SW Architect or SW Gardener

What is the better term?

Dirk Engel, info@engel-internet.de, Dec 29, 2021
https://github.com/dirkengel/

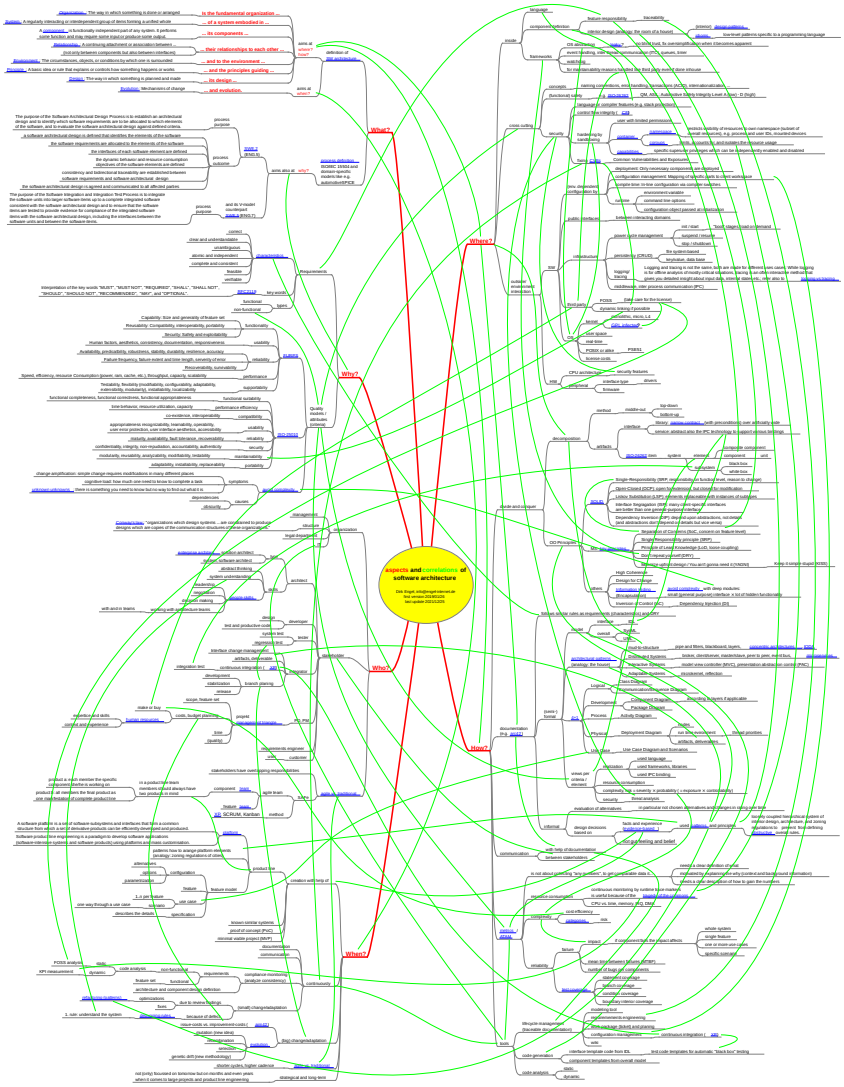
Introduction

Inspired by a SW architecture video (Wolff 2021) that mentions the gardener's annual hedge trimming as a metaphor for keeping your documentation in shape, I started some research about this thought. Not only reduced to the documentation aspect but seen in a wider context, the gardener might be a promising metaphor to explain what the SW architect's work looks like. Promising because I'm sometimes a bit unhappy about the term SW architect and the associations that come with it. Since I started with the SW architecture mind map in 2019 (Engel 2019) as an attempt to create a one-pager (well, admittedly a quite large page) about what SW architecture is all about, I realized that simply borrowing a long existing job title is far not enough to transport the idea of the SW architect tasks. Maybe there is a better name to give a suitable short description, interpretation guideline or inspiration of the SW architect's role? Is *architect* really the best match or would *gardener* or maybe something third be a better term?

Via a quick internet research, the SW gardener idea (not specifically related to SW architecture, but SW engineering in general) can be traced back to Aitchison 2018 "Are you a Software Gardener?" or Papapetrou 2015 "Software Gardening: Yet Another Crappy Analogy or a Reality?" who also announced a book named "The Art of Software Gardening" (Papapetrou 2014) which seems never to have been written. Aitchison's point is mainly that accurate estimations can't be made right in the beginning and therefore deadlines can't be promised in a way like for the construction and completion of a bridge, for instance. I guess one potential reason is that for buildings, architects can make heavy use of patterns, i.e. similar elements can be reused in a parameterized way. For example, different types of walls with different dimensions and materials, considering the required static/structural properties. In this respect, SW is more like a custom-built machine with lots of special parts and a complex dynamic behavior, way trickier compared to the static only part. For Papapetrou software is something organic and living, he even declares a SW gardener manifesto. Furthermore, he refers to Venners 2003 "Programming is Gardening, not Engineering – A Conversation with Andy Hunt and Dave Thomas" which was the first SW gardening article he could find, as he says.

By taking one more step back and putting the topic in a wider frame, we can also go further back in history, much further. The architect is like a craftsman creating and forming something, the gardener in contrast is less active but takes care that the plant or tree flourishes in an optimal way (from gardener's point of view). Both are well-known pedagogical metaphors (Treml 1991). Treml refers to Brunner, who mentions that both notions can already be found in ancient Egyptian writings about education. This means we aren't breaking new ground by letting both ideas compete against each other.

Anyway, after this brief historical excursion it is time to compare both philosophies in more detail. I want to do this with help of the aforementioned SW architecture mind map or let's call it, for this article and context, SW architecture tree. In the following, I will swing from one branch to the other and dive a little into the details. If you take a closer look, you will see that the SW architecture tree even comes with lianas (correlations) that help to move from one branch to the next. Welcome to the SW architecture jungle…



What?

(Starting with the “why” is usually the better approach, however, in this case I like to begin with "what" and the existing standard, which serves as ideal starting point.)

According to IEEE 1471 (IEEE) SW architecture is the systematic organization of components and their relationship to each other and the environment. Additionally, it provides guiding principles for design and evolution.

The gardener has to organize her/his yard, park or orchard in such a way that the plants can get along with each other, fit to the light and climate conditions, but in the end she/he has little control over the weather and their actual thriving. Too little I believe, compared to SW. Well, we often say the “source code has grown” what sounds like an organic property. We talk about it as if speaking about plants and flowers, but of course the source code

isn’t biological material. It is dead and the developers must arrange it properly to bring it to life. Nevertheless, the programmer has, at least theoretically, full control over what she/he is going to create. Each line of code written is one step in the transformation from abstract building blocks towards a concrete application.

What about the guiding principles the standard talks about? Maybe there is a specific plan for how to arrange the garden for each season of the year, but that is more about the organization of repetitive tasks than evolution. Furthermore, I am afraid the plant metaphor could serve as a general excuse that the outcome of SW is something out of our control. That in turn would result in more technical debts and less quality. If SW gets out of hand, we must find ways to regain the control instead of changing the paradigm that defines our profession, haven’t we?

But does much evolution happen when an architect designs a house? An answer to this question will be given at the end of the "why" section.

Anyway, there are two important things I like to mention here: First, SW architecture exists even if there is no dedicated person or role to take care about it. Every SW has some architecture, and no SW system exists without. If no one missed an explicit “caretaker” of architecture, the inherent structure is most likely a good one or at least good enough. This is usually a viable approach for small projects, I guess most people don’t engage a gardener for their own small garden or an architect when updating the kid's room. Second, a few diagrams don’t form a SW architecture, but only the graphical representation of some of its parts.

When going along the “what” branch of the tree (the mind map) and looking at ASPICE, the V-model comes into play. Whenever we plan for the left thigh of the V, we should already have testability and testing as an integral part of the right thigh in mind. Is there something equivalent for the SW gardener? Anyway, testability is a quality attribute and thus a perfect handover point to the non-functional requirements and the why-branch.

Why?

SW architecture is about the hard parts, that stuff that is difficult to change. So, to stick with the metaphor, you could say the gardener is doing a great job when trying to saw off the branch she/he is sitting on by providing an architecture so good that there are no hard-to-change parts left (and the SW gardener promptly vanishes in a puff of logic).

What I think comes up short in the gardener analogy, is the importance of non-functional requirements, also known as quality attributes, and the consequences of neglecting them. For instance, you don’t have to pay too much attention to functional safety in your garden. All right, don’t let the rake lying around. But this is for your own safety and not for your plants. But what happens if you use less reinforcement bars in your buildings than you need? The disaster is inevitable.

There is one more crucial shortcoming that applies for either architect or gardener, both are mainly concerned with the visible features of their product, the new building or garden. In contrast, the SW architect doesn’t focus on the visible features like UX or the functional parameters, but more on the non-functional quality attributes. The main task of the SW architect is to take care of these quality attributes, to pave the way for the developers, the

SW craftsmen, so that they find fertile ground for the best possible realization of functionality. Quality doesn't emerge out of nothing; you have to plan and care for it (Engel 2021). Consequently, the SW architect's work revolves more around the infrastructure and things behind the scenes and less around the visible outer appearance the customer gets directly in touch with. In this sense, SW development is craftsmanship, but not an art, since art usually doesn't offer functionality, *only* beauty. However, because no one visits SW repositories to see beautifully made SW, no one will request and pay for a costly and complexity adding SW architecture just for its pure aesthetics and elegance.

A clean SW architecture reduces the cognitive load and doesn't increase it by adding unnecessary fancy stuff. Sure, functional beauty and elegance in the sense of simple to use is welcome. So, I say the SW architect has to do the job of an urban planner (yet another analogy, but a good one), she/he is planning the streets, road network, power grid, water and drainpipes and everything the SW craftsmen need to build efficient applications. Furthermore, the urban plan will have areas with high-rise buildings (complex multi-threaded applications) and areas with bungalows (single-threaded applications), ports/airports/stations for in- and output. And sometimes you have to re-plan entire quarters of a city to stay future proof, so even evolution and the whole strategic aspect can be explained with the urban planner analogy. Like an urban planner, the SW architect is better off planning for years (platforms, product line) rather than seasons or single houses.

How?

The one basic principle in dealing with complexity is divide-and-conquer. But the resulting elements of the decomposition processes are manifold: Components, interfaces, packages, threads etc. like rooms, stairways, levels/floors, elevations, ground plans but also different standards, principles, concepts, patterns, techniques, methods, metrics, and tools. Of these, I want to single out the concept of a pattern language which was coined by the architect Christopher Alexander (Alexander 1977) and is perhaps one reason why the architect and architecture analogy is so widely used for SW.

That of course raises the question when the term SW architecture was initially introduced. According to the German Wikipedia this happened at the NATO Software Engineering Conference 1969. The report of that conference reveals that Ian P. Sharp said (NATO 1969): "*I think that we have something in addition to software engineering: something that we have talked about in small ways, but which should be brought out into the open and have attention focused on it. This is the subject of software architecture. Architecture is different from engineering.*" Looks like this was the official birth of the term SW architecture.

But back again to Alexander's pattern language. It is a powerful means that helps us in many situations, we nowadays use architectural patterns, design patterns, and refactoring patterns in a naturally way. Apart from the formal character, the pattern idea is much older and omnipresent. Think about the farmer's old weather lores or recipes in general. Patterns are like recipes; they list the necessary ingredients and preparations methods that have proven useful in specific situations. And there are recipes for cooks, bakers, and pharmacists – basic recipes for sauces and dough, but also remedy for headache. Consequently, we could also speak of SW chefs instead of SW architects, especially

because we like to have a SW *cookbook* at hand in some situations. It seems a quite young profession such as SW development, likes to borrow the jargon from older ones.

Interestingly, there is a high affinity of SW engineers to patterns and cookbook sections in manuals, but often some kind of aversion against process descriptions. What is the difference? I think it's the missing freedom of choice, there is only one process (with different paths) to follow. It feels like narrowing down the creative spirit of the SW artist. However, I believe this is a misperception. Process descriptions are exactly like recipes for home-style cooking, they are written down to make a procedure reproducible also for new joiners. Maybe they are sometimes formulated too strict, processes shouldn't be tight corsets preventing any degree of freedom, they should keep the necessary flexibility to work even when some parameters deviate from the usual ones. Furthermore, processes are made for the people, they should reflect proven ways to work, the important point is that not the people are made for the process. Having this in mind, it becomes clear that the processes need to be managed in an agile way and updated from time to time when the proven way to work has changed. In other words, process descriptions aren't set in stone, such ancient artifacts belong in a museum. Make the work on processes and concepts transparent, show cooking can increase appetite and increase confidence. To finish this process discussion, remember that strategies can be much better elaborated and optimized once they are recorded in some way. Maybe not always as boring text, but sometimes as video. I think some of the people who complain about unloved processes at work, will watch YouTube videos in their free time that describe how to replace the broken display of their mobile phone. That's nothing else than a process description, isn't it?

Of course, it makes sense to use modern tools not only to model and document the architecture, e. g. Wiki based documentation can help to keep descriptions vivid and living documents. But tools should also be used to continuously gain values of the defined metrics and such a way keep track of the correct architecture implementation and to identify potential gaps and issues early. The values, the interpretation and effects should be easily accessible to all development teams to raise the cause-and-effect awareness and to showcase why specific architectural decisions are reasonable and how they work. Don't underestimate the power of "why" (Cialdini 2007) and in this case the rationale should be more than just "because". This is especially true since SW metrics aren't as visible and intuitively understandable like height, width, and angles you find in the garden or house. In best case, the metrics and tools can be used and interpreted as simple as a measuring tape or plummet.

Wiki based documentation is much more efficient than writing emails or creating some nice presentations. It can be used for daily work and easily updated by everyone, questions can be asynchronously raised and answered and are publicly visible. Answering questions received via e-mail, messenger, phone call etc. by simply pointing to the relevant Wiki page is just a way of consequently following the DRY principle. However, keep in mind that all text, diagrams and even the most sophisticated models supporting different views, are only representations of the architecture, not the architecture itself. All these things are just means to share, evolve, and refine the actual architectural idea. What automatically leads to the next point, communication.

An important factor in working as a SW architect, at least in my experience, is the hair salon effect. Not the cosmetic aspect, of course, but often the hair salon is a place to catch the latest news and even to fill the unknown unknowns. So, regarding communication and spreading information between the teams and stakeholders, an architect should act like a stereotypical hairdresser. Although this part isn't visible in the SW architecture documentation, managing the human interfaces is as important for the manifestation of the overall architecture as managing the SW interfaces. What does it help if specific problems are already addressed and solved somewhere if this information isn't distributed? Successful information sharing may also need some promotional work, to make other people's ideas interesting.

Wrapping up this "how" section, I conclude that the work of a SW architect is closer to the work of an architect than that of a gardener because of the human factor. I suspect a gardener has fewer human interfaces to manage. However, generating, maintaining, and coordinating the information flow between stakeholders and development teams is a crucial point in the work of a SW architect.

Where?

The architect needs stable ground and foundation. The gardener needs fertile soil and healthy roots. And the same goes for the SW architect. Early decisions about the OS, programming languages, frameworks, architectural approach, and patterns constitute the base for later decisions. This is the part that is *really* hard to change later on.

These decisions shouldn't only follow the latest fancy trends, but also take into account things like project duration and availability of resources – both kind of resources, system as well as human resources. Which languages and frameworks are the developers familiar with, is the project term long enough to train the people in some new technology? Will it scale or is it portable to other systems? For the architect and the gardener, available and present resources are usually more affordable than exotic ones.

In some cases, there is no time to gain the necessary know-how and experience, or it is strategically irrelevant. That is when prefabricated houses come into play. Even the gardener won't grow all plants from seeds. Sow or lay turf corresponds to make-or-buy in SW. However, the integration of COTS, FOSS or outsourced packages requires diligently groomed and traceable requirements, well-designed unambiguous interfaces, aligned processes and schedules. Similar to construction sites, where you have to ensure that all trades can smoothly work hand in hand. For instance, no electricians are needed until the walls are finished. But maybe the electricians can start their work when one of the lower levels of the house is already done while higher levels are still being built. I think those things are comparable to the relationship between DevOps and SW development. Who knows, maybe in the future, driven by recent supply chain issues, the construction sector could learn from the DevOps idea and the various trades and different building sites can be optimized to better share rare resources.

As mentioned under "why", SW architecture is, unlike construction architecture and gardening, mostly about the invisible meta things. The non-functional requirements can usually not be realized within isolated components but are addressed by cross-cutting concepts, e.g. security is everybody's business.

Who?

SW architect is no protected job title and there is no dedicated, specific course at university you can take to become finally a BSc or MSc of SW Architecture. Well, you can get certified (iSAQB). However, the building architect comes with its own course and graduation. Additionally, there are the related civil engineer degrees. What does that mean? Do we need a special SW architect course at university? No, because it's beneficial if the SW architect acts as primus inter pares. We aim for cross-functional teams or teams of teams and of course we don't want astronaut architects (Spolsky 2001). That is, no SW architects sitting around a round table like Arthur's knights designing an ivory tower architecture. Thinking in silos and specialization happens early in enough in professional life. In my opinion, education should build up a broad base of knowledge for as long as possible. The SW architect must understand the needs of all the different parties that participate on a SW project. Therefore, similar education is helpful even if many SW developers are trained electrical engineers or physicists, what all have in common is that they have learned to learn.

The personal experience will differ depending on what kind of systems a SW architect is designing. For example, there will be a difference between large scale cloud computing and embedded SW architectures. A SW architect will be more specialized for that area she or he is working on. Same goes for the construction industry, here you can find architectural firms specialized for private houses, office buildings, public buildings like operas and theaters, and (some Berlins may have prayed for it) airports. In addition to gardeners, we see fruit and vegetable growers. However, this is about the fundamental things that require adapted thinking. But I dare to say, within those categories and in relation to a specific domain, a SW architect should be more of a generalist than a specialist. Unfortunately, you can't be both, you are either a specialist or a generalist. By the way, to point out the opposing position of both, I prefer the term specialist over expert. Anyway, no need to worry, there are usually enough specialists in the empowered teams to which you can reach out and get consultancy. So don't be too disappointed if your team doesn't consist only of experts and specialists. Hopefully, your larger projects will have one or more generalists working as SW architects and taking care of the quality attributes, in bad case you only get an anonymized FTE (Engel 2018).

My point of the "who" section goes to the architect and not the gardener, as I think the work of the gardener is almost entirely focused on the working material, the plants. The architect, like the SW architect, has to spend a significant amount of their time dealing with other stakeholders as well.

When?

Everything starts with creation. Or maybe a little earlier if SW can be derived from an existing product line. Such a SW is formed by a platform that is designed for reuse, i.e. just copying some older code is no product line SW engineering. To make such an approach work successfully, the variation points have to be carefully planned. Similar to standard garages which can be adjusted in height, width and length, or the previously mentioned prefabricated houses. When it comes to gardening, I'm aware of reusable parameterizable elements, but I haven't heard about configurable complete solutions. Gardens and parks are, as far as I know, individually designed places.

However, a garden begins to live and is partly usable right after its formation or even during the formation (well, the elements/plants actually lived before, but the garden as a whole can’t live prior its existence). Buildings, on the other hand, aren’t used until they are completed. In this regard, SW that is developed according to an iterative, evolutionary or agile process model behaves like a garden. First features can already be tested and used in an early state.

And also later in the “when” timeline there are more commonalities between gardening and SW architecture than between architecture and SW architecture. As we left the waterfall process behind us, the SW architect accompanies a project not only in the early design phase, but like a gardener (and the urban planner) throughout the complete life cycle. This kind of support is necessary because SW and gardens are not fully predictable already in the planning phase. The principal idea exists but concept details often can’t be worked out in advance but only when further knowledge has been gained. Compared to buildings, there are fewer parameterizable, but more custom-built elements and subsystems. That downside goes along with the upside that some features can be changed or added also in later states.

However, plants are unbeatable when it comes to flexibility, they come with some inherent modularity. If you are late or missed one cut, you can make a more drastic cut next year. You are free to replace single plants or, as gardens are designed in areas and flowerbeds, to update/upgrade larger parts independently; there aren’t many interfaces you have to care for. That’s different in architecture and SW architecture. Failures, if undetected for a longer time, are way more difficult to fix. Buildings consist of rigid elements and are constructed to remain unchanged for decades and are usually not specially prepared for change. Modifications are therefore costly and difficult, especially if the building structure is impacted. SW can behave in either way depending on how it is realized. Organized in small and loosely coupled parts, it could be treated in a similar way like a garden, otherwise, if designed as a monolith, it is more like a building made of concrete.

Before the project lifetime finally ends, it runs for a long, usually very long time in a maintenance phase. SW developers and SW architects have to reckon with the fact that they are still responsible when a project enters this phase. That means you better treat your quality parameters properly. As I’m quite sure you already helped friends move, I don’t want to give another example from gardening or construction industry here, but from appliance industry. There’s usually one thing that remains untouched until the end because nobody wants to haul it: The washing machine. Whenever the lot fell on me, I’ve always wondered and damned the engineers why on earth they didn’t spend a few cents and equipped the machine with recessed grips instead of just sharp edges. Apparently, portability doesn’t matter at all for washing machines.

No clear winner for “when”, so I want to close this section with two proverbs that show similarities in gardening and architecture in one aspect: Grass won’t grow quicker if you pull on it and Rome wasn’t built in one day.

Conclusion

As expected, there are pros and cons for both analogies. However, while writing this article, I realized that the “original” architecture, at least in my estimation, fits better than

the gardener analogy. The key point is in the “what” section: The gardener metaphor could be interpreted as an excuse that SW outcome is out of our control and less quality and more technical debt are not our faults. That’s too easy. The claim must be not only to write code, but professional SW development.

To achieve this, architectures as an integral part of SW, have to be continuously evaluated. In order to make this possible in an objective and comparable way, the evaluation is – in best case – based on measurements that can provide quantitative and not only qualitative data. Good data is always the starting point for real engineering with continuously improved methods. SW engineering also means sharing of experience and discussing them, eventually leading to agreed best practices and standards. These should finally be reflected in a generally accepted SW architecture role description. Yes, I believe such an approach is also possible in a rapidly changing field like SW. Continuous improvement is fine and welcome, of course, but starting over and over with essential fundamentals seems like a waste of time.

Despite of SAFe and alike frameworks, technical (SW architecture and engineering) and management (PO) perspectives are still not fully aligned. At least not in symmetric way, it seems that the technical perspective gets aligned more with the management point of view than the other way around. Work packages that deal with non-functional requirements are still too often scarified for functional flash in the pan features. Unimaginable in construction industry, nobody would decide to forgo the structural verification or leave out the reinforcement bars to save time. Another example for more appearance than substance are ongoing discussions about agility vs. velocity vs. speed. After many years of agility, it still needs articles like Henney’s *“Agility ≠ Speed”* (Henney 2021) to clarify the meaning of basic elements which are accidentally or intentionally misinterpreted. By the way, his article contains some very nice analogies and metaphors.

After 50 years of SW architecture and 20 years agility, some basics are still not established. Although there are already ways to escape this time loop, such as WSJF prioritization (SAFe). Parity can be reached if the user-business value is assigned by the PO but the risk reduction and/or opportunity enablement value is assigned by an architect. SW architecture has to become louder.

A few words about efficiency as one might wonder why building houses seems to be much faster than building SW. What takes long is the creative part of designing and writing quality code but the actual production is only a matter of seconds or maybe hours. Keep in mind, the process of creating SW is like an urban planner cooperating with many architects, or an architect collaborating with an army of interior designers. Nevertheless, the construction field has seen some disruptive inventions like concrete or CAD which surely had a great positive impact on the business, maybe some future inventions will have similar effects to SW engineering. However, when talking about custom-built individual solutions the positive effect might be less, think about medieval cathedrals which took decades (of decades) to build, even Barcelona’s modern time Sagrada Família is no short running project.

Closing words: Of course, it doesn't matter whether you call yourself SW architect or SW gardener. A job title is a just decorative accessory, but the tasks require reliable commitment. In other words, the **why** and **how** you do **what**, **where** and **when** matters.

References

1. Aitchison, Chris 2018. *Are you a Software Gardener?* https://medium.com/@cmaitchison/are-you-a-software-gardener-f79eba5b7fb7
2. Alexander, Christopher et al. 1977. *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, ISBN 0-19-501919-9.
3. Cialdini, Robert B. 2007. *Influence: The Psychology of Persuasion.* New York: Collins, p. 4.
4. Engel, Dirk 2018. *Why FTE based planning is not useful for managing SW engineering teams.* https://github.com/dirkengel/articles/blob/main/FTEbasedPlanning.pdf
5. Engel, Dirk 2019. *Aspects and correlations of (embedded) software architecture in a mind map.* https://github.com/dirkengel/sw_arch_mindmap/blob/main/SoftwareArchitectureMindMap.pdf
6. Engel, Dirk 2021. *Quality and the Project Management Triangle.* https://github.com/dirkengel/articles/blob/main/QualityAndTheProjectManagementTriangle.pdf
7. Henney, Kevlin 2021. *Agility ≠ Speed – Software development benefits from a sense of direction*. https://kevlinhenney.medium.com/agility-speed-96057078fe40
8. IEEE 1471. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.* http://www.iso-architecture.org/ieee-1471/defining-architecture.html
9. iSAQB. *Internationally Recognized Qualifications.* https://www.isaqb.org/certifications/
10. NATO Science Committee. *Software Engineering Techniques.* In: *Report on a conference sponsored by the NATO Science Committee.* Rome, Italy, 27th to 31 October 1969. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF, p. 9.
11. Papapetrou, Patroklos 2014. *The Art of Software Gardening*. https://leanpub.com/art_software_gardening (maybe the only reference to a non-existing book).
12. Papapetrou, Patroklos 2015, *Software Gardening, 2015: Yet Another Crappy Analogy or a Reality?* https://www.methodsandtools.com/archive/softwaregardening.php
13. SAFe. *Weighted Shortest Job First*. https://www.scaledagileframework.com/wsjf/
14. Spolsky, Joel 2001. *Don't Let Architecture Astronauts Scare You*. https://www.joelonsoftware.com/2001/04/21/dont-let-architecture-astronauts-scare-you/
15. Treml, Alfred K. 1991. *Über die beiden Grundverständnisse von Erziehung.* In: *Pädagogisches Wissen, pp. 347-360.* https://www.pedocs.de/volltexte/2021/21882/pdf/Treml_1991_Ueber_die_beiden_Grundverstaendnisse_von_Erziehung.pdf
16. Venners, Bill 2003. *Programming is Gardening, not Engineering - A Conversation with Andy Hunt and Dave Thomas*, Part VII. https://www.artima.com/articles/ programming-is-gardening-not-engineering
17. Wolff, Eberhard,2021: *Dokumentation – ein Überblick.* In: *Software Architektur im Stream*, part 87. https://youtu.be/yjhOLb1WzpM?t=795