Software Engineering: The Discipline

Raul-Andrei Ariton

Last updated May 21, 2024

Contents

1 What is software engineering?

IEEE[1] defines software engineering as

"The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software".

Software engineering is not just *designing and developing software*. It is rather an *engineering approach* to develop software[1].

2 Activities in software engineering

Denning in [2] describes three major activities (or *paradigms*) carried out in all areas of computer science: **theory**, **experimentation** and **design**.

2.1 Theory

Theories offer common explanations and understanding of basic concepts and underlying mechanisms, and thus they are essential for knowledge of passing trends and the reason for their manifestation[3].

Although the usefulness of theories in software engineering is a subject of debate, as [3] very well explains, *building theories is a means to go beyond the mere observation of phenomena, and to try to understand* why *and* how *these phenomena occur*.

2.1.1 Topics of research

A systematic analysis on research in the area of software engineering conducted in [4] suggests that the main topics of research are:

- Software and software systems concepts, particularly:
  - Methods and techniques, such as design patterns, parallel processing, etc.;
  - Software development tools, for instance compilers, debuggers and linters;
  - Software development life cycle stages, software engineering process models (methodologies, e.g. *agile, RAD, waterfall, DevOps etc.*);
  - Software quality assurance, through benchmarking to measure performance and through implementation of fault tolerance;
- Software and software system management concepts, for instance:
  - Development and use of software measurement methods;
  - Project and product management, including the process of risk management;
- Computer networks and distributed systems;

[1] the Institute of Electrical and Electronics Engineers.

- Database organization and information retrieval;
- Problem-solving methodologies; and others.

2.1.2 Research methods

[5] identifies research models used in the field of software engineering. These are:

- The **Scientific method**, where the world is observed, and a model, based on the observation, is built.

  This method is typically used in applied areas, where simulations must be made in order to evaluate performance, e.g. telecommunication networks[6].

- The **Engineering method**, where current solutions are studied, possible modifications are proposed, and then evaluated.

  Engineering methods may also use simulations as a means for conducting experiments. The engineering method of research is the most dominant in the area.

- The **Empirical method**, in which a model is proposed, then applied to case studies for assessment and validation.

  This method has been gaining popularity over the past decade. In fact, empirical software engineering has grown so popular that it is now considered a sub-field of software engineering.

- The **Analytical method**, in which a formal theory or a set of axioms is proposed, then results are derived, which can then be compared with empirical observations.

  This method is tradionally used in more theory-intensive, formal areas of computer science, e.g. algorithms[6].

2.2 Experimentation

In software engineering, experimentation is a means of evaluating and choosing between various methods, techniques, languages (*formal, programming*) and tools before using them in later software development processes. It also enables understanding, identifying and verifying relationships between different factors, different variables in a process[6].

2.2.1 Purpose of experimentation

Experiments may be used in various contexts, such as[6]:

- to confirm theories;
- to confirm common conceptions about a topic;
- to explore relationships between different factors and variables;
- to verify the accuracy of models;
- to validate measures.

Through experimentation, claims (*i.e. theories*) can be investigated in order to determine in which situations they hold, as well as to determine contexts in which certain standards, methods and tools are suitable to use.

2.2.2 Classification of experiments

Experiments may be *human-oriented*, where humans apply different treatments to objects, or *technology-oriented*, where different tools are applied to different objects[6].

- An example of a human-oriented experiment is one where multiple subjects are asked to inspect a piece of code, as an evaluation of inspection methods.
- An example of a technology-oriented experiment is one where multiple sorting algorithms are applied to the same sorting problem, as a means of assessing their corectness and performance.

2.2.3 Theory in experimentation

Experiments are suitable for *causal description*, i.e. "describing the consequence attributable to deliberately varying a treatment[3]".

Experiments, however, only provide a *description* rather than an *explanation*. The desire for a *causal explanation* leads to the development of theory.

Theories clarify the mechanisms through which, as well as the conditions under which a phenomenon occurs.

The use of theory, however, is scarce in software engineering, as concluded by [3].

Perhaps this is one of the reasons many exclude the field when mentioning engineering sciences, as well as the reason why it is still a maturing field.

Moreover, with the substantial growth empirical software engineering in the past decade, usage of empirical research methods is becoming more of a common practice, and thus less and less theoretical elements are used in research, in favor of simulations, case studies and other techniques.

As [6] very well states, "*theory building in software engineering should be developed, in order for the field to develop into a mature field of science*".

2.3 Design

The design activity in software engineering has to do with *designing* as well as *implementing/building* software that support work in given organizations or application domains[2].

Since software engineering is primarily a design specialty, the majority of those in the area of software engineering deal with design rather than theory and experimentation.

We shall separate the design activity into *design* and *development*, as they represent two separate fields of software engineering thus their distinction is clear.

Designing software. *Software design is the purposeful conception of a [software] product and the planning for its production*[7].

It is concerned with both the internal and the external design of a software system.

External design of a software product. External design defines the features of the product (*what the product can do*) and how it interacts with the users and its environment. It determines how well the product satisfies the client's specifications (and thus the user's desires)[7].

Internal design of a software product. Internal design defines how the aforementioned external features work, and determines many attributes of the product's functionality, such as its performance and reliability.

A critical attribute determined by the internal design of a software product is its evolution: how it can evolve to meet users' changing needs[7].

Software architecture. The architecture of a software system is a set of fundamental design choices, which Taylor in [7] calls the "*principal decisions*". The architecture of a software system is fundamental to its conceptual integrity.

Developing software. Software development is defined by [8] as a sub-field of software engineering that deals with creating, designing[2], deploying and maintaining software.

Compared to software engineers, software developers have a less formal role in the process of software engineering. They are involved in the practical side of things; working with various teams involved in specific project areas to transform specifications into features, and conducting software testing and maintenance.

A common misconception. As one reading this article may have realized, software engineering is not just programming, testing and debugging. Before the software developers even begin coding, a lot of thought is put into the design of the product (*a work of software designers and software architects*), as well as the intricate mechanics with which all of the system's modules communicate with one another.

Thus, the work of a software developer is not comparable to that of a software engineer, but rather *software development is a subset of software engineering*, *a stage in the process of software engineering*.

[2] not to be confused with the designing that software designers deal with. Software developers design the software system at a much more lower level.

3 Open Problems in software engineering

[9] lists a few open problems in the field of software engineering:

- Is it possible to simulate a human mind? Can a machine become conscious?

  Simulations are not a rare thing in software engineering, as they are used in research. Simulating a human mind is not as trivial, though. This problem dates back to the 1950s, with the Turing test. Since then, many attempts have been made, which in turn motivated many advancements in fields such as artificial intelligence in combination with software engineering. With the many chatbots available today based on large language models, one can only wonder how close we are to achieving a *full* simulation of the human mind.

- Systematic detection and prediction of software system defects.

  This problem focuses on inventing tools which facilitate early software defect prediction. While there already exist such tools (*program linters, code analyzers, etc.*), the many types of errors that happen in the process of software engineering, *that are not necessarily exclusive to the software development stage*, lead to the demand for solutions, to avoid errors in the final release of the product.

  The field of artificial intelligence has brought advancements at exponential rates in the past few years, and the industry seeks to implement such powerful capabilities in the software engineering process. Such tools are already available and popular in the software development community (e.g. AI coding assistants), however studies are being made to implement artificial intelligence in the more formal, theoretical parts of software engineering (design, architecture, project management, etc.), see for example [10].

- Software testing cannot always be formally proved[11].

  Turing *et al.* proposed the Halting problem, proving that, given a set of program instructions and an input, there is no algorithm which can determine if the program will stop (i.e. *halt*) given that input.

  Furthermore, Gödel proved with his incompleteness theorems that a set of axioms, that define a set of rules and logic (i.e. an *algorithm*) can contain contradicting axioms, or axioms which cannot be proved. Essentially, even if one uses formal logic or formal methods, which contain rules and axioms, to verify the corectness of a program, it doesn't guarantee that all the rules do not contradict one another. Therefore, verification of any program cannot be done by purely using formal methods.

  Therefore, software testers only have their human intuitions to verify the corectness of a program. Human intuition is based on experience, meaning that, the more experience one has, the easier it gets to recognize mistakes and fix them while they can.

  In conclusion, the problem of software testing will likely remain unsolved. One cannot rely on formal methods to test any type of program, as this has been mathematically proven to be unreliable.

4 Important figures

Pioneers in software engineering

This section lists people who have contributed to fundamental software engineering concepts and inventions. Sources

- Mary Shaw

  Professor at Carnegie Mellon University. IEEE fellow and member of ACM.

  Received the Outstanding Research Award from ACM, for her "significant and lasting software enineering research contributions through he development and promotion of software architecture". Moreover, awarded with the National Medal of Technology and Innovation by US President Barack Obama.

  Research interests: software architecture, programming systems, abstraction technique, program organization for quality human interfaces, reliable software development, software evaluation techniques

  `mary.shaw@cs.cmu.edu`

- Robert L. Glass

  ACM, IEEE fellow and emeritus editor-in-chief of JSS.

  Known for works on software design quality measurement.

  Research interests: software engineering in practice, software quality, software maintenance.

  `rlglass@acm.org`

- Bertrand Meyer

  Porfessor emeritus of Software Engineering at ETH Zurich Professor of Software Engineering and Provost at Constructor Institute

  One of the earliest and most vocal proponents of object-oriented programming. Additionally known for designing the Eiffel object-oriented programming language.

  Research interests: high quality software production, automated testing techniques, automated techniques for correcting program bugs, automated proofs and applications of formal methods and more.

  `bertrand.meyer@inf.ethz.ch`

- Grady Booch

  ACM, IEEE, IBM fellow. Chief Scientist for Software Engineering at IBM

  Known for major contributions in object-oriented programming, component-based software engineering as well as being one of the creators of UML.

  Research interests: development of cognitive systems, large-scale (of national importance) software architecture.

  `egrady@booch.com`

- Ivar Jacobson

  CEO of Ivar Jacobson international, a company that prives software consulting, coaching and training solutions implementing large scale software. Flourishing career in both academia and business.

  Known for major contributions in object-oriented programming, component-based software engineering as well as the creation of UML.

  Research interests: using software development methods and tools efficiently, in an agile way, components, component architecture

  `ivarjacobson.com`

- Michael A. Jackson

  ACM, IEEE fellow. Independent software consultant on specification and design of software-intensive systems. Visiting research professor at the Open University in the UK.

  Known for contributions to modular programming, and most importantly for developing the Jackson Structured Programming method, as well as the Jackson System Development (JSD) methodology. Additionally, Jackson has developed the problem frames approach to software requirement analysis.

  Research interests: problem frames approach, sequential program design, information system development

  `jacksonma@acm.org`

- Larry Constantine

  ACM, IEEE fellow. Professor at the University of Madeira's Department of Mathematics and Engineering.

  Known for introducing data flow diagrams as a software analysis and design tool, as well as popularizing software structure analysis and design.

  Research interests: safely-critical interaction, model-driven software design.

  `lconstantine@uma.pt`

- Margaret E. Hamilton

  Developed the on-board flight software for NASA's Apollo program and was the director of the Software Engineering Division of the MIT Instrumentation Laboratory. Invented the term "software engineering".

  Research interests: *ultra-reliable* software systems, error detection and recovery, formal theory, formal language, operating systems.

  `mhh@htius.com`

Active researchers

[12] has provided a very useful assessment of software engineering scholars which was used throughout this section. Additional information was provided by various sources online (Wikipedia, Google Scholar, Institution pages, etc.).

- Ahmed E. Hassan

  Industrial Research Chair in Software Engineering at the Natural Sciences and Engineering Research Council of Canada. Leads the Software Analysis and Intelligence Lab at Queen's University of Canada.

  Research interests: techniques for large-scale complex software system production, maintenance and evolution, mining software repositories, software evolution and architecture, performance engineering, capacity engineering, debugging and monitoring of distributed systems.

  `research.cs.queensu.ca/home/ahmed/home/`

- David Lo

  Professor and Director at the School of Computing and Information Systems at Singapore Management University. Lead of the Software Analytics Research group.

  Research interests: (as part of the Software Analytics Research group) analysis of different kinds of software artefacts (code, execution traces, bug reports, developer networks), transformation of passive software engineering data into automated tools that improve system reliability, security, performance as well as increase developer productivity.

  `davidlo@smu.edu.sg`

- Paris Avgeriou

  Professor (chair of software Engineering) at the University of Groningen and head of the SEARCH research group

  Research interests: (as part of the SEARCH research group) software architecture, software maintenance and evolution, technical debt, empirical software engineering

  `paris@cs.rug.nl`

- Tsong Yueh Chen

  Professor in the Department of Computer Science and Software Engineering at the Swinburne University of Technnology, Australia.

  Research interests: software testing, software analysis, debugging, test case generation, metamorphic software testing, adaptive random testing, program repair

  `tychen@swin.edu.au`

5 Important publication venues (journals, conferences)

Abbreviation	Name
TSE	IEEE Transactions on Software Engineering
JSS	Journal of Systems and Software
SW	IEEE Software
IST	Information and Software Technology
TOSEM	ACM Transactions on Software Engineering and Methodology
JSEP	Journal of Software: Evolution and Process
STTT	International Journal on Software Tools for Technology Transfer
EMSE	Empirical Software Engineering
ICSA	International Conference on Software Architecture
ASE	Automated Software Engineering Conference
ICSE	International Conference on Software Engineering
ISTTA	International Symposium on Software Testing and Analysis
EASE	International Conference on Evaluation and Assessment in Software Engineering
FSE	International Symposium on the Foundations of Software Engineering
ESEM	International Symposium on Empirical Software Engineering and Measurement
ICSME	International Conference on Software Maintenance and Evolution
FASE	Fundamental Approaches to Software Engineering

6 Connection to other areas

Connection to other areas of Computer Science

Software engineering can be described as an application of all computer science fields.

One way or another, during the process of software development, competencies and knowledge from another field of computer science will be used, whether that be algorithms and data structures or human-computer interaction. Some fields have more impact on the process of software development than others. A list follows.

- Computer Engineering

  At times, low-level knowledge is required; meaning the software engineer must modify the product design accordingly, in order to match the capabilities of the average machine running the product.

  For instance, a software product for mobile phones is designed much differently than one made for computers, due to their different architectures and computing limits.

- Mathematics

  As mathematics is closely related to computer science, it is also related to a field of it.

- Data Science and Engineering, Data Analysis

  At times, the development of a software product requires statistical research and data collection. More commonly, with the release of a software product to users, data collection begins, which is then analysed by software engineers to use in the maintenance, improvement phase of the process.

- Database Engineering

  Designing data-intensive applications is not trivial. Data retrieval, modification and storage are all processes used in today's software systems, and thus they must be conducted efficiently.

Connection to other areas outside of Computer Science

The use of technology and software has been implemented in many domains. Of course, to develop a software product for use in a particular fields (e.g. medical, manufacturing, agriculture, etc.), a software engineer will require competences related to that field, in order to understand the client's needs and specifications better, and to optimize user experience.

Software engineering as a discipline encompasses competences outside of computer science, thus we shall list related disciplines with competences similar to those in software engineering, as presented in [13] as well as [2]

- Project management, Team management, Management Science

  The development of a software product is a big project, and during the development lifecycle of software, software engineers must be capable to manage a

team effectively. The primary goal of a software engineer is to have a quality software product developed as soon as possible.

- Psychology, Cognitive and Behavioral sciences

  During the design phase of a software product, a software engineer must keep the user in mind: Is this interface simple enough for the user? In cases where the software product is used in emergency situations, is its interface designed well so that it can be used in such situations of increased stress? and many other questions.

- Economics

  The cost of the software development process must also be kept in mind. Resources, in the form of people (developers, designers, etc.) as well as in the material form (machines, power supply, etc.) can become costly if they are not taken into consideration.

7 Software Engineering at UVT

The West University of Timisoara offers a Master's degree in software engineering[3], with a duration of two years. Below follow the competences which one is expected to acquire by following this study program, in a summarized manner, as described by the syllabus.

Professional competences

- Knowledge of the software development lifecycle as well as methodologies for software development, particularly agile methods. The ability to propose the suitable method and to contribute to its improvement
- Software product development by systematically applying development processes.
- Collection, documentation and analysis of client needs and creation of specifications for the software product. Knowledge of definition and negotiation of quality attributes in software architecture
- Analysis of data and extraction of information using data analysis tools, including the cases of large data sets.
- Analysis, projection, documentation and evaluation of the software system architecture, data management, behavior and human-computer interaction. Knowledge and use of corresponding modelling tools and languages (e.g. UML[4])
- Building superior quality software systems, according to specifications and coding standards.
- Development and application of software product verification and validation plans, as well as quality assurance
- Use of specific tools for the different phases of the software development process as well as tools for team collaboration
- Creation of technical documentation and usage documentation
- Analysis and development of distributed systems

Transversal[5] competences

- Ability to efficiently plan and organize workload
- Critical analysis of results
- Respecting ethical norms specific to the domain of activity
- Ability to communicate and transfer knowledge
- Ability to work productively, in an interdisciplinary context, whether that be individually or as part of a team.

[3] the degree is taught in Romanian.
[4] Unified Modelling Language; used to design object-oriented software architectures.
[5] i.e. competences that are not necessarily bound to the field of software engineering.

References

[1] R. Mall, *Fundamentals of Software Engineering, Fifth Edition* (Eastern Economy Edition). PHI Learning Private Limited, 2018, ISBN: 9789388028035. [Online]. Available: https://books.google.com/books?id=-JNuDwAAQBAJ.

[2] P. J. Denning, "Computer science: The discipline," *Encyclopedia of computer science*, vol. 32, no. 1, pp. 9–23, 2000.

[3] J. Hannay, D. Sjøberg, and T. Dybå, "A systematic review of theory use in software engineering experiments," *Software Engineering, IEEE Transactions on*, vol. 33, pp. 87–107, Mar. 2007. DOI: 10.1109/TSE.2007.12.

[4] R. L. Glass, I. Vessey, and V. Ramesh, "Research in software engineering: An analysis of the literature," *Information and Software technology*, vol. 44, no. 8, pp. 491–506, 2002.

[5] R. L. Glass, "The software-research crisis," *IEEE Software*, vol. 11, no. 6, pp. 42–47, 1994.

[6] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering* (Computer Science). Springer Berlin Heidelberg, 2012, ISBN: 9783642290442. [Online]. Available: https://books.google.com/books?id=QPVsM1_U8nkC.

[7] R. N. Taylor, "Software architecture and design," in *Handbook of Software Engineering*, S. Cha, R. N. Taylor, and K. Kang, Eds. Cham: Springer International Publishing, 2019, pp. 93–122, ISBN: 978-3-030-00262-6. DOI: 10.1007/978-3-030-00262-6_3. [Online]. Available: https://doi.org/10.1007/978-3-030-00262-6_3.

[8] *What Is Software Development?* https://www.ibm.com/topics/software-development.

[9] *Unsolved problems in software engineering*, https://en.wikiversity.org/wiki/Unsolved_problems_in_software_engineering.

[10] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools," *Engineering Applications of Artificial Intelligence*, vol. 111, p. 104 773, 2022, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2022.104773. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197622000616.

[11] *Testing is an Unsolved Problem*, https://www.linkedin.com/pulse/testing-unsolvedproblem-jason-arbon/.

[12] W. E. Wong, N. Mittas, E. M. Arvanitou, and Y. Li, "A bibliometric assessment of software engineering themes, scholars and institutions (2013–2020)," *Journal of Systems and Software*, vol. 180, p. 111 029, 2021.

[13] P. Bourque and R. Fairley, *Guide to the Software Engineering Body of Knowledge - SWEBOK (version 3.0)*. IEEE Press, 2014. [Online]. Available: www.swebok.org.