Professional Front-End Architecture

Helping Front-End Development Reach Its Full Potential

Fabio Nolasco

This book may be purchased for educational, business, or promotional use. Online edition is also available. For more information, visit fabionolasco.com

Cover & Interior Designer: Natalia Nolasco

August 2018: First Edition

While the author has used good faith efforts to ensure that the information and instructions contained in this work are accurate, the author disclaims all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

Dedication

To my wonderful wife Natie, for her immense help with this book and also for making my life happy beyond comparison!

I would also like to thank Anmol Saraf and Akhilesh Nair for their encouragement in writing this book, without which I may not have even started.

I am indeed a very blessed man!

Fabio Nolasco

Forum

To further your studies in Front-end Architecture, please visit the website below. There you will find an open forum where people can ask questions and discuss the concepts and practices of the field.

frontend-architecture.com

Table of Contents

Introduction

This is the moment where I am expected to make a sweeping, powerful statement that will set out my aims for this book and capture your attention. However, that is a very challenging task, because there is no industry-standard definition for front-end architecture, and therefore no quick way to state the problems I will address and the solutions I will propose.

I recently read the book *Frontend Architecture for Design Systems* by Micah Godbolt. It was an excellent book with great content, but like most material on front-end architecture, it focused on the implementation of front-end technologies and other practical perspectives.

Despite having a different take on what front-end architecture is, one sentence in Micah's book got to me. I'll paraphrase his idea: "Nobody would build a skyscraper without proper planning. However, that seems to be the case with front-end products."

Indeed, unfortunately, that is how things are still done in most companies. However, front-end development nowadays is undoubtedly too significant, too expensive, too large, and too complex to be contemplated merely as a subpart of the web development pipeline. We are no longer creating simple web pages, but real web-based software. The lack of methodological principles is leading to a significant loss of revenue and opportunities.

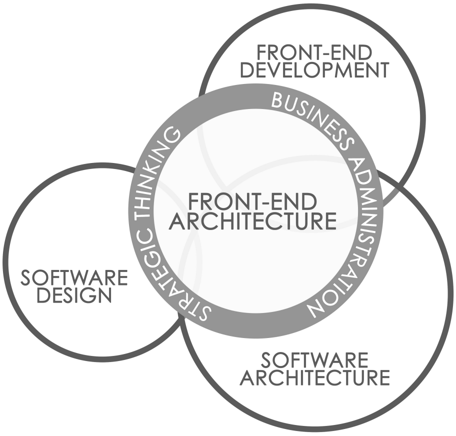

Figure 0.1 – Overview of the scope of knowledge proposed for front-end architecture.

The proposal that I present here is that front-end architecture should be a dedicated specialization of software architecture, merged with some practices and principles of software design. It should also embrace, more purposefully, concepts from strategic thinking and business administration, creating a robust approach to support the architectural work done to guide front-end projects.

Software development is commonly thought of in three levels: software design, software engineering, and software architecture. Currently, the majority of front-end applications created are essentially web-based software. Therefore, understanding the three parts of the conventional software development process can help us achieve higher maturity levels in front-end development work.

Software design is the process between requirements gathering and programming, through which we conceptualize, strategize, organize and plan the distinct parts of the system and their connections. It starts with the analysis of the requirements and results in the creation of high-level blueprints to guide coding.

Software engineering has a broad range of concerns related to software production. It includes design, development, testing, and evaluation of software, always aiming to promote system quality and solutions for complex challenges.

Software architecture is a higher level of abstraction, providing strategic direction for the organization. It takes into consideration factors such as other systems created and used by the company, infrastructure, deployment strategies, marketing strategies, business goals, the future of technologies, market tendencies, etc.

These definitions are simplified, but allow us to continue with our line of thought. The point here is to show how front-end architecture has been commonly thought of from the software design level, and sometimes not even from that. It may be the result of years of belief that front-end development is easy and simple. Nevertheless, even if that was the case in the past, it certainly is not simple anymore.

Companies spend thousands and sometimes even millions of dollars on front-end projects, yet frequently neglect architectural work. In consequence, even though their tech-stack may be technologically sound, their front-end projects are fated to have constant issues, not contributing to the company's success as much as they could.

We find, then, that solving problems from the technical perspective is not enough. Allowing front-end code to grow "organically," based on frequent changes spurred on by Agile cycles, result in lower quality products that may require a lot of reworking. What is referred to as "emergent design," in the end becomes "no-design," which has incredibly limited strategic value. Despite this approach working in the past, times have changed.

Software architecture has increasingly incorporated topics about the web. Nowadays many, if not most, of the topics covered in related conferences are directly focused on it. However, even the topic “web” is immensely vast.

Front-end development is a universe in itself. It is extremely challenging to keep up with its incredible change rate. How can any architect provide sensible recommendations without accurate knowledge of the past, present and future of browsers, operating systems, devices, testing mechanisms, frameworks, languages, language versions, language supersets, tools, UX and UI tendencies, and so much more?

From my perspective, software architecture has become sort of a Pandora’s Box. Open it up, and you’ll find a far-reaching breadth of subjects, including infrastructure, cloud management, security, database, back-end development, front-end development, software development, mobile development, devops, artificial intelligence, and so on. These are obviously genuinely important subjects but are too much for one person to be an expert on. Keeping software architecture as a catchall box causes architects to provide unrealistic and difficult-to-follow recommendations, resulting in many people simply rejecting the idea of architects altogether.

Also, traditionally speaking, software architecture doesn’t usually take into account front-end concerns nearly enough, neglecting to provide conceptual principles on how to plan and manage successful front-end projects and front-end shops.

We have already identified two problems: 1) that front-end development has become very important and complex, yet is still done without proper architectural foresight; and 2) that even if a developer wanted to do that, they would lack the methodological principles to do so.

This book proposes to provide precisely that: a methodology for front-end architecture based on lessons learned from software design and software architecture adapted to the front-end world. By stopping to address our front-end issues from a merely technical perspective, we can finally break from the endless cycle of putting out fires and start to truly engage in continuous innovation.

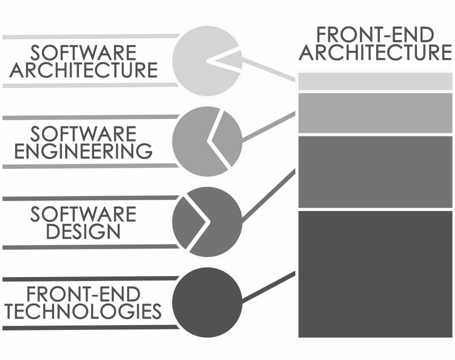

Figure 0.2 – Importance of certain computer science disciplines for the new front-end architecture methodology.

I recently heard rumors of certain companies dissolving architects' roles. They justified the layoffs this way: "Why waste time on something that will not happen? How can a person who hasn't touched code in twenty years give any appropriate direction? Architectural work is a hindrance to our projects' progress."

Architects have been accused of a) offering unrealistic recommendations; b) providing low-quality plans; c) creating unnecessary noise in the communication between developers and stakeholders; and d) causing difficulty for projects to adapt to new business strategies. Unfortunately, I need to agree with most of it. A common defense they provide, though, is that those issues come from "problems in communication." But wait, wasn't that one of their primary responsibilities, to be the promoter of proper communication?

So, if architectural plans don't work, why would I write a book about it? Easy, because architectural work can be extraordinarily profitable when done correctly. It is my experience and belief that through an adequate understanding of what front-end architecture is, it can generate tremendous benefits, giving companies a significant edge over their competitors, both in cost savings and in time to market. The four pillars for this methodology are:

a) To consider front-end architecture from a higher scope, beyond merely choosing tech-stacks and file organization;

b) To absorb concepts from software design and software architecture and adapt them to front-end development;

c) To focus on front-end related topics, leading the architects to have more profound and relevant knowledge; and

d) To proactively implement principles of business administration and strategic thinking;

As you can see, these changes result in a new type of profession, distinctive from the conventional software architect. It also yields different results.

Insight about how changes in the scope and role of an architect can lead to better plans came from history lessons regarding “strategic planning.” In the 1990’s, despite the vast fame it had, its usage was heavily discouraged once business owners and managers finally noticed the inefficiency of the plans created. Experts in the subject researched and saw that the problem wasn’t with the concept of planning itself, but with the lack of strategic thinking, weak strategic management to follow through on plans, and the inadequate scope of the planners’ roles.

Planners were following processes that were way too formal. Could excellent strategy be forcibly produced? They were not involving the necessary stakeholders, or applying enough strategic thinking, and were coming up with strategies themselves instead of capturing emerged strategies from managers and hands-on workers.

In the same way, software architecture can suffer if those involved in the planning are doing it incorrectly. Eliminating architects does not solve the problem, since at the end of the day, whether formally or informally, architectural decisions WILL be made and plans WILL be created. But how correct will those choices be? How will they impact the company in the short and long-term? Will they promote or hinder the fulfillment of the company's goals, mission, and vision?

> *“Strategy without tactics is the slowest route to victory, tactics without strategy is the noise before defeat.”*
>
> – Sun Tsu, Ancient Chinese Military Strategist

In other words, conscious architectural work is necessary, but it needs to be done right. This fact is clear to us, but managers who do not understand the numerous and unique challenges of front-end development will often fail to consider the idea of investing in front-end architecture. Fortunately, after years of losses, many of them are starting to wake up to that reality.

They hid behind the simplicity of HTML, CSS, and JavaScript as a way to ignore the need for proper planning. These languages are a fraction of the complexity of the languages they used before. However, it is precisely their simplicity that makes it harder for us to build skyscrapers with them, especially when we consider the variety of deployment destinations and run-time environments.

The front-end code is the part of the system that reaches the customer, and it is also the most frequently changed. It needs to accommodate new trends, sensors, UX recommendations, marketing directives, and functionalities.

Even in cases where architects are not formally present in the organization, and the architectural responsibilities are distributed between managers and engineers, they still need to be well prepared to make those decisions. It has been said that: "The difference between an architectural and an engineering decision is that the first is usually way more difficult and costly to change than the second."

Architectural choices are important not only because of their cost but because they can either open or close the door for improvements and innovation going forward. Changing algorithms can be relatively easy and cheap to perform, but it can be costly and time-consuming to fix a poor architectural choice after the fact.

In other words, anyone with authority over front-end architectural decisions should educate themselves. It is difficult to say who needs a more robust front-end architecture methodology, if bigger companies, since their systems are numerous and complex, or smaller companies, since they have a much lower margin for errors and loss of revenue. Planning, if done correctly, is never a waste of time, and can often lead to enormous payoffs.

> *"Strategic planning is not strategic thinking. Indeed, strategic planning often spoils strategic thinking, causing managers to confuse real vision with the manipulation of numbers."*
>
> – Henry Mintzberg

It's time to rethink how we plan. Great architectural work involves more than learning new tools or following recipes. It requires theoretical concepts to be proactively contrasted and smartly applied to the different scenarios that each of us faces every day. Only then will we be able to create realistic and effective plans.

To achieve this, programmers will need to study more theoretical principles, and managers will need to obtain more technical knowledge. We all know that change can be uncomfortable. However, this is the nature of architectural work: As architects, we must keep one foot in the conceptual realm and the other in the practical. Because of this, you will not find recipes here.

There are many great architects and thinkers out there, which at first made me wonder if I even needed to write this book. However, I felt the market was lacking something specific: a curated list of foundational principles selected explicitly for front-end architecture, aiming to help architects or those vested with architectural responsibilities to do excellent professional work. This is what I offer here.

To achieve this, we will not be able to dive too deeply into any specific technology or topic. There are some fantastic authors out there, such as John Papa, Dan Wahlin, Mattias Johansson (from Fun Fun Function), Deborah Kurata, Uncle Bob, Steve Krug, Kent C. Dodds, John Lindquist, Joe Eames, Mark Zamoyta, Mark Richards, Neal Ford, Jafar Husain, Sam Ramji, and many others. If you are looking to dive more deeply into any of the topics I cover, I cannot recommend their work enough.

This book may be more technical than most managers are used to, and programmers may find it too theoretical. However, any architectural decision made merely from one side is potentially fatal. Front-end architecture needs to be considered a multidisciplinary study, providing an efficient and effective bridge between all those with a vested interest in the project, filling the gaps, and promoting synergy through the compilation of ideas and strategies.

In addition to the list of authors above, I advise you to take note of the names of people, companies, and technologies I cite throughout the book, looking them up as you go along, if necessary. Since no one is sponsoring this book, I am at liberty to mention anything I believe might benefit you. However, this does not necessarily mean that I am endorsing anyone or vice versa. I hope these real-world examples will help you apply knowledge more effectively as you read and your curiosity is peaked.

Likewise, we will cover many practical topics, but always from an architectural perspective. For this reason, if you see a term that you are not familiar with, like Dart, Elm, Rust, Web Assembly, Virtual Dom, Hyper-HTML, Accelerated Mobile Pages, Quantum CSS, Applitools, and others, just read a quick summary about it online and you will be good to return to the book. If you are interested in these new subjects, there are many materials out there to help you learn and grow in your knowledge.

At the same time, I realize you may be reading this book to gain a concise understanding of front-end architecture, avoiding spending too much time seeking good content online, so I will make sure to pass on all my best tips.

This book has six main target audiences:

a) Developers aiming to become front-end architects.

b) Front-end architects seeking a continued education.

c) Recruiters seeking a better understanding of the field.

d) Managers and engineers seeking formal preparation on how to perform architectural work.

e) Managers needing guidance on how to hire and monitor the work of architects.

f) People with influence over architectural decisions seeking better ways to contribute.

In summary, I will present a proposal for the role and the field of front-end architecture based on a higher level of abstraction than is currently understood. We will review its importance, tasks, traps, pitfalls, best practices, the problems it solves, the business value it adds, and its trade secrets, as well as how to run successful front-end projects and front-end shops.

It will not teach any specific technology, nor explain how to configure or implement any particular framework or tool. What it will provide you with are high-level scenarios, thought-provoking ideas and hopefully, enough questions to get this discussion started across the front-end community. My ultimate goal is to help you develop an excellent architectural mindset, leading to realistic and professional front-end architecture work.

1. What is Front-End Architecture?

Most professional fields have specialized branches growing off of their original trunk. Medicine has cardiology, dentistry has orthodontology, and so on. That tends to happen when the amount of information in a subject becomes too much for one person to know it all. At the same time, holistic views are obviously still necessary. But how can we find an adequate balance between depth and breadth of knowledge?

On one side we have the old view of front-end architecture, where it is only about file organization, tech-stack selection, dev environment setup, and maybe the definition of data-flow strategy. On the other side is the concept of software architecture, with dozens of exceedingly broad subjects, where each of them could turn into multiple distinct professions. It seems evident to me why neither approach works: the first fails to consider the big picture (not enough breadth), and the second tends to result in shallow and unrealistic plans (not enough depth).

> *"… everything has a past. Everything – a person, an object, a word, everything. If you don't know the past, you can't understand the present and plan properly for the future."*
>
> - Chaim Potok

My vision for front-end architecture isn't that it should be a middle ground between both sides, but something new, with parts carefully selected from both system design and software architecture, supported by principles of strategic thinking and business administration, and focused and adapted to the front-end world. Front-end development already is decoupled enough from other parts of the system, and complex enough to justify specialized professionals. To provide it with specialized attention for planning and management seems the logical next step, especially when we consider the investments already done in it and its strategic part on every company's survival.

In order to understand how such view of front-end architecture would manifest in practice, we need first to refresh our memory a little bit regarding some concepts and historical processes. We can't perform an extended literary review, but a quick recap will allow us to define front-end architecture better.

Making a very long story short, and without going too far back, the 70s presented a peak in the usage of the expression "software design." In generic terms, software design is the formal study of how to analyze requirements, define structures, and select the best computational solutions to guide the creation of high-quality software.

Academically speaking, software design can be divided into two parts: architectural design and detailed design. I believe the most commonly used concept of front-end architecture nowadays is based on the sub-area: architectural design.

Software design is the first phase of the software development life cycle (SDLC), and it involves topics like modularization, coupling, and concurrence. The final product typically includes a matured version of the requirements' list, diagrams, and pseudo codes.

Even though this is an established practice in software development, we don't always see it applied to front-end projects. The illusion that front-end code is easier to change than software code can cause the impression that software design is unnecessary.

A frequent and unfortunate process widely adopted by many organizations is: managers and product owners agree on a project, the idea is then presented to the UX designers, the design is implemented by the UI and JavaScript programmers, and as the project moves along, a torrent of changes bubble down into the Agile cycles' backlogs in order to keep up with new ideas and business requirements that emerge. It is not surprising how frequently programmers request the opportunity to just recreate an application from scratch once they experience the difficulty in maintaining a current project.

A fallacy that emerges at this point is that front-end projects are better off if redone, given the fact that front-end technologies change so rapidly anyway. This is a poorly founded excuse since it ignores the reality that companies do not always have the time and money to recreate projects, the complexities of recapturing all requirements, the amount of work it will take to rewrite and retest code, etc. It also ignores the fact that cheap and fast updates can help the company free up resources to work on more relevant and timely projects.

When a system is well designed, “modernizing” it is more straightforward and painless. It saves money, speeds up the time to market and releases workforce for new projects. So even though recreating systems from scratch is very appealing to programmers and to new managers who want to attach their name to newly launched applications, it is not always the ideal solution for the companies.

Software design can bring many benefits to front-end projects. Therefore, let's put that into our bucket of absorbed principles. We will cover software design in more detail in further chapters.

As computers evolved, becoming more powerful and more complex, software design became insufficient to guarantee applications’ success. It was and still is essential, but it’s certainly not enough. Thus the 80s brought about a spike in the usage of the term "software engineering,” although the term itself started to gain popularity much earlier, around the late 60’s.

At the time, it seemed that everyone was trying to find the perfect methodology for software development, a silver bullet that would solve all problems. The so-called "software crisis" was no joke; a quick online search reveals crazy numbers, such as indications that maintaining software was often up to twice as expensive as creating it.

By merging a couple of definitions from IEEE (the Institute of Electrical and Electronic Engineers), we can define software engineering as a systematic, disciplined, and quantifiable approach for the application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software. It has many sub-disciplines, such as software quality, software design, and engineering management.

By the early 90s, it was becoming clear that the success of a project involved more than just following software engineering guidance. There were too many disruptions and unpredicted changes in the market, causing the systems to become difficult to maintain, even when all the technical best practices were observed. That is when the concept of "software architecture" came into focus, proposing an even higher level of abstraction for the orchestration of the software development.

Defining software architecture is a difficult task; even scholars have not reached a consensus. It is just too tough to find the right sequence of words to define this field succinctly.

In their video course "Software Architecture Fundamentals (2017)," Neal Ford and Mark Richards speak to that very challenge. They explain that the best approach they found was to use mind maps that include many aspects of software architecture work. I share that belief. Nonetheless, I believe it is important for us to examine some definitions.

During my research, I examined many academic papers, ranging from a fundamental lexical analysis of software architecture to full literary reviews on the subject. Luckily for you, however, this is not that kind of book. I will reference only a few of these definitions, and you can delve deeper into the discussion on your own if you wish.

As you evaluate the following quotes, think about their parallels with front-end architecture. Also notice how a restrictive definition can lead us to close the door on other essential aspects of the job, while a definition that is too open can cause us to lose focus in our daily work. With that in mind, let's see what some well-known people have said.

> [1] "architecture is concerned with the selection of architectural elements, their interactions, and the constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements and serve as a basis for the design." *[Perry & Wolf 92]*

> [2] "A set of artifacts (that is: principles, guidelines, policies, models, standards, and processes) and the relationships between these artifacts, that guide the selection, creation, and implementation of solutions aligned with business goals. Software architecture is the structure of structures of an information system consisting of entities and their externally visible properties, and the relationships among them." *[Dr. Jean-Claude Franchitti]*

> [3] "Software architecture is a level of design that goes beyond the algorithms and data structures of the computation; designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives." *[Garlan & Shaw 93]*

Most researchers of software architecture leveraged the lessons learned from areas that came before it, like hardware architecture, network architecture and even building architecture while elaborating on their thesis. However, it's my personal conclusion, as well as that of many others, that building architecture provides the closest (while not perfect) parallel.

1990s managers and engineers alike felt the need to incorporate more context into their planning, keeping in mind concerns from business administration, human resources, infrastructure, IT governance, and others. The point wasn't to discover new design patterns, write better code, or to invent a new framework; they needed to find a way to make smarter decisions and to create blueprints that were strategic both in technical and business perspectives.

I recently examined the list of topics covered in software architecture books and conferences and became discouraged. It is a tremendous amount of topics, some of them focusing on enterprise architecture, infrastructure architecture, system architecture, solutions architecture, domain architecture, and so forth. I realize that terminology discussions are complicated, but it illustrates that they don’t have practical relevance for front-end architecture.

Time is a limited resource, and evaluating those less essential topics takes away from our ability to study more profitable subjects that could be used to enhance the quality of our architectural recommendations. After all, how can I keep up with front-end technologies while being "specialized" in everything else? How valuable would my contributions be if I were to lose ground contact with the front-end world?

The following graphic shows a list of the possible breath of knowledge expected from a software architect. The values are for illustration purposes only, and will certainly vary for different companies, projects, seasons, and job descriptions.

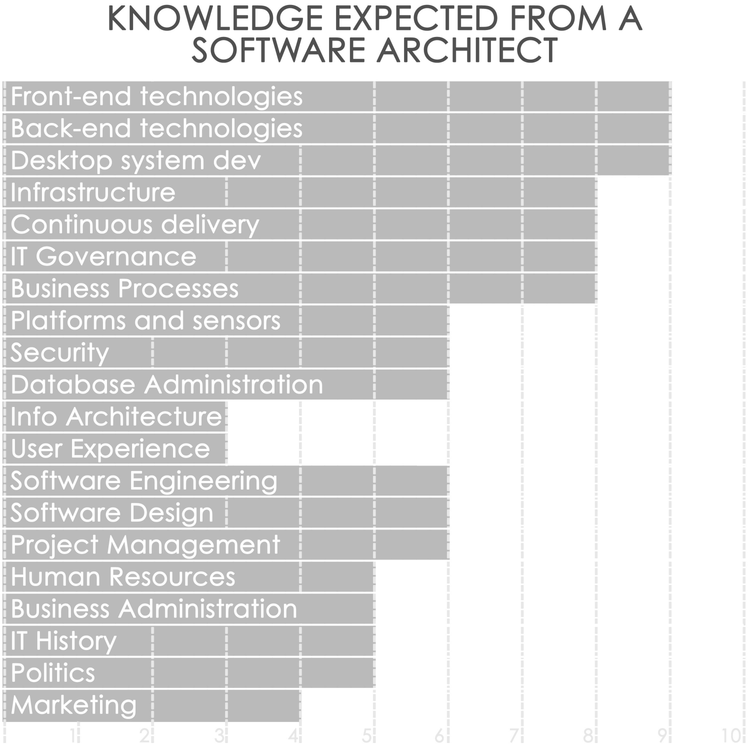

Figure 1.1 – Chart showing the various levels of knowledge expected from a Software Architect.

Now let us complete the same exercise while contextualizing it to front-end architecture. Again, these values are hypothetical, and merely represent a simple logical inference of a possible typical case. The darker bars are the ones that have changed in comparison to the software architecture graphic.

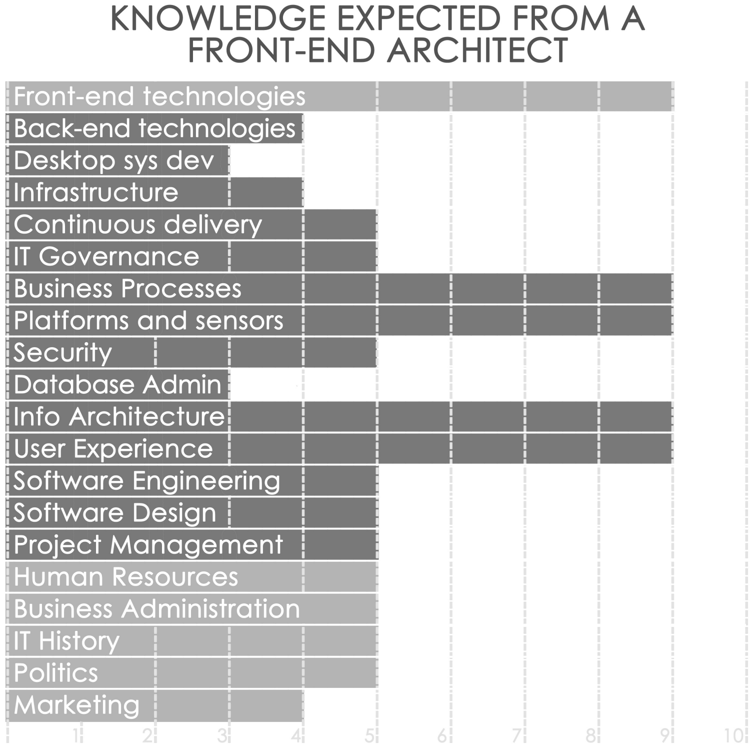

Figure 1.2 – Chart showing the variance in levels of knowledge expected from a front-end architect, with the darker bars being the ones that changed from the previous chart.

As you can see, all the bars remained. What changed were not the subjects, but the focus. Someone working with front-end development would naturally need to understand more about platforms, sensors, and user experience than someone working on back-end development, for instance. Other subjects such as infrastructure and continuous delivery, while still present, become less relevant.

You'll notice that I did not value any subject as a zero or a ten. This implies that architects need to be well informed about many topics, but not necessarily a specialist in any of them. We need to be familiar with them, but not necessarily work on implementing them. Many more topics can be added to these graphics, but it hopefully served to help clarify the profile of front-end architecture work.

This could be an irrelevant observation, except for the fact that the overly broad umbrella of software architecture is causing both unrealistic and low-quality plans. Unrealistic plans have low feasibility and little consideration of the aspects of practical implementation. Low quality means teams did not contemplate everything that should've been considered – not because of a lack of breadth, but for the lack of depth of knowledge in front-end technologies.

My goal with these graphics is to show the multidisciplinary nature of architectural work and how it varies across companies. It is up to the employer, as much as it is the architect, to define expectations. Maybe a genius software architect could achieve all the front-end architectural work with excellence, but companies can't reliably depend on that happening. In light of this fact, and to make work environments less stressful while reducing decision fatigue (which can lead to bad choices), front-end architecture is better off as its own field.

Front-end architecture is not only about collaboratively creating grand plans, but also to maintain a close watch over the multiple projects in the company through a sober and active process of "strategic management." In other words, managers needed to find ways to constantly steer the projects back onto the road. The issue isn't that well-architected projects will always get sidetracked, but that the roads change all the time. Each project needs to adapt continuously to remain moving toward success – for both itself and the business as a whole.

Using software engineering to solve computer problems, without focusing on helping businesses achieve their strategic goals, is to fulfill Bill Gates' famous quote: "The computer was born to solve problems that did not exist before."

As you can see, the goal of software architecture was not to propose new tools, create new UML symbols, or to add new constraints to the requirements list. Through software engineering companies were already able to create projects within the expected budget, time, scope, and quality. But was that a fact? Not really. That only seemed possible if no changes in scope were accepted, which was detrimental to the application's success. On the other hand, if changes were allowed, the systems would grow without order, and lower in quality, making maintenance difficult and sometimes impossible.

Software architecture brings a higher level of abstraction to orchestrate the planning and management of computer systems and their parts. Developers can't make every aspect of the systems configurable, extensible and modular since it would significantly increase their complexity, the development time, and the cost.

Software architecture, therefore, aims to find ways to make software and software development practices better suited to satisfy business' needs and goals. It obviously still prioritizes the company's short-term needs, but without compromising its future survival. That is the only way IT departments can go from possible loopholes (struggling even to support basic business needs), to become a promoter of innovation.

Digital transformation does not need to be an expensive "digital remake." Proper architectural work would allow it to become a straightforward "digital evolution."

That kind of architectural work won't happen by accident, and it can't just be done once and for all; it requires constant review of both the plans and the environment (internal and external). It is also not the fruit of one single genius architect. In other words, for that to happen architects should abandon the old normative view of their role and embrace the facilitator mindset.

As facilitators, architects are expected to investigate and compile directives from multiple stakeholders (such as the CIO, corporate IT, business analysts, marketing, and developers), translating findings into technical and actionable plans. Once plans are created, architects should define processes to monitor and measure their efficiency and efficacy, providing recommendations for course-corrections to guarantee that they are still adequate to satisfy business goals. Within our scope, this active work of monitoring and correcting plans is called "strategic management."

I can almost hear the skeptics saying: "Why do we need an architect for that? Can't we just add these concerns to the work managers and engineers are already doing? Won't architects just add one extra layer of complexity to the workflow?"

These are great points. The model where managers and engineers take over the architectural work is sometimes desirable. It can help foster a "startup" mentality, which can promote innovation and speed things up.

In that model, however, both managers and engineers would still need to improve their knowledge and skills in front-end development and architecture. Wearing multiple hats does not mean doing all your work half quality.

A well-established thesis in business administration is that the most effective ideas and strategies emerge from the people working hands-on, not from planners. That is undoubtedly an argument in favor of manager-engineer teams. If front-end architects are mere planners, why should we listen to them?

Business managers are considered hands-on because they are seeing what is happening on the business side, and developers are as well because they are seeing the technical side. Together they can conduct the architectural work and generate amazing plans. However, that needs to be intentional and formalized through purposeful and mindful weekly meetings. Let us call them "strategy and innovation meetings" (SIMs).

During these meetings, many strategies can be formulated through active discussions. However, once the meetings are over, someone needs to be alert and trained to identify and capture strategies (ideas) that emerge spontaneously. Companies need to be intentional and consistent in identifying and capturing insights, success cases, failed attempts, rejected ideas, and many other things that will not only save time but also increase plan quality.

The startup approach is really enticing. However, managers and engineers are often too busy for any extra hat. They could certainly make it to the meetings, but how could they do all of their work and still keep themselves updated with new languages, new browser versions, new platforms, new standards, new testing tools, new debugging tools, new UX tools, new trends, discontinued projects, newly found security vulnerabilities, innovations and so on?

More than that, in architecture work we are not required just to have knowledge of technologies and terminologies, but also to: a) learn their implications, b) ruminate over their correlations and possible synergy with technologies already adopted by the company; c) evaluate their side effects and interactions; d) analyze their ecosystem, community, learning curves and learning resources; e) check their future-proofness; f) verify their workforce availability; g) examine their licensing constraints; h) check their security aspects; i) contrast them with other possible solutions; j) know their pitfalls and challenges for different use cases; and more. The list goes on and on.

That amount of effort could lead managers and engineers to neglect their primary functions, in which case it would be good to have an "intermediate layer" – the architects. But the architects need to follow good practices and be well prepared; otherwise they risk becoming the "ignorant layer" that clogs the engine.

In short, if we spread architectural responsibilities between managers and engineers, we risk either their primary roles becoming neglected or the architectural work becoming sloppy. Both managers and engineers need to have the capacity and to be intentional with the work and studies related to both hats. Even the most exceptional professionals could involuntarily favor one role more than the other. And while it is easy to see when a developer is not finishing their regular workload, it is more difficult to notice if the architectural work is being done correctly or not.

That is why my proposal for front-end architecture has the following two foundations:

a) It needs to be intentionally and consciously done;

b) It needs to be strategically and smartly done;

By intentional I mean that both dedicated architects and manager-engineer teams need to have checks and balances to make sure that the work is being conducted properly. By strategic I mean that architectural plans should be dynamic, considering more than just the short-term technical needs. In other words, our objective should not be to create amazing plans but to help the company succeed, even if great plans need to be redone entirely.

To add strategic thinking into this new concept of front-end architecture, we first need to remove the false assumption that changes are impossible to predict. Strategic thinking might be a new thing to people with a strictly technical background, but it is a well-known and widely established practice in the business world.

What are we trying to solve with this? Well, a system can be really well done and still be difficult to change. Why? Because we don't always prepare it to expand in the direction in which it will actually grow. This leads to maintenance nightmares or, very often, the need to build a new system entirely.

Therefore, this front-end architecture approach has an incredible potential to add a lot of business value while solving many known issues, varying from the rigidity of basic software design to the unrealistic nature of software architecture plans.

The old popular and reductive view of front-end architecture – that it is just about selecting a tech-stack, setting up dev environments, defining a data flow strategy, and organizing the module structure – is no longer an option. That is clearly not enough to guarantee the success of our front-end projects.

As I said earlier in this chapter, defining front-end architecture is difficult, but can get easier through the usage of mind maps and other resources that list the work and responsibilities of architects. As you move along through the next chapters, you will gain a better understanding of what it is all about.

My proposal for defining front-end architecture is not normative. You can and should examine everything, adapting it to your specific case. I also hope that it will get the front-end community excited about furthering the discussions on the subject. One thing is clear to me though: Front-end architecture has an extraordinary and highly strategic opportunity to transform organizations of all sizes and sectors, not only to reduce product cost and time to market but also to promote innovation. However, that requires solid fundaments, many of which will be covered in this book.

2. A Case for Front-End Architecture

If you are a front-end developer, maybe you don't need any more convincing about the value of front-end architecture. However, we often need to explain our reasoning to business managers, investors, peers, and others. So let us reaffirm why we need this.

Every now and then I meet people who are against this level of specialization. They say: "Wasn't it enough to split web development into back end and front end? Why the new title? Isn't it just ego?" I counter with another question: Is cardiology a result of ego or of a need? Architectural work, regardless of the reason that moved us to get involved in it, is a necessity.

Currently, front-end development is just too important, too big, too complex, and too expensive to be treated as a mere sub-part of the software or web development pipeline. Here are some examples of topics that are important for front-end architects to know:

Progressive Web Apps ❧ Accelerated Mobile Page ◆ Web Services ❧ Web Workers ❧ Web Drivers ❧ Web Components ❧ Cross-browser Compatibility ❧ Shadow DOM ❧ Hyper-HTML ❧ Lit-HTML ❧ Bazel ❧ TensorFlow ❧ AWS Lambda ❧ Tree Shaking ❧ Serverless ❧ WebRTC ❧ WebUSB ❧ WebNFC ❧ WebMidi ❧ Websockets ❧ Isomorphic JavaScript ❧ Storybook ❧ GraphQL ❧ Cordova ❧ Crosswalk ❧ Gatsby ❧ Abstract Syntax Tree ❧ SEO ❧ Accessibility ❧ WebVR ❧ WebAR ❧ Rust ❧ WebAssembly ❧ Cypress ❧ Applitools

This is just a small sample of the most prominent topics. They will all influence our front-end architecture plans to some degree.

Some of these topics are conceptual, like architectural patterns, licensing and policies, while others will be quite technical, such as Canvas, WebGL, Drag and Drop API, Offline concerns and strategies, History API, Push Notifications, Geolocation API, Camera API, Media API, Sensor API, Text

Track API, High Resolution Time API, Performance Timeline API, Navigation Timing API, User Timing API, Resource Timing API, Vibration API, Battery Status API, Page Visibility API, Web Animation API, Resize Observer, Mutation Observer, Performance Observer, Intersection Observer, and so on.

We can’t assume that managers or developers are up-to-date on all of it. If we don’t know what those things are and what implications they have on the systems’ architecture and over daily hands-on work, we can’t recommend them. Also, a shallow knowledge about them will not allow us to explore their potential for creative architectural solutions.

Notice that I didn't even cover frameworks, libraries, language supersets or tools. A basic list would include Svelte, VueJS, Angular, React, Ember, Meteor, Electron, Yew, Flutter, Polymer, D3JS, RxJS, Tensorflow.js, Brain.js, Babel, Koa, Webpack, Yarn, Underscore, Jasmine, Mocha, Cucumber, Istanbul, Capybara, TypeScript, Flow, CoffeeScript, Dart, Elm, and more.

DID YOU KNOW? *According to the HTTP Archive, the average total page load is currently about 2.3 Mb? This is almost the same size as the installation of the classic 3D game Doom (shareware version). We went from 4 kb, which was the size of the first web page ever created (by Tim Berners-Lee), to 2390 kb – a 59,750% increase! There are many factors involved, of course, but it helps put into perspective the sheer amount of data we are expected to manage.*

There are also other topics, such as responsive web design, adaptive web design, OOCSS, BEMCSS, security, server-side rendering, programming paradigms, gamification, browser rendering processes, JavaScript optimization, accessibility, internationalization, A/B testing, usability tests, unit testing, integration testing, end to end testing, visual regression testing, user experience, authentication strategies, advanced debugging techniques, and more.

Finally, what could we say about the implications and possible relationships between front-end development and mobile development, IoT, 3d printing, machine learning, artificial intelligence and even quantum computing? Should we be concerned with those? Are there any zero or low-cost steps we can take to help us leverage them in the future?

As architects, our goal is to go beyond just knowing what these things are. We need to have a good grasp on how we can leverage them, their constraints and limitations, their stage of maturity, their future-proofness, their development speed, how and when they can be combined together, the type and availability of workforce necessary, their community, their learning resources and learning curves, their alignment with other technologies already adopted by the company, and more.

I hope this long and frightening list of terms helps illustrate how vast the front-end world already is. The importance of going beyond the basics of software design while, at the same time, being more focused than software architecture, with enough depth of knowledge of front-end technologies, focusing on business goals and tying everything together with strategic thinking.

A front-end architect does not need to be specialized in all these topics but should have enough depth in them to make their recommendations both valuable and realistic. For example, without knowledge of debugging tools, workforce availability, framework performance, the company's current and future projects, third-party products and their licensing restrictions, internal policies, budget, deployment processes, quality assurance guidelines, browser rendering strategies, and more, it could be difficult to make an accurate decision between native or hybrid mobile app development. In this case, the decision is taken based on trend, hype or personal interest.

Managers may have knowledge of some of these criteria, while developers have knowledge of others. Ideally, they would communicate well, without anyone holding back contributions out of fear, timidity, tiredness, or from simply assuming that everybody already knows "that" so it is not worth mentioning.

It can be very difficult to keep yourself up-to-date in all these topics, not only before the projects start but also while they are being implemented. Architectural decisions depend on information quality and active work. Whether you are an architect or a manager-engineer team, these topics are vital for you and will consciously or unconsciously affect your architectural decisions. For example:

a) The Performance interface, from the High Resolution Time standard, could help you prove that you need (or don't need) to implement server-side rendering;

b) Technical knowledge of the web components' maturity level and browser adoption might encourage you to use Angular Elements over Polymer;

c) By leveraging your knowledge of custom Cordova Plugins, you might decide to drop the idea for big dedicated OS specific teams, and just hire a couple of talented Swift and Java programmers to work with the existing group of front-end developers;

d) By combining Web Assembly and Crosswalk, you might decide to make your new app entirely in Cordova;

e) You might not need to create image sprites or complex code bundles if you and your customers could leverage HTTP 2 available on NodeJS v10; or

f) You might consider using Applitools with SauceLabs to reduce the amount of e2e and unit tests necessary, freeing up your work capacity while, at the same time, increasing testing quality.

I often see people blinded by the fact that HTML, CSS, and JavaScript are relatively easy to learn and implement. They jump right into the coding phase of their applications with little or no architectural work whatsoever. But as I've mentioned, while these languages are easy to get started with, they are certainly difficult to master. Indeed, it is because of their simplicity that the architectural work becomes even more necessary.

Rushing into development without investing in architecture can have other downsides. It includes risks such as a) the company dying before taking off (because of maintenance traps); b) a burned brand image from product malfunction; c) huge lawsuits; and d) lost opportunities, since it would become increasingly more difficult to add new functionalities to the product or to make it portable to new platforms. When planning is done correctly, you don't lose time, you gain it.

Some technologies have a future, while others will soon be abandoned. Some have a sizable community, while others don't. Some will leverage the expertise of the company's workforce, while others cause a drastic drop in productivity. Some might require changes to pre-existing internal systems, while others work seamlessly with them. Some may achieve better browser speed, while others produce better development speed and tooling. Some adapt well to the company's structure and culture, while others do not, causing all sorts of clogs in the machine. Yes, human aspects will affect our architectural plans, whether we like it or not.

At this point, we have yet to cover our biggest issue. If in the past an application had 100,000 lines of code, now a big chunk of that has moved to the front end. Even when the back end still keeps some of the business logic, the front end normally replicates it, so users can have early validation for their inputs. How can we achieve that with languages that are clearly much less robust than Java, PHP, and C#?

I have seen cases where 70% of an application's code was moved to the front end. How can JavaScript handle that amount of code? I don't mean only in terms of performance, but also stability and maintainability. JavaScript was not invented with all that we do nowadays in mind. Having JavaScript supersets or languages that can be transpired into JavaScript is a great start, but not the complete solution. We need to be aware of standards and conventions, browser rendering processes, server-side rendering, isomorphic JavaScript, portability, JavaScript engines and optimization, testability, and more.

Even just a few years ago, it was quite common to see a developer joining a company and starting a crusade to convince everyone to throw away their old system and build a new one from scratch. Their justification was that the amount of time it would take to study and modify an existing system would be very close to the amount necessary to create a new one.

Technically that may have been true, but managers knew that writing an app involved way more than writing code. There were countless requirements that were not properly logged, the need for new analysis and approvals from the business side, new testing suites, a tremendous amount of meetings, and so on.

Front-end frameworks and libraries are very helpful in their ability to organize those thousands of lines of code, creating standardization, promoting good practices, and freeing us from the weight of writing a lot of boilerplate code. However, we need to remember that a tool alone cannot guarantee the success of a project.

We've briefly discussed the technical knowledge architectural work requires. Those things, however, need to be contrasted and balanced out with directives given by many other internal and external sources:

1) Business vision, mission, and goals
2) CIO
3) Corporate IT
4) Marketing
5) UX specialists
6) Usability data
7) Business analysts
8) Business managers
9) Product owners

10) Benchmarks
11) W3C and ECMA current and upcoming standards
12) New technologies and trends
13) New products and services
14) New platforms and sensors

When I say that front-end projects are difficult to create and even more difficult to maintain, doesn't it sound just like the "software crisis" from previous decades? And how was it solved? With proper software engineering and software architecture. Similarly, I believe that we can overcome our front-end challenges with a correct approach to both.

We need to fight the urge for simplistic and precipitated answers so we can evaluate each choice carefully. What if the cost of the project was coming out of my pocket instead? Would I still feel confident? Architectural decisions are usually long-lasting, difficult to change and deeply impactful in many areas of an organization. We can’t allow ourselves to be persuaded by hypes, fads or manipulative blog posts.

While junior developers often take sides and become ferocious defendants of specific technologies, as architects we need to keep our minds open and examine everything from many angles. We need the expertise of an engineer and the wisdom of a manager. We simply cannot make decisions based on pressure or persuasive communities.

Front-end technologies emerge faster than we can adapt. New tools and new solutions are being launched every week. New devices and new sensors are popping up all the time. Who is capable of boiling it all down to a practical, yet highly sophisticated, strategic plan? It certainly requires for the front-end architects or the manager-engineer teams to be not only well informed, but also wise, careful and astute.

It is easy to see why front-end architecture requires so much dedication and time for continued education. Someone needs to be watching over the company's architectural decisions; otherwise, those decisions will be made anyway, but without the necessary care.

For all intents and purposes, front-end projects are real, browser-based software. I would say they are even more complex than desktop software, since we make them with less robust computer languages, targeting many operating systems and browsers (and their versions), multiple screen sizes and resolutions, different platforms, often with live internationalization, etc. – all at the same time within an ever-changing ecosystem.

These are significant challenges, but not bad things. Actually, it is because of these challenges, along with the fact that front end is what gives a face to our products, that we can say that front-end development is the most strategic promoter of continuous innovation at the business and technical level.

We obviously need to innovate at all levels constantly, but the front end is certainly the piece that we modify more frequently. We don't always need to change the back end or infrastructure to meet the customers' demands, new aesthetic trends, or new devices. Little delays in the updates of back end and infrastructure can be tolerated, as long as the front end is providing the experience that users expect.

The front end is, ultimately, what reaches the customers; it is the biggest source of innovation and the stronger promoter of customer retention. A company can have great products and services, but if their web applications (or websites) are not captivating, mobile friendly, and modern, people will likely just move on. It is a key piece for making companies leaders in their fields.

On the other hand, we know that being too reactive to market changes can be detrimental for companies. Front-end architects have an exceptional opportunity to help managers identify what the best time is to move ahead with experimentation and implementation of new technologies. Trends come and go, so we either need to reserve our energy for what is most relevant, or we need to have ways to implement them quickly and cheaply.

Adaptation and timing are crucial elements for business success. In a first instance, it might seem that the lack of front-end architecture caused by Agile workflows is promoting both easy adaptation and speedy changes. The reality, however, is that it causes the opposite. What some people call "organic growth" is better defined as "unorganized growth," since it leads to messy projects that are error-prone, as well as difficult to maintain, update and audit, with lower performance and confidence. Only through a smart and strategic front-end architecture can our projects really accommodate frequent changes in the speed, cost, and quality expected.

The adaptability, maintainability, alignment with future market changes, and many other things we want and need won't just happen spontaneously. It is through the care and practice of professional front-end architecture that not only can we stop the bleeding (preventing repeated mistakes), but also steer the company toward progress.