37 THINGS ONE ARCHITECT KNOWS ABOUT IT TRANSFORMATION

A CHIEF ARCHITECT'S JOURNEY

Gregor Hohpe

An Architect Elevator Guide

37 Things One Architect Knows About IT Transformation

A Chief Architect's Journey

Gregor Hohpe

This book is available at http://leanpub.com/37things

This version was published on 2024-07-28

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Gregor Hohpe by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Just learned 37things @ghohpe knows about large-scale IT organizations and architecture: www.ArchitectElevator.com/book

The suggested hashtag for this book is #ArchitectElevator.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#ArchitectElevator

Contents

About this Book

Many large enterprises are facing pressure from the rapid digitalization of the world: "digital disruptors" attack unexpectedly with brand-new business models; the "FaceBook generation" sets dramatically different user expectations; and new technologies have become available in the cloud to everyone with a credit card. This is tough stuff for enterprises that have been, and still are, very successful, but are built around traditional technology and organizational structures. "Turning the tanker", as the need to transform is often described, has become a board room-level topic in many traditional enterprises: "we need new digital products!", "we need to renew our tech stack!", "we need to de-layer our organization!", "we need to become agile!", "we need to become digital!" are the battle cries that often emerge out of the board room. Not as easily done as said.

Chief IT Architects and CTOs play a key role in such a digital transformation endeavor. They combine the technical, communication, and organizational skill to understand how a tech stack refresh can actually benefit the business, what "being agile" and "DevOps" really mean, and what technology infrastructure is needed to assure quality while moving faster. Their job is not an easy one, though: they must maneuver in an organization where IT is often still seen as a cost center, where operations means "run"as opposed to "change", and where middle-aged middle-management has become cozy neither understanding the business strategy nor the underlying technology. It's no surprise then that software / IT architects

have become some of the most sought-after IT professionals around the globe.

With such high expectations, though, what does it take to become a successful Chief Architect? And once you get there, how do you get support and keep up? When I became a Chief IT Architect, I wasn't expecting any magic answers, but I was looking for a book that would at least help me not having to reinvent the wheel all the time. I attended many useful CIO/CTO events, but most focused on what the business wanted to achieve and little on how to actually accomplish it on a technical level. Having been unable to find such a book, I decided to collect my experience of over two decades as software engineer, consultant, startup co-founder, and chief architect into a book of my own.

What Things Will I Learn?

The five major sections in this book correspond to the aspects a chief architect has to tackle to effectively support a large IT transformation:

- The role and qualities of an enterprise or IT architect
- The value of architecture in a large enterprise
- Communicating to a variety of stakeholders
- Understanding organizational structures and systems
- Transforming traditional organizations

This isn't a technical book. It's a book about how to grow your horizon as an architect to better utilize your technical skill in large organizations. This book won't teach you how to configure a Hadoop cluster or how to setup container orchestration with Docker. Instead, it will teach you how to reason about large-scale architectures, how to ensure your architecture benefits the business strategy, how to leverage vendors' expertise, and how to communicate to upper management.

Are the Things Proven to Work?

As the title may suggest, this is a personal and somewhat opinionated book: it's based on my daily experiences of two decades in IT, which led me through being a start-up co-founder (lots of fun, not lots of money), system integrator (made tax audits more efficient), consultant (lots of PowerPoint!), author (collecting and documenting insights), Internet software engineer (building the future) and chief architect of a large multi-national organization (tough, but rewarding).

I felt that taking a personal account of IT transformation is appropriate because architecture is by nature a somewhat personal business. In building architecture you can easily identify the architect from afar: white box - Richard Meier, all crooked - Frank Gehry, looks like made from fabric: Zaha Hadid. While not as dramatic, every (Chief) IT architect also has his or her personal emphasis and style that's reflected in their works.

The collection of insights that make up this book reflect my personal point of view but are written such that the "nuggets" can be easily extracted and put to broader use. Architects are busy people. I therefore tried to package my insights so that they are easy to consume and even a bit fun to read.

Tell me a Story

If you are looking for a scientifically proven, repeatable "method" of transforming a technical organization, you may be disappointed. This book's structure is rather loose and you may even be annoyed to have to read through little anecdotes when all you want is the one bit of advice you need in order to be successful.

I purposefully chose to structure the book as a collection of stories. As our world is becoming more complex and difficult to understand, telling stories is one of the best ways to engage and teach. Studies

have shown that people remember stories much better than sheer facts and there appears to be evidence that listening to a story activates additional parts of our brain that helps with understanding and retention. Aristotle already knew that a good speech contains not only *logos*, the facts and structure, but also *ethos*, a credible character, and *pathos*, emotions. Emotions distinguish a story from a scientific analysis.

To transform an organization you don't need to remember facts and solve mathematical equations. You need to move people. That's why you need to be able to tell a good story. It's fine to start out by using some of the attention-catching slogans from this book ("Zombies will eat your brain!") and later supplementing them with your own stories. Stories get attention and evoke emotions - the best way to get people moving. Have you seen people cry and laugh when watching movies, even though they know exactly that the story is fictitious and all acting is fake? That's the power of storytelling in action.

Architects

Overhead or Corporate Savior?

Architects have an exciting, but sometimes challenging life in corporate IT. Many managers and technical staff consider them overpaid ivory tower residents who, detached from reality, bestow their thoughts upon the rest of the company with slides and wall-sized posters, while their quest for irrelevant ideals causes missed project timelines.

Still, IT architects have become some of the most sought-after IT professionals as traditional enterprises are looking to transform their IT landscape. At the same time, many of the most successful digital companies have a world-class software and systems architecture, but don't have architects at all. So what makes a successful architect?

What Architects Are Not

Sometimes it's easier to describe what something *isn't* rather than trying to come up with an exact definition of what it is. In the case of architects, exaggerated expectations can paint a picture of someone who solves intermittent performance problems in the morning while transforming the enterprise culture in the afternoon. This leads to a scenario where architects are pulled into several roles that clearly miss the purpose of being an architect:

Firefighter - Many managers expect architects to be able to troubleshoot and solve any crisis based on their broad understanding of the current system landscape. An architect shouldn't ignore production issues because they provide valuable feedback into possible architectural weaknesses. But an architect that runs from one fire drill to the next won't have any time to do actual architecture. Architecture requires *thinking*, which cannot be achieved in 30-minute time slots.

Senior Developer - Developers often feel they need to become an architect as the next step in their career (and their pay grade). However, becoming an architect and a superstar engineer are two different career paths, with neither being superior to the other. Architects tend to have a broader scope, including organizational and strategic aspects, whereas engineers tend to specialize and deliver running software. Ideally, the Chief IT Architect in a large organization is good friends with the senior developers.

Project Manager - Architects must be able to juggle many distinct, but interrelated topics. Their decisions also take into account, and impact, project time lines, staffing, and required skill sets. As a result, upper management often comes to rely on the architect for information and decisions regarding the project, especially if the project manager is busy *filling out status report templates*. This is a slippery slope for an architect because it's valuable work, but distracts from the architect's main responsibility.

Scientist - While architects need to sport a sharp intellect and must be able to *think in models and systems*, an architect's role is to make decisions related to concrete projects and business initiatives. This often separates the role of a *Chief Architect* from that of a *Chief Scientist*, although the lines are blurry – I know a few Chief Scientists who are very hands-on. Personally, I prefer the title *Chief Engineer* to highlight that architects produce more than paper. Lastly, while scientists have a tendency to make things more complex and difficult to understand, an architect's job is to *make complex topics easy to understand*.

Measuring an Architect's Value

Someone once asked me what KPI (Key Performance Indicator) should be used to measure an architect's value. When he suggested the number of decisions made, I was a little stunned and at the same time convinced that this wasn't the right one. Making decisions is important, but *avoiding decisions* can also be a key element of being an architect.

Instead, I suggest two alternative ways to measure value. First, if your systems are still running and can absorb change at a reasonable rate after 5 years, there was likely a good architect involved. For a more concrete description, senior architects in the enterprise work at three levels:

- Define the *IT Strategy*, e.g., by assuring that the IT landscape adequately supports the business strategy or defining a set of necessary IT characteristics for systems to be built or bought. Strategy also includes "retiring" systems (in the *Blade Runner* sense of the word) lest you want to *live among Zombies*.
- Exercise *Governance* over the IT landscape to achieve harmonization, complexity reduction, and to make sure that systems integrate into a meaningful whole. Governance occurs through architecture review boards and *inception*.
- Deliver *Projects* to stay grounded in reality and receive feedback on decisions from real project implementations. Otherwise *control remains an illusion*.

Architects as Change Agents

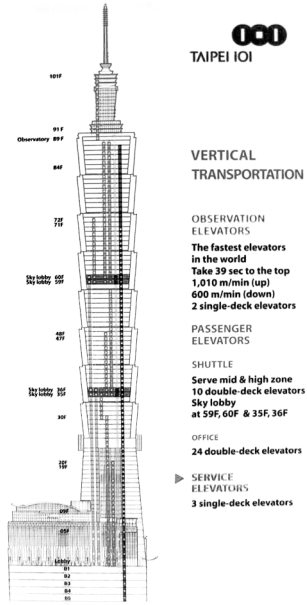
As the digital economy changes the rules of the game, an architect's role also fundamentally changes. Today's large-scale architects are

a critical enabler of IT transformation. To do so, they must be equipped with a special set of skills beyond just technology so that they can…

- …transcend organizational levels by *riding the architect elevator*.
- …adopt multiple personas which may resemble *movie characters*.
- …*connect business and IT*.
- …bring more than skill as that's just one of the *three legs they stand on*.
- …have *good decision discipline* in face of uncertainty.
- …*question everything* to get to the root of problems.

The Architect Elevator

From the penthouse to the engine room and back



Tall buildings need someone to ride the elevator

A Missing Link

Architects play a critical role as a connecting and translating element, especially in large organizations where departments speak different languages, have different viewpoints, and drive towards

conflicting objectives. Many layers of management only exacerbate the problem as communicating up-and-down the corporate ladder resembles the "telephone game"[1]. The worst case scenario materializes when people holding relevant information or expertise aren't empowered to make decisions while the decision makers lack relevant information. Not a good state to be in for a corporate IT department, especially in the days where technology has become a driving factor for most businesses.

The Architect Elevator

Architects can fill an important void in large enterprises: they work and communicate closely with technical staff on projects, but are also able to convey technical topics to upper management *without losing the essence of the message*. Conversely, they understand the company's business strategy and can translate it into technical decisions that support it.

If you picture the levels of an organization as the floors in a building, architects can ride what I call the *architect elevator*: they ride the elevator up and down to move between a large enterprise's board room and the *engine room* where software is being built. Such a direct linkage between the levels has become more important than ever in times of rapid IT evolution and digital disruption.

Stretching the analogy to that of a large ship, if the bridge spots an obstacle and needs to turn the proverbial tanker, it will set the engines to reverse and the rudder turned hard to starboard. But if in reality the engines are running full speed ahead, a major disaster is preprogrammed. This is why even old steamboats had a pipe to echo commands directly from the captain to the boiler room and back. In large enterprises architects have to play exactly that role!

[1] In the telephone game children form a circle and relay a message from one child to the next. When the message returns to the originator they realize that it has completely changed along the way.

Some Organizations Have More Floors Than Others

Coming back to the building metaphor, the number of floors an architect has to ride in the elevator depends on the type of organization. Flat organizations may not need the elevator at all – a few flights of stairs are sufficient. This may also mean that the up-and-down role of an architect is less critical: if management is keenly aware of the technical reality at the necessary level of detail and technical staff have direct access to senior management, fewer "enterprise" architects are needed. One could say that digital companies live in a bungalow and hence don't need the elevator.

However, classic IT shops in large organizations tend to have many, many floors above them. They live in a skyscraper so tall that a single architect elevator may not be able to span all levels. In this case it's OK if a technical architect and an enterprise architect meet in the middle and cover their respective "half" of the building. The value of the architects in this scenario shouldn't be measured by how "high" they travel, but by how many floors they span. It's a common mistake in large organizations for the folks in the penthouse to only see and value the architects in the upper half of the building. Conversely, many developers or technical architects consider such "enterprise" architects less useful because they don't code. This can be true in some cases – such architects often enjoy life in the upper floors so much that they aren't keen to take the elevator ever down again. But an "enterprise" architect who travels half way down the building to share the strategic vision with technical architects can have a significant value.

Not a One Way Street

Invariably you will meet folks who ride the elevator, but only once to the top and never back down. They enjoy the good view from the penthouse too much and feel that they didn't work so hard to

still be visiting the grimy engine room. Frequently you can identify these folks by statements like: "I used to be technical". I can't help but retort: "I used to be a manager" (it's true) or "Why did you stop? Were you no good at it"? If you want to be more diplomatic (and philosophical) about it, cite Fritz Lang's movie *Metropolis* where the separation between penthouse and engine room almost led to the city's complete destruction before people realized that "the head and the hands need a mediator". In any case: the elevator is meant to be ridden up and down. Eating caviar in the penthouse while the basement is flooded isn't the way to transform corporate IT.

Riding the elevator up-and-down is also an important mechanism for the architect to obtain feedback on decisions and to understand their ramifications at the implementation level. Long project implementation cycles don't provide a good Learning Loop and can lead to an *Architect's Dream, Developer's Nightmare* scenario. Allowing architects to only enjoy the view from high up, invariably leads to the dreaded authority without responsibility[2] anti-pattern. This pattern can only be broken if architects have to live with, or at least observe, the consequences of their decisions. To do so, they must keep riding the elevator.

High-Speed Elevators

In the past, IT decisions were fairly far removed from the business strategy: IT was pretty "vanilla" and the main parameter (or KPI = Key Performance Indicator) was *cost*. Therefore, riding the elevator wasn't as critical as new information was rare. Nowadays, though, the linkage between business goals and technology choices has become much more direct, even for "traditional" businesses. For example, the desire for faster time to market to meet competitive pressures translates into the need for an elastic cloud approach to computing, which in turn requires applications that scale horizontally and thus should be designed to be stateless. Targeted content

[2] http://c2.com/cgi/wiki?AuthorityWithoutResponsibility

on customer channels necessitates analytical models which are tuned by churning through large amounts of data via a Hadoop cluster, which favors local hard drive storage over shared network storage. The fact that in one or two sentences a business need has turned into application or infrastructure design highlights the need for architects to ride the elevator. Increasingly they have to take the express elevator, though, to keep up with the pace at which business and IT are intertwined.

In traditional IT shops, the lower floors of the building can be exclusively occupied by external consultants, which allows enterprise architects to avoid getting their hands dirty. However, because it focuses solely on efficiency and ignores *Economies of Speed*, it's a poor setup in times of rapid technology evolution. Architects who are used to such an environment must stretch their role from being pure consumers of vendors' technology roadmaps to actively defining it. To do so, they must develop their own *IT World View*.

Other Passengers

If you are riding the elevator up and down as a successful architect, you may encounter other folks riding with you. You may, for example, meet business or non-technical folks who learned that a deeper understanding of IT is critical to the business. Be kind to those folks, take them with you and show them around. Engage them in a dialog – it will allow you to better understand business needs and goals. They might even take you to the higher floors you haven't been to.

You may also encounter folks who ride the elevator down merely to pick up buzzwords to sell as their own ideas in the penthouse. We don't call these people architects. People who ride the elevator but don't get out are commonly called *lift boys*. They benefit from the ignorance in the penthouse to pursue a "technical" career without touching actual technology. You may be able to convert some of these folks by getting them genuinely interested in what's going

on in the engine room. If you don't succeed, it's best to maintain the proverbial elevator silence, avoiding eye contact by examining every ceiling tile in detail. Keep your "elevator pitch" for those moments when you share the cabin with a senior executive, not a mere messenger.

Dangers of Riding the Elevator

You would think that architects riding the elevator up and down are highly appreciated by their employer. After all, they provide significant value to businesses transforming their IT to better compete in a digital world. Surprisingly, such architects may encounter resistance. Both the penthouse and the engine room may actually have grown quite content with being disconnected: the company leadership is under the false impression that the digital transformation is proceeding nicely while the folks in the engine room enjoy the freedom to try out new technologies without much supervision. Such a disconnect between penthouse and engine room resembles a cruise ship heading for an iceberg with the engines running at full speed ahead: by the time the leadership realizes what's going on, it's likely too late.

I was once criticized by the engine room for pushing corporate agenda against the will of the developers while at the same time corporate leadership chastized me for wanting to try new solutions just for fun. Ironically, this likely meant I found a good balance.

One can liken such organizations to the Leaning Tower of Pisa where the foundation and the penthouse aren't vertically aligned. Riding the elevator in such a building is certainly more challenging. When stepping into such an environment, the elevator architect must be prepared to face resistance from both sides. No one ever said being a disruptor is easy, especially as *systems resist change*.

The best strategy in these situations is to start linking the levels carefully, waiting for the right moment to share information. For example, you could start by helping the folks in the engine room convey to management what great work they are doing. It'll give them more visibility and recognition while you gain access to detailed technical information.

Other corporate denizens not content with you riding the elevator can be found on the middle floors: seeing you whiz by to connect leadership and the engine room makes them feel bypassed. Thus, the organization has an "hourglass" shape of appreciation for your work: top management sees you as a critical transformation enabler while the folks in the engine room are happy to have someone to talk to who actually understands and appreciates their work. The folks in the middle, though, see you as a threat to their livelihood, including their children's education and their vacation home in the mountains. This is a delicate affair. Some may even actively block you on your way: being stopped at every floor to give an explanation, aka aligning, makes riding the elevator not really faster than taking the stairs.

Lastly, because folks riding the elevator are rare, being good at one thing often leads others to conclude that you aren't good at anything else. For example, architects giving meaningful and inspiring presentations to management are often assumed to not be great technologists, even though that's the very reason their presentations are meaningful. So, every once in a while you're going to want to let the upper floors know that you can hold your own down in the engine room.

Flattening the Building

Instead of tirelessly riding the elevator up and down, why not get rid of all those unnecessary floors? After all, the digital companies your business is trying to compete with have much fewer floors. Unfortunately, you can't simply pull some floors out of a building.

And blowing the whole thing up just leaves you with a pile of rubble, not a lower building. The guys on the middle floors are often critical knowledge holders about the organization and IT landscape, especially if there's a large *black market*, so the organization can't function without them in the near term.

Flattening the building little-by-little may be a sound long-term strategy, but it would take too long because it requires fundamental changes to the company culture. It also changes or eliminates the role played by the folks inhabiting the middle floors, who will put up a fierce resistance. This isn't a fight an architect can win. However, an architect can start to loosen things up a little bit, for example by getting the penthouse interested in information from the engine room, providing faster feedback loops, and reducing the number of PowerPoint status updates given by middle management.

Architecture

Defining Architecture

There appear to be almost as many definitions of IT architecture as there are practicing architects. Most software architecture definitions cite a system's elements and components plus their interrelationships. In my view, this covers only one aspect of architecture. First, IT architecture is much more than software architecture: unless you outsourced all your IT infrastructure into the public cloud, you need to architect networks, data centers, computing infrastructure, storage, and much more. Second, defining which "components" you are focusing on constitutes a significant aspect of architecture.

A manager once stated that he can't understand the many network issues despite all the network stuff "being there". His view was a physical one: Ethernet cables plugged into servers and switches. The complexity of network architecture, however, lies in virtual network segregation, routing, address translation, and much more.

Architecture as a Function

In large enterprises, the word "architecture" tends to describe both the structure of technical systems and an organizational unit: "we

are setting up enterprise architecture." Most of my discussions on "architecture" focus on the system properties. For organizational aspects, I speak about "architects" - it's based on humans, after all.

There Always is an Architecture

It's worth pointing out that any system has an architecture, which puts statements like "we don't have time for architecture" into a questionable light. It's simply a matter of whether you consciously choose your architecture or whether you let it happen to you, with the latter invariably leading to the infamous *Big Ball of Mud*[3] architecture, also referred to as *Shanty Town*. While that architecture does allow for rapid implementation without central planning or specialized skills, it also tends to ignore critical infrastructure aspects and doesn't make for a great living environment. Fatalism isn't a great enterprise architecture strategy, so I suggest you pick your architecture.

The Value of Architecture

Because there always is an architecture, an organization should be clear on what it expects from setting up an architecture function. Setting up an architecture team and then not letting them do their job, for example by routinely subjecting architecture decisions to management decisions, is actually worse than intentionally letting things drift into a "big ball of mud": you pretend to define your architecture, but in reality you don't. Worse yet, good architects don't want to be in a place where architecture is seen as a form of corporate entertainment. If you don't take architecture seriously, you won't be able to attract and retain serious architects.

IT management often believes that "architecture" is a long-term investment that will only pay off far into the future. While this is true for some aspects, e.g. managed system evolution over time,

[3] http://www.laputan.org/mud/

architecture can also pay off in the short-term, e.g. when you can accommodate a customer requirement late in the development cycle, when you gain leverage in vendor negotiations because you avoided lock-in, or when you can easily migrate your systems to a new data center location. Good architecture can also make a team more productive by allowing concurrent development and testing of components. Generally, good architecture buys you flexibility. In a rapidly changing world, this seems like a smart investment.

As most upper management is well versed in financial models, I often describe investing in architecture as the equivalent to *buying an option*: an option gives the buyer the right, but not the obligation to execute on a contract, e.g. buying or selling a financial instrument, in the future. In IT architecture, the option allows you to make changes to the system design, the run-time platform, or functional capabilities. Just as in the financial world, options aren't free - the ability to act on a contract in the future, when more information is available, has a value and therefore a price. I don't think the *Black-Scholes model*[4] accurately computes the value of large-scale IT architecture, but it makes apparent that architecture has a measurable value and can therefore demand a price.

Principles Drive Decisions

Architecture is a matter of trade-offs: there rarely is one single "best" architecture. Architects therefore must take the context into consideration when making architectural decisions and aim to achieve conceptual integrity, i.e. uniformity across system designs. This is best accomplished by selecting a well-defined set of architecture principles which are consistently applied to architectural decisions. Deriving these principles from a declared architecture strategy assures that the decisions support the strategy.

[4] https://en.wikipedia.org/wiki/Black%E2%80%93Scholes_model

Vertical Cohesion

A good architecture is not only consistent across systems, but also considers all layers of a software and hardware stack. Investigating new types of scale-out compute hardware or software-defined networks is useful, but if all your applications are inflexible monoliths with hard-coded IP addresses you gain little. Architects therefore not only need to *Ride the Elevator* across the organization but also up and down the technology stack.

Architecting the Real World

The real world is full of architectures; not just building architectures, but also cities, corporate organizations, or political systems. The real world has to deal with many of the same issues faced by large enterprises lack of central governance, difficult to reverse decisions, complexity, constant evolution, slow feedback cycles. Architects should walk through the world with open eyes, always looking to learn from the architectures they encounter.

When defining architecture in large organizations, architects need to know more than how to draw UML diagrams. They need to:

- gain architecture insights while *Waiting in the line at a Coffee Shop*.
- tell whether *Something Is Architecture* in the first place.
- tackle complexity by *Thinking in Systems*.
- know that *Configuration isn't better than coding*.
- hunt zombies so they *Don't have their brain eaten*.
- navigate the IT landscape with an *Undistorted world map*.
- automate everything so that they *Never have to send a human to do a machine's job*.
- think like software developers as *Everything becomes software-defined*.

Is this Architecture?

Look for decisions!

Would you pay an architect for this?

Part of my job as Chief Architect is to review and approve system architectures. When I ask teams to show me "their architecture", I frequently don't consider what I receive an architecture document. The counter-question "what do you expect?" isn't so easy for me to answer: despite many formal definitions, it isn't immediately clear what architecture is or whether a document really depicts an architecture. Too often we have to fall back to the "I know it when I see it" test famously applied to obscene material by the Supreme Court. We'd hope that identifying architecture is a more noble task than identifying obscene material, so let's try a little harder. I am not a big believer in all-encompassing definitions but prefer to use lists of defining characteristics or tests that can be applied. One of my favorite tests for architecture documentation is whether it contains any non-trivial decisions and the rationale behind them.

Defining Software Architecture

Enough attempts at defining software architecture have been made that the Software Engineering Institute (SEI) maintains a reference page of software architecture definitions[5].

The most widely used definitions include the one from Garlan and Perry from 1995:

> *The structure of the components of a system, their interrelationships, and principles and guidelines governing their design and evolution over time*

In 2000 the ANSI/IEEE Std 1471 chose the following definition: (adopted as ISO/IEC 42010 in 2007):

> *The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution*

The Open Group adopted a variation thereof for TOGAF:

> *The structure of the components, their interrelationships, and principles and guidelines governing their design and evolution over time*

One of my personal favorites is from Desmond D'Souza's and Alan Cameron Wills' book[6]:

> *Design decisions about any system that keep implementors and maintainers from exercising needless creativity*

[5] http://www.sei.cmu.edu/architecture/start/glossary/bibliographicdefs.cfm

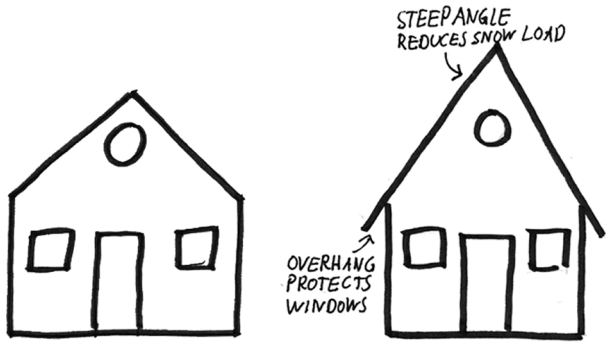
[6] D'Souza, Wills: Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach; Addison-Wesley Professional, 1998

The key point here isn't that architecture should dampen all creativity, but *needless* creativity, of which I witness ample amounts. It also highlights the importance of making decisions .

Architectural Decisions

These well-thought-out definitions aren't easy to apply, though, when someone walks up with a PowerPoint slide showing boxes *and lines*, claiming "this is my system architecture". The first test I tend to apply is whether the documentation contains meaningful decisions. After all, if no decisions needed to be made, why employ an architect and prepare architectural documentation?

Martin Fowler's knack for explaining the essence of things using extremely simple examples motivated me to illustrate the "architectural decision test" with the simplest example I could think of, drawing from the (admittedly limping) analogy to building architecture. I even took it upon myself to supply the artwork to make sure things end up about as basic as one could wish for and to pay homage to Christopher Alexander's Pattern Sketches[7].



Is this architecture?

[7] Alexander: A Pattern Language: Towns, Buildings, Construction; Oxford University Press, 1977

Consider the drawing of a house on the left. It has many of the elements required by the popular definitions of systems architecture: we see the main *components of the system* (door, windows, roof) and their *interrelationships* (door and windows in the wall, roof on the top). We might be a tad thin, though, on *principles governing its design*, but we do notice that we have a single door that reaches the ground and multiple windows, which follows common building principles.

Yet, to build such a house I wouldn't want to pay an architect. This house is "cookie-cutter", meaning I don't see any non-obvious decisions that an architect would have made. Consequently, I wouldn't consider this architecture.

Let's compare this to the sketch on the right-hand side. The sketch is equally simple and poorly drawn and the house is almost the same, except for the roof. This house has a steep roof and for a good reason: the house is designed for a cold climate where winters bring extensive snowfall. Snow is quite heavy and can easily overload the roof of the house. A steep roof allows the snow to slide off or be easily removed thanks to gravity, a pretty cheap and widely available resource. Additionally, an overhang prevents the sliding snow from piling up right in front of the windows.

To me, this is architecture: non-trivial decisions have been made and documented. The decisions are driven by the system context, in this case, the climate: it's unlikely that the customer explicitly stated a requirement that the roof not be crushed. Additionally, the documentation highlights relevant decisions and omits unnecessary noise.

If you believe these architectural decisions were pretty obvious, let's look at a very different house:

Great architecture on a napkin

This house was designed for a different climate, a hot and sunny one, which allows the walls to be made out of glass as insulation against low temperatures is less of a concern. However, walls of glass have the problem that the sun heats up the building, making it feel more like a greenhouse than a residence. The solution? Extending the roof well beyond the glass walls keeps the interior in the shade, especially in summer when the sun is high in the sky. In the winter, when the sun is low on the horizon, the sun reaches through the windows and helps warm the building interior. Also, a flat roof with an overhang isn't a problem in this climate. Again, the architecture is defined by a fairly simple, but fundamental decision documented in an easy-to-understand format that highlights the essence of the decision and the rationale behind it.

Fundamental Decisions Don't Need to be Complicated

If you think the idea of building an overhanging roof isn't all that original or significant, try buying one of the first homes to feature such a design, e.g. the Case Study House No 22 in Los Angeles by architect Pierre Koenig. It's easily in the league of most recognized residential building in Los Angeles or beyond (aided by Julius Shulman's iconic photograph) and surely isn't for sale. You can tour it, though, if you sign up far in advance. Significant architecture decisions may look obvious in hindsight

but that doesn't diminish their value. No one is perfect, though: UCLA PhD students have measured that the overhang works better on the south-facing facade than west or east[8].

Fit for Purpose

The simple house example also highlights another important property of architecture: rarely is an architecture simply "good" or "bad". Rather, architecture is fit or unfit for purpose. A house with glass walls and a flat roof may be regarded as great architecture, but probably not in the Swiss Alps where it will collapse after a few winters or suffer from a leaking roof. It also doesn't do much good near the equator where the sun's path on the sky remains fairly constant throughout the year. In those regions, you are better off with thick walls, small windows, and lots of air conditioning.

Assessing the context and identifying implicit constraints or assumptions in proposed designs is an architect's key responsibility. Sometimes, architects are described as the people dealing with non-functional requirements. I find that more often than not architects have to deal with *non-requirements*, implicit needs or assumptions that weren't communicated at all.

Even the dreaded Big Ball of Mud[9] can be "fit for purpose", e.g. when you need to make a deadline at all cost and can't care much about what happens afterwards. This may not be the context you wish for, but just like houses in some regions have to be earthquake proof, some architectures have to be management-proof.

Passing the Test

Having stretched the overused building architecture analogy one more time, how do we translate it back to software systems architecture? Systems architecture doesn't have to be something terribly

[8] La Roche: The Case Study House Program in Los Angeles: A Case for Sustainability; in Proc. of Conference on Passive and Low Energy Architecture, 2002

[9] http://www.laputan.org/mud/

complicated. It must include, however, significant decisions that are well documented and are based on a clear rationale. The word "significant" may be open to some interpretation and depend on the level of sophistication of the organization, but "we separate front-end from back-end code" or "we use monitoring" surely have the ring of "my door reaches the ground so people can walk in" or "I put windows in the walls so light can enter". Instead, when discussing architectures let's talk about what isn't obvious or something that involved heavy trade-offs. For example, "do you use a service layer and why?" (some people may find even this obvious) or "why do you use a session-oriented conversation protocol?"

It's quite amazing how many "architecture documents" don't pass this relatively simple test. I hope using the building analogy provides a simple and non-threatening way to provide feedback and to motivate architects to better document their designs and decisions.

Transformation

Bringing change into large organizations is rewarding but challenging – you'll need everything you learned so far to tackle this ultimate challenge. You must first understand how a complex organization works before you can undertake to change it. Your architectural thinking will help you understand organizations as complex systems. Superb communication skills help you garner support while leadership skills are needed to effect a lasting change. Lastly, your IT architect skills allow you to implement the necessary technical changes necessary for the organization to work in a different way.

Citing *The Matrix* one more time (after all, Neo is quite a change agent in a tough environment!), the exchange between *The Architect* and *The Oracle* draws the apt context:

> The Architect: You played a very dangerous game.
>
> The Oracle: Change always is.

Interestingly, in *The Matrix* the Architect is the main entity trying to prevent change. You should identify yourself with Neo instead, making sure to have an Oracle to back you up.

Not all Change is Transformation

Not every change deserves to be called *transformation*. You can change the layout of the furniture in your living room, but you

transform (or maybe convert) your house into a club, retail store, or place of worship. The Latin origin of the word *trans-form* means changing shape or structure. When we speak of IT Transformation we therefore imply not an incremental evolution, but a fundamental restructuring of the technology landscape, the organizational setup, and the culture. Basically, expect to have to turn the house upside down, cut it into pieces, and put it back together in a new shape. As an architect you are best qualified to understand how technical and organizational changes depend on each other so you can solve the Gordian knot of interdependencies.

Bursting the Boiler

A prevalent risk in corporate transformation agendas is upper management recognizing the need for change and subsequently applying pressure to the organization, e.g. to become faster, more agile, more customer-centric, etc. However, the organization, and especially middle management, is often not ready to transform and attempts to achieve the targets set by upper management within the old way of working. This can put enormous strain on the organization and is unlikely to meet the ambitions. I compare this to a steam engine, which is surpassed by a fast electric train. In an attempt to speed up, the operator may throw more coals onto the fire to increase the boiler pressure. Unfortunately, this will burst the boiler rather than beat the electric train. As an architect you have to devise a new engine that can keep up instead of simply turning up the dials.

Why me?

As an architect you may think: "Why me? Isn't this where the high-paid consultants come in?" They can certainly help, but you can't just inject change from the outside; it must come from the inside. This is one reason why I am doing transformation not as a

consultant, but as a full-time employee, even though it has some challenges (see *Fifty Shades of IT*).

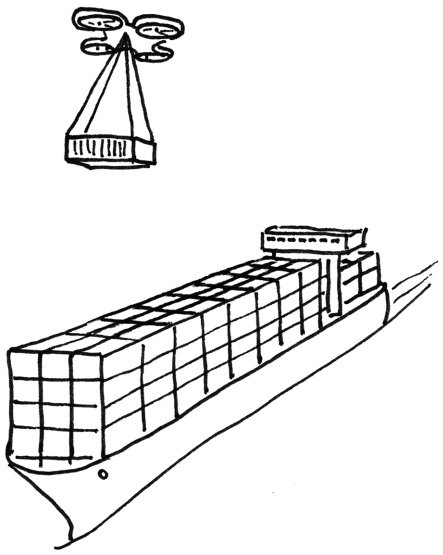
Triggering a change in technology or development approaches also requires you to take a role in changing the organization: you can't be agile or use a DevOps development style if you don't adjust the organization and its culture.

To affect lasting change in an organization you need to understand:

- That organizations *Will not change if there's no pain*.
- How to *Lead Change* by showing a better way of doing things.
- Why organizations need to think in *Economies of Speed* instead of Economies of Scale.
- Why an *Infinite Loop* is an essential part of digital organizations.
- Why excessively *Buying IT services* can be a fallacy.
- How to speed up organizations by *Spending less Time Standing in Line*.
- How you can get the organization to *Think in New Dimensions*.

Economies of Speed

Death by efficiency is slow and painful



Economies of Scale vs. Economies of Speed

Large companies looking at their digital competitors are often surprised to find out that those companies don't move 10% faster, but 10x faster. A quick example shows, that even this is still quite conservative.

A large IT organization looking to define a standard for source control invested 6 months of community work to conclude that they should be using GIT. Alas, it was considered too difficult to migrate other projects off Subversion, so both products were recommended. The preparation cycle for the global architecture steering board meeting took another month, bringing the total elapsed time to 7 months or roughly 210 days.

A modern IT organization or start-up would have spent a few minutes deciding on the product and have accounts setup, a private repository created, and the first commit made in about 10 minutes. The speed-up factor comes to 210 days * (24 hours / day) * (60 minutes / hour) / 10 minutes ≈ 30,000! If that number alone doesn't scare you, keep in mind that one organization published a paper (without selecting or implementing a product such as BitBucket, GitHub, or GitLab) and is merrily dragging their legacy along. Their "decision" is thus about as meaningful as prescribing that men should wear black shoes, but brown is also allowed for historical reasons. Meanwhile the other organization is already committing code in a live repository. If you extrapolate the traditional organization's timeline to include vendor selection, license negotiation, internal alignment, paperwork, and setting up the running service, the ratio may well end up in the hundreds of thousands. Should they be scared? Yes!

Old Economies of Scale

How can this happen? Traditional organizations pursue economies of scale, meaning they are looking to benefit from their size. Size can indeed be an advantage, as can be seen in cities: density and scale provide short transportation and communication paths, diverse labor supply, better education, and more cultural offerings. Cities grow because the socioeconomic factors scale in a superlinear

fashion (a city of double the size offers more than double the socioeconomic benefits), while increases in infrastructure costs are sublinear (you don't need twice as many roads for a city twice the size). But density and size also bring pollution, risk of epidemics, and congestion problems, which ultimately limit the size of cities. Still, cities grow larger and live longer than corporate organizations. One reason lies in the fact that organizations suffer more severely from the overhead introduced by processes and control structures that are required or perceived to be required to keep a large organization in check. Geoffrey West, past president of the Santa Fe Institute, summarized this dynamic in his fascinating video conversation *Why cities keep growing, corporations and people always die, and life gets faster*[10].

In corporations, economies of scale are generally driven by the desire for efficiency: resources such as machines and people must be used as efficiently as possible, avoiding downtimes due to idling and retooling. This efficiency is often pursued by using large batch sizes: making 10000 of the same widget in one production run costs less than making 10 different batches of 1000 each. The bigger you are, the larger batches you can make, and the more efficient you become. This view is overly simplistic, though, as it ignores the cost of storing intermediate products, for example. Worse yet, it doesn't consider revenue lost by not being able to serve an urgent customer order because you are in the midst of a large production run: the organization values *resource efficiency* over *customer efficiency*.

The manufacturing business has realized this about half a century ago, resulting in most things being manufactured in small batches or in one continuous batch of highly customized products. Think about today's cars: the number of options you can order are mind boggling, causing the traditional "batch" thinking to completely fall apart: cars are essentially batches of one. With all the thinking about "lean" and "just in time" manufacturing it's especially aston-

[10] https://www.edge.org/conversation/geoffrey_west-why-cities-keep-growing-corporations-and-people-always-die-and-life-gets

ishing that the IT industry is often still chasing efficiency instead of speed.

A software vendor once stated that "obviously the license cost per unit goes down if you buy more licenses". To me, this isn't obvious at all as there's no distribution cost per unit of software, aside from that very sales person sitting across the table from me. Whether 10,000 customers download one license or one customer buys 10,000 licenses should be the same, as long as the software vendor doesn't *Send Humans to do a Machine's Job*.

It looks like enterprise software sales still has some transformations to make. To their defense, though, one has to admit that their behavior is determined by enterprise customers still stuck in the old thought pattern: super-size it to get a better deal!

In the digital world, the limiting factor for an organization's size becomes its ability to change. While in static environments being big is an advantage thanks to economies of scale, in times of rapid change *economies of speed* win over and allow start-ups and digital native companies to disrupt much larger companies. Or as Jack Welsh famously stated: "If the rate of change on the outside exceeds the rate of change on the inside, the end is near."

Behold the Flow!

The quest for efficiency focuses on the individual production steps, looking to optimize their utilization. What's completely missing is the awareness of the production flow, i.e. the flow of a piece of work through a series of production steps. Translated into organizations, individual task optimization results in every department requiring lengthy forms to be filled out before work can begin: I have been told that some organizations require firewall changes to be

requested 10 days in advance. And all too often the customer is subsequently told that some thing or another is missing from the request form and is sent back to the beginning of the line. After all, helping the customer fill out the form would be less efficient. If that reminds you of government agencies, you may get the hint that such processes aren't designed for maximum speed and agility.

Besides the inevitable frustration with such setups, they trade off *flow efficiency* for *processing efficiency*: the work stations are nicely efficient, but the customers (or products or widgets) chase from station to station, fill out a form, pick a number, and wait. *And wait*. And wait some more just to find out they are in the wrong line or their need cannot be processed. This is dead time that isn't measured anywhere except in the customers' blood pressure. Come to think of it, in most of these places, the people going through the flow are not customers in the true sense as they don't choose to visit this process, but are forced to. That's why you are bound to experience such setups at government offices, where you could at least argue that misguided efficiency is driven by the pursuit to preserve taxpayer money. You'll also commonly find it in IT departments that exert strong *governance*.

Cost of Delay

For innovation and product development processes, this type of efficiency is pure poison. While digital companies do care about resource utilization (at Google data center utilization was a CEO-level topic), their real driver is speed: time-to-market.

Traditional organizations often don't understand or underestimate the value of speed. In a joint business-IT workshop, a business owner once described that his product carries substantial revenue opportunities. At the same time, the product owner asked for a specific feature that required significant development effort, but had value only in a later stage when the product would be rolled out in another country. I quickly concluded that deferring that specific

feature speeds up the initial launch and harvests the portrayed revenue opportunities sooner.

Flow-based thinking calls this concept the *cost of delay* (see the excellent book *The Principles of Product Development Flow*[11]), which must be added to the cost of development. Launching a promising product later means that you lose the opportunity to gain revenue during the time of delay. For products with large revenue upside, the cost of delay can be much higher than the cost of development, but it's often ignored. On top of avoiding the cost of delay, deferring a feature and launching sooner also allows you to learn from the initial launch and adjust your requirements accordingly. The initial launch may be an utter failure, causing the product to never be launched in the second country. By deferring this feature you avoided wasting time building something that would have never been used. Gathering more information allows you to *make a better decision*.

A great example of a non high-tech company that embraced economies of speed is the fashion brand *Zara*, part of the Inditex fashion empire. When the pursuit of efficiency drove most fashion retailers to outsource production to low-cost suppliers in Asia, Zara implemented a vertically integrated model and manufactured three-quarters of its clothing in Europe, which allowed it to bring new designs into stores in a matter of weeks as opposed to the industry average of 3 to 6 months. In the fast-moving fashion retail industry, speed is such a significant advantage that it made Inditex' founder the second richest man on the planet.

The Value and Cost of Predictability

How come intelligent people ignore basic economic arguments such as calculating the cost of delay? They are working in a system that favors predictability over speed. Adding a feature later, or,

[11] Reinertsen: The Principles of Product Development Flow: Second Generation Lean Product Development, Celeritas Publishing, 2009

worse yet, deciding later whether to add it or not may require going through lengthy budget approval processes. Those processes exist because the people who control the budget value predictability over agility. Predictability makes their lives easier because they plan the budget for the next 12-24 months, and sometimes for good reasons: they don't want to disappoint shareholders with run-away costs that unexpectedly reduce the company profit. As these teams manage cost, not opportunity, they don't benefit from an early product launch.

Chasing predictability causes another well-known phenomenon: *sandbagging*. Project and budget plans sandbag by overestimating timelines or cost in order to more easily achieve their target. Keep in mind that estimates aren't single numbers, but probability distributions: a project may have a 50 percent chance of being done in four weeks' time. If "you are lucky and all goes well" it may be done in 3 weeks, but with only a 20% likelihood. Sandbaggers pick a number far off on the other end of the probability spectrum and would estimate eight weeks for the project, giving them a greater than 95% chance of meeting the target. Worse yet, if the project happens to be done in four weeks, the sandbaggers idle for another four weeks before release to avoid having their time or budget estimates cut the next time. If a deliverable depends on a series of activities, sandbagging compounds and can extend the time to delivery enormously.

The Value and Cost of Avoiding Duplication

On the list of inefficiencies, *duplication* of work must be high up: what could be more inefficient than doing the same thing twice? That's sound reasoning, but one must also consider that avoiding duplication doesn't come for free: you have to actively de-duplicate. The primary cost involved in de-duplication is coordination: to avoid duplication you first need to detect it. In a large code base this can be done efficiently through code search. In a large organization, it usually requires "alignment" meetings, i.e. synchronization

points, high up in the hierarchy, which we know to *Not Scale* in both computer systems and organizations.

Evolving a widely reused resource also requires coordination because changes must be compatible with existing all systems or users. Such coordination can slow down innovation. On the flip side, modern development tools, such as automated testing, can reduce the traditional dangers of duplication. Some digital companies have even begun to explicitly favor duplication because their business environment rewards economies of speed.

How to make the Switch?

Changing from efficiency-based thinking to speed-based thinking can be difficult for organizations: after all, it's less efficient! In most people's minds being less efficient translates into wasting money. On top of that, people being idle is more visible than the damage done by missed market opportunities.

Usually, this change in attitude happens only when IT is seen as driving business opportunity instead of being a cost center. While corporate IT is stuck in a cycle of cutting cost and increasing efficiency, economies of scale will prevail, which gives the digital giants an ever-bigger lead over traditional companies that dream of becoming digital, but cannot shed their old habits.