25]. Modeling approaches remained as a core element in software development and were included in a significant number of emerging methodologies. After this burgeoning of methods, conflicts began to appear. Rational Corporation (now part of IBM) solved them by integrating top three methods (James Rumbaugh's, Grady Booch's and Ivar Jacobson's methods), which led to the release in 1997 of *Unified Modeling Language* (UML) [10].

Software factories needed to ensure quality control, which required the effective separation between process and product: *the process vs product* milestone. The *Rational Unified Process* (RUP) was defined as UML partner to model the development processes [24] RUP is use case driven, architecture centric, iterative and incremental and including cycles, phases, workflows, risk mitigation, quality control, project management and configuration control. Certain frameworks appeared to manage both the product and the process, providing guidance for developing or improving processes to meet the business goals of an organization. These included *CMM/CMMI* (capability maturity model/CMMintegrated) by the Software Engineering Institute (SEI) which adapted the principles of process improvement from the manufacturing field to the software field [34]

3.2 Mastering the Productivity

The fourth milestone, agilism, was a turning point in SE (agilemanifesto.org). Agile methods promoted frequent inspection and adaptation by introducing checkpoints where one can reassign customer requirements. They also encouraged software development as an incremental, cooperative, straightforward and adaptive process. At the same time, Open-source movement propelled SE to improve the collaborative and distributed software building methods, Not only companies, but also developer communities started to share and enhance software applications (www.linuxfoundation.org, www.eclipse.org). And even more, supporting knowledge also started to be shared (stackoverflow.com). *Open source soaring* and agilism radically affected how the software had to be built, setting up the basis at manufacturing level.

Agilism ushered in a new era, **Mastering the Productivity (2001-2010)**, in which software became yet another company asset. The goals of software factories became to reduce defects; to perform faster and more reliable processes; to increase customer satisfaction and to get greater profits. It also was at the start of this century, with the appearance of agilism, that one can date other two areas of interest: *Model-Driven Engineering* (MDE) [31] and *V&V testing* (730-2014 - IEEE Standard for Software Quality Assurance Processes, IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004) - IEEE Standard for System and Software Verification and Validation).

MDE empowered models as first class artifacts in software development, adapting methodologies by increasing the abstraction levels of SE tasks up to those used in the problem description; enhancing productivity by the automatic execution of model transformations and code generation; and providing domain specific languages for many software development tasks. To address the software quality issue, verification and validation (*V&V*) methods focused on *testing* technologies as a way of identifying software correctness from an agile perspective.

Although quality management methods had already been applied to SE, it is worth highlighting the use of quantitative approaches in this era, such as *6σ* (Six Sigma) and *Lean* [7], to support decision making at the strategic level in SE companies. Six Sigma, which was defined as a quality measurement to reduce variation and prevent defects, also became a management approach that promoted the need for fact-based decisions, customer focus, and teamwork.

At the end of the decade, the widespread use of *mobile devices*, together with the extended use of *collaborative software development platforms* (e.g. GitHub, Jazz Project and StackOverflow), led to *service-oriented approaches* (SOA) [18] as a response to complexity, agile application development and software evolution that helped to improve business logic management in software industries. Services encapsulated data and business logic and they contain a management component, meaning that they can be perceived as units in the software assembly line.

3.3 Mastering the Diversity

The soaring of mobile devices, the expansion in services, and increased data availability all brought in the **Mastering the Diversity (2010-2017)** era. A wide range of widgets, from smartphones to wearables, now interconnect and exchange data continuously forming the *Internet of Things (IoT)*. In this setting, the *Cloud Computing* information technology paradigm came into play, enabling ubiquitous access to shared resources and services over the Internet. Cloud computing [2] redefines how applications and services are deployed, providing scalable resources to serve customers quickly and effectively, using the required global and connected infrastructure as needed.

Continuous Integration and Continuous Delivery (CI/CD) [38] provides development teams with the automation tools and techniques necessary to decrease time to market, giving a rapid software quality feedback to developers. The *DevOps* concept,[4] based on CI/CD, is considered the evolution of the agile methods thread. It focuses on the collaboration between development and operations staff throughout the development lifecycle, making operators to use the same techniques harnessed by developers to ensure their systems work. An important advance in both development and operating systems is *Containers technology* [28], which enables developers and operators to set up isolated boxes where applications are run both in development and production stages, thus reducing problems in deploying applications and services. This technology allowed software design to evolve into microservices architecture [3], connecting suites of independently deployable services that can work together.

The soaring levels of available services have led to the *XaaS* concept (*everything-as-a-service*), [16] which evolved as a generalization of the services provided in Cloud Computing (i.e, next element in the thread). XaaS became established not only as a way for providers to offer services, but also, from a strategic perspective, as a means for software companies to access up-to-date technology that is available as services, hence reducing expenditure on service consumption as well as indicating the level of cloud adoption. Set also the strategic level, two important areas of interest can be timelined in this era: *Artificial Intelligence methods* adoption and *Big Data applied to SE*. SE should deal with decision-making processes at the strategic level throughout the lifetime of a software product. Consequently, SE can be considered a knowledge-intensive process and therefore be framed within the AI domain. Furthermore, if a portion of expert knowledge was modeled and then was incorporated into the SE lifecycle (and into the tools that support it), it would greatly benefit any development process. Several software development and deployment processes have already seen the use of AI algorithms, such as predictive models for software economics and risk assessment, or the search methods for finding "good-enough" solutions to large-scale SE problems caused by their computational complexity. For instance, Search Based SE [22] has been applied to almost all SE activities, being software testing the most prolific one due to their importance for collaborative development.

A second recent "disrupt" to SE theory and practice is the widespread avail-

ability of Big Data methods that extract valuable information from data in order to use it in intelligent ways, such as to revolutionize decision-making in businesses, science and society. This may lead to radical methods for overcoming SE problems as well as unprecedented opportunities. Huge datasets can now be stored and managed efficiently on cloud databases, providing the source for Big Data applications. Many sources, such as forums, forges, blogs, Q&A sites, and social networks provide a wealth of data that can be analyzed to uncover new requirements. They provide evidence on usage and development trends for application frameworks on which empirical studies involving real-world software developers can be carried out [36]. In addition, real-time data collected from mobile and cloud applications is being analyzed to detect user trends, preferences and optimization opportunities. Diversity in these emerging areas of interest has greatly impacted SE, forcing it to adopt a multidisciplinary standpoint (**the multidisciplinary approach** milestone), where knowledge gathered from diverse disciplines should be embedded in the SE processes leading to an open SE era governed by knowledge and defining the current **Mastering the Knowledge era (2018-)**.

4 Multidisciplinary Trends. Towards Mastering the Knowledge era

Given that today is yesterday's tomorrow, it is time to glimpse the movements that, in our opinion, will stay in power in the Software Engineering field during the next years. Society, software developers and leading technologies are the sources feeding the mainstream in which the three tributaries converge: connectivity, artificial intelligence and security. We strongly believe that these trends can help software engineers make advances in the software development lifecycle, either at product, manufacturing or strategic levels. Without further ado, let us discuss each of these trends in turn.

4.1 Connectivity

Systems are becoming more and more complex, and at an even faster rate than before. You simply need to look at the variety of objects that are now connected via wearables and appliances, from mobile devices to smart cars and homes. These devices have to be able to work properly in places with both good and bad connectivity. Not only appears the need of interconnection at gadget level, but also at software level.Software engineers must focus on developing reliable software applications in case of temporary connection loss; development process supported by connected devices and applications cannot bring the development to a halt in case of eventual connectivity problems; and new formats and protocols for sharing data and linking behaviour between connected services will be addressed. These are not new issues, in the same way as Model-Driven Engineering, Service-Oriented Architectures, IoT, Cloud and CI/CD are not either; indeed, connectivity has been addressed yet, at least partially, by all of

them. However, this diversity of solutions should be approached from an SE unified point of view, thus, the SE challenge is to improve continuous abstraction and integration of the numerous platforms, technologies and services that will be increased in a common interconnected ecosystem. With this in mind, SE should focus on achieving agreed faster delivery of the software quality process in response to the needs of software availability and innovation.

4.2 Artificial Intelligence

Software Engineering, as any other business that wants to stay relevant, needs to adopt AI. Nowadays, data are everywhere and the software business is no exception. Data are gathered at every software stage, from requirements to maintenance. In addition, it is almost certain that for any software decision, an AI method can be found that can provide valuable help at the time of (when) making it. Search-Based Software Engineering has provided the first insights into this assertion, as it applies search-based optimization to SE lifecycle problems. Moreover, it should be pointed out that uncertainty reasoning and classification/prediction in AI have also provided assistance to software engineers in modeling software reliability and project planning, respectively. Now, in the 21st century, the big challenge is to incorporate data into software project development and to evolve towards intelligent SE automation, so that machines carry out software engineering activities as well as humans do [17].

4.3 Security

Recently, security flaws have been found in computer processors or their software applications that allow hackers to steal sensitive data without the users knowing, such as the soldiers' smartphones [20].Other vulnerabilities have likewise affected the National Health Service and several electoral campaigns in different countries [5]. It is therefore essential that, as the guarantor of software safety and quality, SE faces the challenge of ensuring that security vulnerabilities are not introduced during software development. The first attempts at this can be found in the Open Web Applications Security Project, where software security is assessed and knowledge-based documentation concerning application security is issued. Great efforts are being carried out to automate security vulnerability checking in the software lifecycle. As example, GitHub started to report security vulnerabilities in project dependencies [21] quite recently. However, that is only the beginning of a wider and huge effort that must be addressed in the following years.

Acknowledgements

This research has been financed by the Spanish Ministry of Economy and Competitiveness under project TIN2016-77902-C3-3-P (PGM-SDA II project) and

partially supported by Data, Knowledge and Software Engineering (DKSE) research group (TIC-181) of the University of Almería, the Agrifood Campus of International Excellence (ceiA3).

References

[1] M. W. Alford. A requirements engineering methodology for real-time processing requirements. *IEEE Transactions on Software Engineering*, (1): 60–69, 1977.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010. ISSN 0001-0782. doi: 10.1145/1721654.1721672.

[3] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, 33(3):42–52, May 2016. ISSN 0740-7459. doi: 10.1109/MS.2016.64.

[4] L. Bass. The Software Architect and DevOps. *IEEE Software*, 35(1):8–10, January 2018. ISSN 0740-7459. doi: 10.1109/MS.2017.4541051.

[5] H. Berghel. Malice Domestic: The Cambridge Analytica Dystopia. *Computer*, 51(5):84–89, May 2018. ISSN 0018-9162. doi: 10.1109/MC.2018.2381135.

[6] T. Berners-Lee. WWW: Past, present, and future. *Computer*, 29(10):69–77, 1996.

[7] R. E. Biehl. Six Sigma for software. *IEEE Software*, 21(2):68–70, 2004.

[8] G. Booch. Object-oriented development. *IEEE transactions on Software Engineering*, (2):211–221, 1986.

[9] G. Booch. Software archeology and the handbook of software architecture. In *Workshop Software Reengineering*, volume 126, pages 5–6, 2008.

[10] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Professional, 1998.

[11] P. Bourque, R. E. Fairley, et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0.* IEEE Computer Society Press, 2014.

[12] A. Brennecke and R. Keil-Slawik. History of software engineering. In *Position Papers for Dagstuhl Seminar*, volume 9635, 1996.

[13] F. Brooks and H. Kugler. *No silver bullet - Essence and Accident in Software Engineering.* April, 1987.

[14] O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare. *Structured programming*. Academic Press Ltd., 1972.

[15] I. M. del Águila, J. Palma, and S. Túnez. Milestones in software engineering and knowledge engineering history: A comparative review. *The Scientific World Journal*, 2014, 2014.

[16] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu. Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 621–628, June 2015. doi: 10.1109/CLOUD.2015.88.

[17] C. Ebert and S. Counsell. Toward software technology 2050. *IEEE Software*, 34(4):82–88, 2017. ISSN 0740-7459. doi: 10.1109/MS.2017.100.

[18] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. ISBN 0131858580.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 0-201-63361-2.

[20] D. Guerra. How to manage personal device risk. *Risk Management*, 64(11):8–10, 12 2017. URL `https://search.proquest.com/docview/1973343407?accountid=14477`.

[21] M. Han. Introducing security alerts on github, Nov 2017. Retreived July 2018 from https://blog.github.com/2017-11-16-introducing-security-alerts-on-github/.

[22] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833 – 839, 2001. ISSN 0950-5849. doi: https://doi.org/10.1016/S0950-5849(01)00189-6.

[23] M. A. Jackson. *Principles of program design*, volume 197. Academic press London, 1975.

[24] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Professional, 1999.

[25] V. Kozaczynski and J. Q. Ning. Component-based software engineering. In *Proceedings of the 4th International Conference on Software Reuse (ICSR '96)*, page 236. IEEE, 1996.

[26] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff. A brief history of the internet. *ACM SIGCOMM Computer Communication Review*, 39(5):22–31, 2009.

[27] M. S. Mahoney. Finding a history for software engineering. *IEEE Annals of the History of Computing*, 26(1):8–19, 2004. ISSN 10586180. doi: 10.1109/MAHC.2004.1268389.

[28] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), Mar. 2014. ISSN 1075-3583.

[29] B. Meyer. Reusability: The case for object-oriented design. *IEEE software*, 4(2):50, 1987.

[30] P. Naur and B. Randell. Software engineering. *Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968*, 231, 1969. URL `https://ci.nii.ac.jp/naid/10029650767/en/`.

[31] Object Management Group. MDA Guide Version 1.0.1. OMG document: omg/2003-06-01, 2003. Retreived July 2018, from http://www.omg.org.

[32] L. Osterweil, C. Ghezzi, J. Kramer, and A. Wolf. Determining the Impact of Software Engineering Research on Practice. *Computer*, 41(3):39–49, 2008. ISSN 0018-9162. doi: 10.1109/MC.2008.85.

[33] D. L. Parnas. Software aging. In *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*, pages 279–287. IEEE, 1994.

[34] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber. Capability maturity model, version 1.1. *IEEE software*, 10(4):18–27, 1993.

[35] R. S. Pressman. *Software engineering: a practitioner's approach*. McGraw Hill Book Company, 8th edition, 2015.

[36] F. Qi, X.-Y. Jing, X. Zhu, X. Xie, B. Xu, and S. Ying. Software effort estimation based on open source projects: Case study of Github. *Information and Software Technology*, 92:145 – 157, 2017. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2017.07.015.

[37] D. T. Ross and K. E. Schoman. Structured analysis for requirements definition. *IEEE transactions on Software Engineering*, (1):6–15, 1977.

[38] M. Shahin, M. A. Babar, and L. Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017. doi: 10.1109/ACCESS.2017.2685629.

[39] I. Sommerville. *Software engineering*. New York: Addison-Wesley, 2010.

[40] L. Tripp and J. Fendrich. Taxonomy of software engineering standards: A development history. *Computer Standards & Interfaces*, 6(2):195 – 205, 1987. ISSN 0920-5489. doi: https://doi.org/10.1016/0920-5489(87)90059-6.