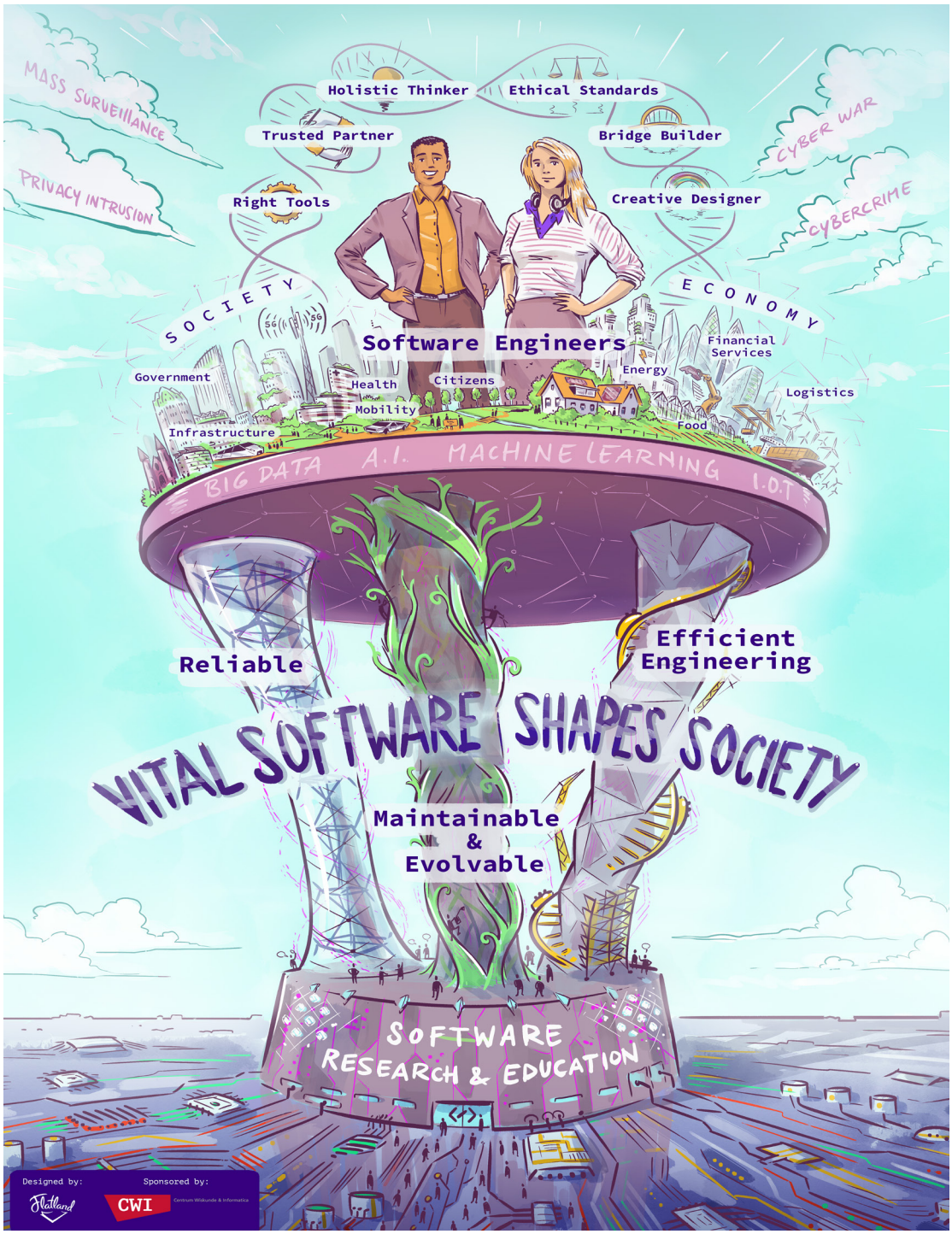
Manifesto on Software Research and Education in the Netherlands



Manifesto on Software Research and Education in the Netherlands

by VERSEN[1]

Software plays a role in almost every aspect of our daily lives and dominates corporate, political and societal discourse. Over the last decade, we have seen a dramatic and broad technological and economic shift in which software systems take over large parts of the economy. This was foreseen already in 2011 by Marc Andreessen in his essay Why Software Is Eating the World,[2] and is not likely to change any time soon (NESSI, 2014[3]). The Netherlands has the potential to be at the forefront of this shift if it develops a sustainable strategy in software research and education.

Producing industrial software is not merely a matter of writing code; to be of quality it requires a strong research foundation. Modern software used in cars, airplanes, medical robots, banks, healthcare systems or the public sector, comprises millions of lines of code. To produce such software, many challenges have to be overcome. How do we develop this software and simultaneously manage its complexity? How do we ensure the correctness and security of this software, as human well-being, economic prosperity and the environment depend on it? How can we guarantee that software is maintainable and usable for decades to come? How can we construct software efficiently and effectively?

Despite the fact that software impacts everyone everywhere, the effort that is needed to make this software reliable, maintainable and usable for longer periods is routinely underestimated, both by developers and their managers. As a result, we see news items every day about expensive software bugs and over-the-budget or failed software development projects. ICT systems of e.g., tax authorities, banks, hospitals, and high-tech companies, contain crucial legacy components of at least 30 years old, which makes maintenance difficult and expensive, and sometimes even impossible.

The Netherlands aspires to take the lead in digitization: this gives our Country the opportunity to influence technological developments in a way that is beneficial for both citizens and the economy (NLDigitaliseringStrategie[4]). In order to achieve this goal, a high level of research and education in software is absolutely essential. The industry of software(-intensive) systems is crucial for our economy (5,5% of the total workforce in the Netherlands works in ICT, https://dutchdigitaldelta.nl[5]).
Still, companies are starved for qualified software engineers. Our graduates enable the success of innovative companies such as Adyen, ASML, Bol.com, booking.com, CGI, Elastic Search, Océ, Philips, and Transavia. Software research in the Netherlands is excellent but under siege due to the steep increase in student numbers and teaching load in recent years. A serious investment in software research and education by the Dutch government is urgently needed.

Edsger Dijkstra, the most influential Dutch computer scientist, stated in his Turing award acceptance speech in 1972 that software construction is an intellectual challenge without precedent in the cultural history of mankind. Since then, research on computer science and software engineering has produced many powerful methods, techniques, theories, and tools for building software systems that are correct, secure and maintainable. It has also led to a deep understanding of how people build and evolve software systems. However, with the lightning-fast technological change and churn, many new challenges for software have come into existence. This manifesto discusses these new and upcoming major fundamental challenges for software research and education.

In doing so, it does (so far) mainly focus on the fundamental challenges for the discipline itself. However, to address these challenges we envisage an evolution of the field where research synergies with other disciplines, and across sectors, are not only mutually beneficial but instrumental for a significant scientific advancement and a true positive impact on society. We hope this Manifesto starts by inspiring other researchers to reach out and provide feedback in this (and many other) directions.

---

[1] www.versen.nl This is version 1.0 of the VERSEN Software Manifesto. We see this as a living document, and we appreciate all your feedback, which we will use as input for a next version of the document (which we plan to start discussing end of 2020).

[2] Why Software Is Eating the World > https://www.wsj.com/articles/SB10001424053111903480904576512250915629460

[3] NESSI, 2014 > http://www.nessi-europe.eu/Files/Private/NESSI_SE_WhitePaper-FINAL.pdf

[4] NLDigitaliseringStrategie > https://www.rijksoverheid.nl/documenten/rapporten/2018/06/01/nederlandse-digitaliseringsstrategie

[5] https://dutchdigitaldelta.nl/

Software Research Challenges

We position the challenges for software research in three clusters: (1) the reliability of software systems, (2) the efficiency and effectiveness of the construction of software systems, and (3) the long-term maintainability of software systems.

Challenges in Software Reliability

One fundamental problem of software research is how to ensure the reliability of software systems. In the first place, this entails that a software system should faithfully realize its intended purpose. Moreover, reliability encompasses aspects such as security, performance, energy-efficiency, and usability. Statistics show that all software contains errors, and the earlier they are detected (or prevented), the better. Many verification technologies have been developed to improve software reliability, such as model checking, theorem proving, and monitoring systems, but applying them on a large scale to modern software systems remains a challenge. Concrete challenges that need significant research are:

[Reliability by construction] Ideally, software is developed with reliability in mind from the start. We need to investigate if it is possible to automatically construct (efficient) software from a high-level, domain-specific description of the software's intended use, in such a way that it is correct and reliable by construction.

[Scaling of verification techniques] Verification technologies are applied to (parts of) restricted systems, not to full software frameworks. The question is how to scale existing techniques to such large settings, and how to make the verification feasible on such a large scale. By using manual intervention it is possible to verify complex software, but as it stands this is forbiddingly inefficient.

[Reliable software on modern hardware] New promising hardware such as multi-core systems, GPUs, and heterogeneous systems are not used as widely as they could be, because the inherent parallelism makes it even harder to make the software reliable. As a consequence, software is often still run on the energy-inefficient classical von Neumann hardware, and avoids parallelism. We need to understand how to develop reliable software also for such post-von-Neumann architectures.

[Robustness against unexpected uses] Software can be used in many ways that were not initially foreseen. This can be good, where users find creative ways to use the software. But this can also be bad, transforming useful software into economic or security hazards. Where human users can often find ways around such problems, computer abusing the services of other computers may lead to long lasting and hard to observe misbehaviour.

[Software that fulfils social, economic, and environmental sustainability needs] Software has a pervasive role in the digital society. It is expected to support the needs of citizens, the industry sectors and the society at large without doing unnecessary harm. We are, however, lacking a conceptual framework on the fundamental inbedding of software into society that guarantees that software does what society needs. This regards, for instance, the protection of privacy, the efficient use of resources (including energy), the stability of software over time and in the presence of change, the traceability of decisions about- and by software, responsibility of software, etc.

Challenges in the Efficient Engineering of Software

The demands for software development and change are higher than we can humanly deliver. In other words: software engineering is exceeding human scale in terms of velocity, volume and variety. Therefore, we need to work smarter, not harder, and to achieve this, we need to address the following concrete research challenges:

[Shorter development feedback loops] To make software development more effective and efficient, we need to create techniques and tools to analyze requirements, software artifacts (source code, architecture, models, documentation, etc.) and contextual data (version history, the software ecosystem, dependencies). We also need to understand how to make optimal systems for continuous delivery in a safe, verified, secure, and responsible software supply chain. So far, this need has been approached from a purely technical standpoint, whereas it depends on multiple aspects, e.g., organizational culture, social interaction, and economic drivers.

[Tool-supported software development] While the current software development process is increasingly supported by tools, it still requires substantial manual activity. Thus, we need to understand how to create efficient tools that support dedicated, new software languages, development environments and low code platforms, and which help us reduce the size and complexity of software.

[Empirical software engineering] Decisions made during software development should be based on solid empirical evidence. This calls for identification of the determinants for effective and productive software engineering practices. To this aim, intensive experimentation is imperative, hence calling for large-scale collaboration between research and practice.

[Automated software engineering] We must bring automation in software engineering to the next level, e.g. by leveraging the advances in AI for augmented software engineering, test engineering, and search-based software engineering.

[Safe and dependable software ecosystems] We must ensure that the organizations that develop critical systems do so in a transparent and dependable manner, independent of their country of origin, quality of education, or economic background. Existing systems should expose verifiable, testable, and transparent behaviors, and the origins of the software artifacts within the ecosystems should be made explicit.

Challenges in Software Maintainability and Evolution

Software not only has to be created, it also has to be maintained and adapted over time. If this is not effectively done, the software becomes too complex, and maintenance and evolution become too expensive, until they are no longer sustainable. We must break this vicious cycle, and find new ways to create software that is long-living and that can be cost-efficiently evolved and migrated to new technologies. Concrete challenges that require significant research in software sustainability include:

[Organisations lose control over software] As software complexity is increasing, it becomes increasingly difficult for organizations to understand which part of their software is worth maintaining and which parts need to be re-developed from scratch. Therefore, we need methods to reduce the complexity of software worth maintaining, and extract domain knowledge from existing systems as part of the re-development effort. This also relates to our inability to monitor and predict when software quality is degrading, and accurately estimate the costs to repair it. Consequently it is also hard to prioritise which maintainability issues to deal with first.

[Modern software cannot cope with continuous and unpredictable change] Sustainability of the software is often an afterthought. This needs to be flipped around, i.e. we need to be able to design 'future-proof' software that can be changed efficiently and effectively. How can the software facilitate change by design?

[Software sustainability is not a purely technical challenge] There are several socio-technical aspects that help, or hinder, software change. Deep down, many software maintenance problems are not technical but people problems. We need to be able to organize the development team (group, community, etc.) in such a way that it embraces change and facilitates maintenance and evolution, not only immediately after deployment of the software, but for the decades that follow.

Software Education Challenges

There is an urgent societal need for people with expertise in developing quality software.

A fundamental barrier to addressing this need is that, especially in the Netherlands, the knowledge in society about computing is low. At secondary school, there is insufficient basic education in computing as there is e.g., in mathematics, physics, biology. This means that a large part of the otherwise educated society does not have the basic skills to make proper judgments or decisions related to computing that some of them undoubtedly have to make in their careers. We should start by ensuring that these people get from "unknowingly unable" to "knowingly unable" as that will do a lot of good in society.

On top of this, many current software developers do not have a computer science or software engineering background, and are sometimes unaware of software engineering problems and advanced software engineering practices. In addition, even experienced software engineers regularly need to learn new languages, frameworks, tools, or software engineering practices. The software industry in the Netherlands employs hundreds of thousands of developers, which all need further professional development and lifelong learning.

The needs for software development skills are addressed by Computer Science and related programmes for students in vocational and higher education, and postgraduate programmes and professional education services for software practitioners. The steeply rising numbers of students and developers lead to clear and urgent scaling problems. Furthermore, a lot of software skills development happens on the job, which makes it challenging to provide access to help and support. The challenge in software education is to inject a significant change in society so that:

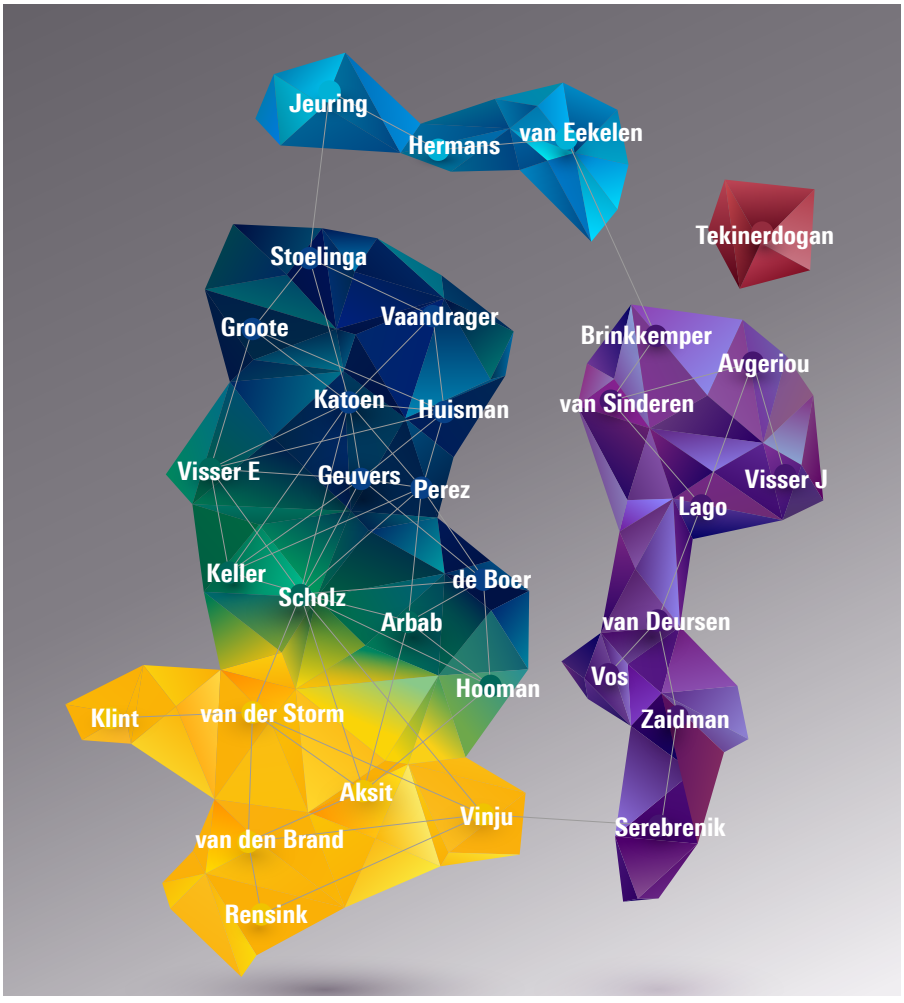
- All educated people have a proper elementary view on what software is, in the same way as they do this regarding, for instance, physics.
- We can rely on the science of software education. How can we teach students such that they become productive, creative and reliable software professionals? In which order do students learn and understand software-related concepts most efficiently. What is the most appropriate balance between, e.g., critical thinking, programming competences, discrete mathematics, logic, abstraction?
- We are able to sustainably satisfy the job market. This is possible only by increasing the number of graduates, and by attracting a truly balanced and diverse student population representative of the whole society.

To address these challenges, we need new educational practices, new tooling that provide high-quality feedback on software system quality, new ways to assess software development skills, easier access to lifelong learning. These require both pedagogical soundness, and a true integration and engagement of all relevant stakeholders (and end-users in primis) in the software development loop for continuous feedback.

The Dutch Software Landscape

The software researchers in the Netherlands are well-prepared to address the challenges identified above. The following picture provides a rough sketch of the landscape of Dutch software research. This picture is based on a review of research activities from end 2019. Lines indicate a thematic similarity between the researchers. However, we stress that there are also many collaborations between researchers with less thematic similarity, taking advantage of their complementary perspectives.

The dark-blue group of researchers works on challenges related to software reliability, while the yellow and purple groups of researchers work on the challenges related to efficient engineering of software, and software maintainability and evolution. There is a substantial number of people that work on between all these challenges (coloured green). The difference between the yellow and the purple group is mainly in how they develop their techniques: the results of the yellow group are based on programming language theory, while the purple group is more focused on observation of current software development practices. Finally, the light blue coloured group of researchers studies software engineering education, while the red group combines software research and information systems.



Software researchers

Below is a list with contact details of the software researchers in The Netherlands (researchers in italics have recently retired, but are still active).

Researchers	Research group University	Email	Website
Mehmet Akşit	Formal Methods and Tools U Twente	m.aksit@utwente.nl	Website
Farhad Arbab	Formal methods CWI Foundations of Software Technology U Leiden	f.arbab@liacs.leidenuniv.nl	Website
Paris Avgeriou	Software Engineering U Groningen	p.avgeriou@rug.nl	Website
Frank de Boer	Formal methods CWI	f.s.de.boer@cwi.nl	Website
Mark van den Brand	Software Engineering and Technology TU Eindhoven	m.g.j.v.d.brand@tue.nl	Website
Sjaak Brinkkemper	Organization and Information U Utrecht	S.Brinkkemper@uu.nl	Website
Arie van Deursen	Software Engineering Research Group TU Del	A.vanDeursen@tuedelft.nl	Website
Marko van Eekelen	Software Quality, Open University Digital Security, RU Nijmegen	marko.vaneekelen@ou.nl	Website
Herman Geuvers	Software Science RU Nijmegen	herman@cs.ru.nl	Website
Clemens Grelck	Parallel Computing Systems UV Amsterdam	c.grelck@uva.nl	Website
Jan Friso Groote	Formal System Analysis TU Eindhoven	jfg@win.tue.nl	Website
Felienne Hermans	Programming Education Research Lab U Leiden	f.f.j.hermans@liacs.leidenuniv.nl	Website
Jozef Hooman	Software Science RU Nijmegen	hooman@cs.ru.nl	Website
Marieke Huisman	Formal Methods and Tools U Twente	M.Huisman@utwente.nl	Website
Johan Jeuring	Software Technology for Learning and Teaching U Utrecht	j.t.jeuring@uu.nl	Website
Joost Pieter Katoen	Formal Methods and Tools U Twente Software Modeling and Verification RWTH Aachen (Germany)	j.p.katoen@utwente.nl	Website
Gabriele Keller	Software Technology U Utrecht	g.k.keller@uu.nl	Website

Researchers	Research group University	Email	Website
Paul Klint	Software Analysis and Transformation CWI	p.klint@cwi.nl	Website
Patricia Lago	Software and Sustainability VU Amsterdam	p.lago@vu.nl	Website
Jorge A. Perez	Fundamental Computing Science U Groningen	j.a.perez@rug.nl	Website
Hajo Reijers	Business Process Management & Analytics U Utrecht	h.a.reijers@uu.nl	Website
Arend Rensink	Formal Methods and Tools U Twente	arend.rensink@utwente.nl	Website
Sven-Bodo Scholz	Software Science RU Nijmegen	svenbodo.scholz@ru.nl	Website
Alexander Serebrenik	Software Engineering and Technology TU Eindhoven	a.serebrenik@tue.nl	Website
Marten van Sinderen	Services and Cybersecurity U Twente	m.j.vansinderen@utwente.nl	Website
Marielle Stoelinga	Formal Methods and Tools U Twente Software Science RU Nijmegen	m.i.a.stoelinga@utwente.nl	Website
Tijs van der Storm	Software Analysis and Transformation CWI Software Engineering U Groningen	T.van.der.Storm@cwi.nl	Website
Bedir Tekinerdogan	Information Technology Wageningen U	bedir.tekinerdogan@wur.nl	Website
Frits Vaandrager	Software Science RU Nijmegen	f.vaandrager@cs.ru.nl	Website
Hans van Vliet	Software and Sustainability VU Amsterdam	j.c.van.vliet@vu.nl	Website
Jurgen Vinju	Software Analysis and Transformation CWI Software Engineering and Technology TU Eindhoven	Jurgen.Vinju@cwi.nl	Website
Eelco Visser	Programming Languages TU Delft	e.visser@tudelft.nl	Website
Joost Visser	U Leiden	j.visser@sig.eu	
Tanja Vos	Software Quality Open University	tanja.vos@ou.nl	Website
Andy Zaidman	Software Engineering Research Group TU Delft	a.e.zaidman@tudelft.nl	Website

VERSEN

Science Park 123
1098 XG AMSTERDAM
Email: info@versen.nl
Webpage: http://www.versen.nl

The manifesto has been supported by CWI (visual) and University of Twente (infographics, editing, layout).

CWI

UNIVERSITY OF TWENTE.