

Top-Down versus Bottom-Up Approaches to Architecting

White Paper Resulting from Architecture Forum Meeting

April 27-28, 2017, Roche Diabetes Care, Indianapolis, Indiana, USA

Edited by:

Teun Hendriks, TNO-ESI

Gerrit Muller, USN-NISE. TNO-ESI

Eirik Hole, Stevens Institute of Technology

Input was provided by the following participants in the Architecture Forum:

Name	Organization
Leandre Adifon	Ingersoll Rand
Charity Barlow	Roche Diabetes Care
Tim Beck	Roche Diabetes Care
Joe Forler	Roche Diabetes Care
Teun Hendriks	TNO-ESI
Eirik Hole	Stevens Institute of Technology
Kees Kooijman	Sioux Embedded Systems
Wouter Leibbrandt	TNO-ESI

Name	Organization
Hugo van Leeuwen	Thermo Fischer Scientific
Thomas Meenken	Roche Diabetes Care
Peter Nacken	OCé
Jurjen Nicolai	Bosch
Bob Reinke	Roche Diabetes Care
Rolf Siegers	Raytheon
Martin Simons	Daimler
Paul Zenden	Sioux Embedded Systems

Published, September, 2017

1. Introduction

The trends of Systems of Systems (SoS) and Internet of Things (IoT) trigger (unforeseen) emerging behaviour and properties. These trends trigger a discussion on Top-Down versus Bottom-Up approaches to architecting. One of the questions is how much we architect Top-Down (from needs to realization) or Bottom-Up (from potential realizations discover the customer value). We may relate Top-Down to "directed" and Bottom-Up with "emerging".

The members of the architecting forum discussed this topic, using the following questions:

- When and why do we choose to work Top-Down versus Bottom-Up?
- How do we balance between Top-Down and Bottom-Up?
- How do we ensure that Bottom-Up results in viable solutions?
- How can Top-Down benefit from (technical) opportunities and strengths?
- How does scope impact the balance between Top-Down and Bottom-Up?
- Who decides on the approach to use?

2. System architecting: an inherently messy, random walk process?

A forum member presented his view on the process of systems architecting. Systems architecting involves making explicit business, market, and technology drivers, and considering all these to work towards system concepts and designs, and ultimately system realizations (see Figure 1).

The provocative statement put forward was that the systems architecting process is inherently a messy, random walk process. It *needs to be messy* so that the "magic" can happen. While systems architects may like to be seen as rational professionals, working systematically and intentionally Top-Down, often they act quite irrational, driven by intuition, based on available assets and know-how, in a Bottom-Up fashion.

While messy, the presented architecting process was structured to an extent with a "fast failing forward" agile approach to development. Uncertainties and knowledge gaps are tackled by building and improving prototypes until they work; only then system specifications are created. Nonetheless an essential part of this "messy" architecting

process is sowing seeds; and observe what grows (develops) in the context of a guiding structure¹.

After the fact, this messy process is then covered-up by a post-mortem rationalization which makes it seem like a systematic way of working: as if everything was fully “under control” and conducted Top-Down. This isn’t a new insight; Parnas (1986) describes the same phenomena for software design.

Architecting

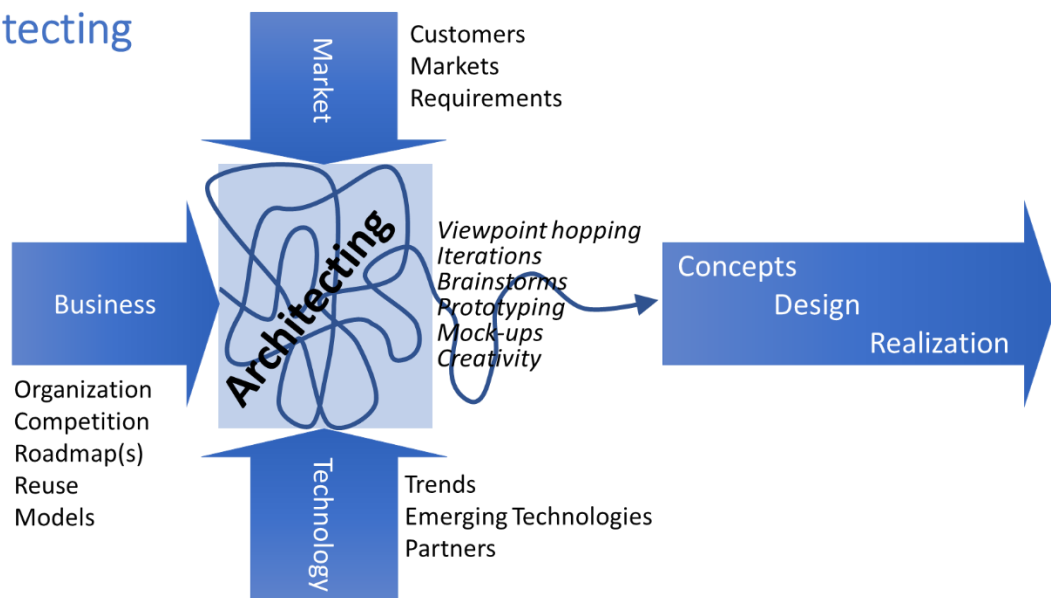


Figure 1: system architecting: a messy, random walk, process to scope and architect system solutions?

This provocative statement triggered several reactions:

- Such a messy process totally depends on the persons involved and is not scalable and not repeatable. More complex systems require more people and thus more communication, structure, and layers in the organization.
- Maturing organizations will not tolerate such a process due to risk aversion. Typically, they will force the architecture process to be more Top-Down. Such risk

¹ In analogy of a garden where, given a structure of the lanes and footpaths, seeds are sown to grow and blossom in flower beds.

aversion can come from multiple different directions (e.g., system size, organization size, regulation, IT) and force an organization to be "stiffer". This can also stifle some of the messy magic.

- Even in an agile process, some upfront (Top-Down) architecture is necessary as context for (virtual) prototyping to do risk assessment and risk reduction.

The question raised was whether something created Bottom-Up can be called an architecture at all? Is Bottom-Up architecting irrational and Top-Down architecting rational? The agile community would certainly disagree; they assert that the best architectures are created emergently by empowered teams.

Nonetheless, some projects and systems are simply too big to fail (i.e. to be developed in a "fast failing forward" fashion), and thus need a level of Top-Down architecting. A Top-Down only approach (i.e. Big Architecture Up Front) is also not the solution, yet there is value in iteration. All real developments involve both Top-Down and Bottom-Up architecting.

So, when does Top-Down architecting make sense; when does Bottom-Up? How to mix and match approaches, and when to switch? And, what is Top-Down and Bottom-Up architecting?

3. Dimensions of Top-Down versus Bottom-Up architecting

Top-Down architecting is typically associated with creating a skeleton and a first structure as proof point for a decomposition and to achieve separation of concerns. The distinction between Top-Down versus Bottom-Up architecting may take on various dimensions (see Table 1) as relevant to the scope of a product development. Top-Down versus bottom architecting can be also viewed in terms of market dimensions, green field versus legacy accommodating product-development, product versus portfolio development, or outside-in versus inside-out.

Table 1: Dimensions of Top-Down versus Bottom-Up Architecting

Top-Down versus Bottom-Up Architecting dimensions	
Top-Down	Bottom-Up
Deductive (generic \Rightarrow specific)	Inductive (specific \Rightarrow generic)
Directive (prescribed structure)	Emergent (structure emerges)
Holistic	Opportunistic (fragmented)
Abstract to concrete	Concrete to abstract
Market pull	Technology push
Outside in	Inside out
Green field/unconstrained	Legacy accommodating
Explicit design	Emergent design

New opportunities tend to arise Bottom-Up (e.g. a new technology or new use of technology) but their application may cause unexpected consequences in a system. These need to be analysed and captured Top-Down. While individual products may be architected Top-Down, product portfolios are typically managed Bottom-Up.

Bottom-Up architectural decisions sometimes tend to be made relatively mindlessly (e.g., it always worked that way and never caused problems). Yet, sometimes these fail because the underlying assumptions do not apply in a new market. On the other hand, Top-Down architectural decisions may not have the real-world proof that they work either. Hence, cross-validation of decisions is essential.

Principle 19.1: Top-Down decisions have to be validated Bottom-Up, Bottom-Up decisions have to be validated Top-Down.

Architecture involvement early in the business process is not always a given. At times, the business side already makes Top-Down decisions without technical input. Decisions and system concepts are then pushed down to R&D with milestone dates already defined and political pressure exerted to make it happen.

To avoid such disconnect, i.e. to get Top-Down architecture involvement early on, it is helpful to have a management champion. Training the business side about the necessity of

Top-Down technical involvement doesn't really work. Instead, architects need to earn their trust by asking the pertinent questions to move marketing in the right direction. Experience is that it takes years of patient and constructive collaboration to grow acceptance of the value of Top-Down architecture, and value of the contribution of architects. Further acceptance and alignment between business and R&D can be achieved through making the core values of an organization explicit in the design decision criteria.

A forum member reported success with incorporating a Top-Down architecture milestone early in their new product development stage-gate process (see Figure 2). In gate 1 an initial business hypothesis is defined. At gate 2 with the business hypothesis now framed and validated, and requirements plus business case plus risks well-defined, also a defined top-level architecture was required.

After gate 2, with the top-level architecture in place, development teams can proceed Bottom-Up (but the architects still have responsibility to monitor developments). Architectures created in this way were found to be more mature and more stable. Subsequent Top-Down phases became faster as a result.

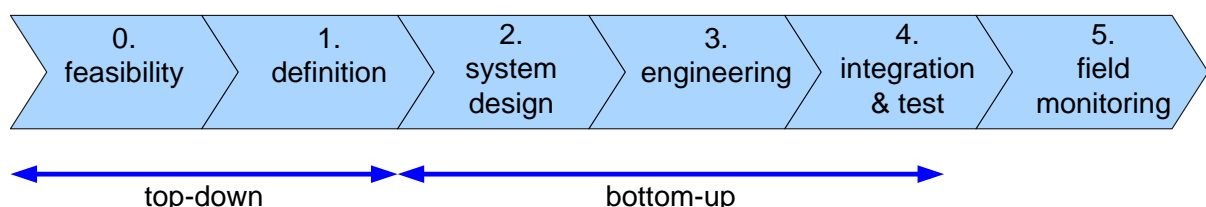


Figure 2: NPD stage gate process with requirement for top-level architecture at gate 2.

On the down side, radical technology changes took much longer to pass through the initial phases. Even though these were triggered Bottom-Up, they also had to go through the Top-Down process. Also, they were looked at from an overall portfolio architectural perspective to ensure that they are not pursuing inconsistent directions. Architects were charged with maintaining the portfolio architecture as projects proceeded.

All courses at TNO-ESI emphasize the need to work from both directions, as visualized by Figure 3. Most dimensions in Table 1, which appear as opposites, are often desirable at the same time. A way to achieve this is by iterating fast Top-Down as well as Bottom-Up.

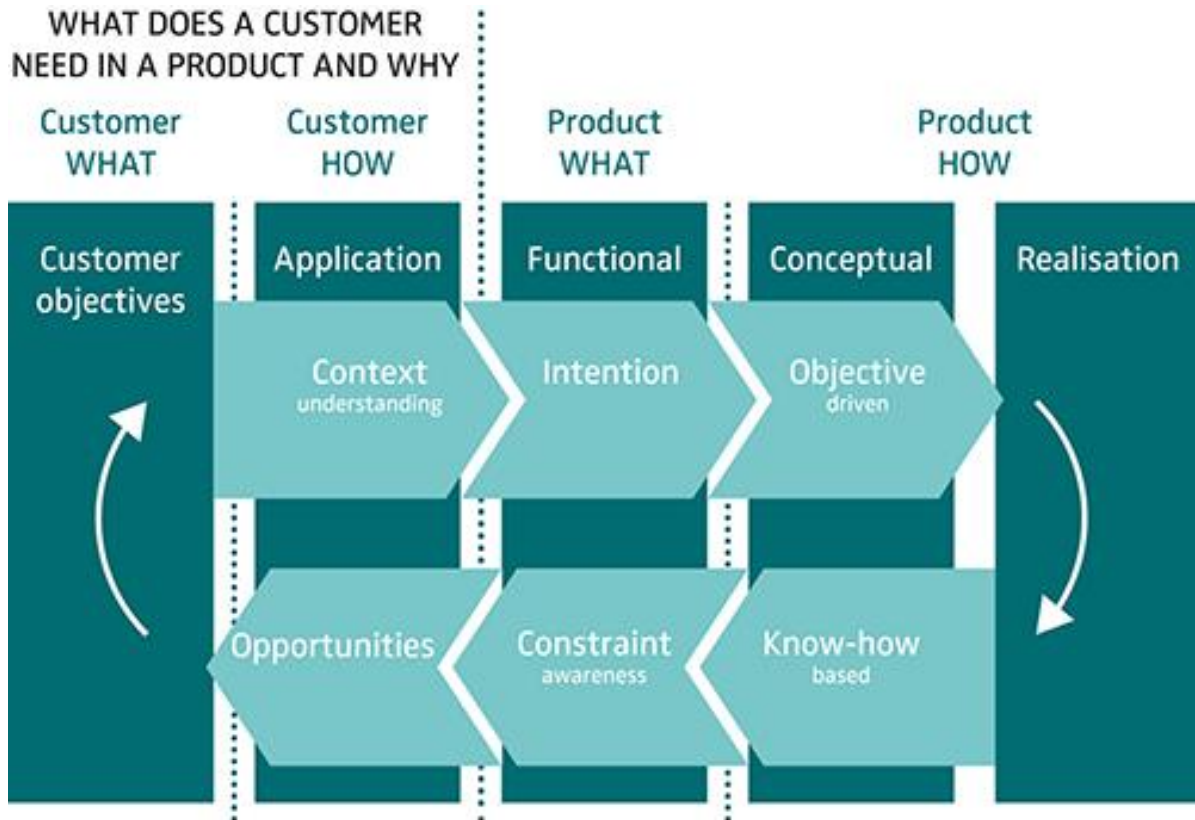


Figure 3: Top-Down (from Customer objectives to Realization) and Bottom-Up (from Realization to Customer Objectives) via fast iteration.

4. When is which direction useful and successful

The choice of Top-Down vs. Bottom-Up architecting is not one-size-fits all. If issues are big enough as a project moves forward, these must be brought back to Top-Down consideration. Sometimes such an approach can cause delays, which may be politically difficult. Thus, architects often start Top-Down, but as schedule and other business demands increase often a switch to Bottom-Up occurs to show "real" progress.

The forum discussed in general in what situations Bottom-Up architecting would be successful, and what situations Top-Down architecting. Table 2 presents some success examples for either architecting direction.

Table 2: Success examples of Top-Down versus Bottom-Up architecting

Top-Down versus Bottom-Up Architecting: success examples	
Top-Down	Bottom-Up
Printed Circuit Board development from a library of functions (existing and foreseen) which comply to architectural guidance, and are testable on function level	LCM of obsolete electronics - as late as possible - but packaging "final clean-up" - minimal impact on existing architecture
Product/capability generation update e.g. replacement / next generation of test set, scrutinizing external interfaces to support/discontinue	Combining existing services into new products - Google maps - Craig's list
Introducing a modular approach into a product to achieve better reliability.	Integration of business acquisitions
Big multiple systems areas (e.g. a naval destroyer)	Internet (enabled by TCP/IP and HTTP)
High-volume products (e.g. glucose test strips), they may be simple but need to be extremely accurate, and produced in extreme quantities	Usage of System-on-Chip technology (Bottom-Up exploration of new technology into a product line)
PDF format (Top-Down redesign of postscript)	HTML5 for UI design
Systems-of-systems/interconnected systems	New technology introduction
Adaptation existing architecture to new markets (e.g. roadwork equipment adapted to farming)	Single technology core
Long lead-time allowed, e.g., in top-of-the line passenger vehicle development	New legal requirements

In a second seed presentation, a case involving the integration of a business acquisition into a product line was described. When bringing companies with independent products and product development cultures together it is important to establish mutual trust and a common language and terminology.

The respective architects who have evolved in an independent environment, now have to adapt to this new situation. This needs a gentle process to integrate, with give and take at times. Similar terms may have a different meaning in the respective companies. It is then dangerous to rely on one-liners: the same terms may have e.g. in one company an internal-

technical connotation; in the other company an external, user-oriented connotation. When integrating different products, the integration process thus needs to start Bottom-Up: to create shared understanding, agree on differences, and agree on terminology.

Only with this shared language present, integration of product architectures can take place. The shared language enables a mutually understood reverse engineering (Bottom-Up creation) of existing product architectures. Then, in a Top-Down architecting effort, the individual product architectures were successfully merged into a common architecture observing the key architectural drivers.

In this case of a business acquisition and product architecture integration/evolution, a three-step process was followed:

1. Define a common language
2. Bottom-Up architecting: reverse engineer the individual product architectures using the common language
3. Top-Down architecting: merge the individual product architectures into a joint common architecture observing key architectural drivers

The forum discussed what in general success factors for Top-Down versus Bottom-Up architecting could be defined. Table 3 lists a number of success factors.

Table 3: Success factors Top-Down and Bottom-Up architecting

Top-Down versus Bottom-Up Architecting: success factors	
Top-Down	Bottom-Up
Generic framework with room for Bottom-Up evolution	Component obsolescence (implies a deep understanding of components, data driven)
Long lead time plus technology testbed	Short lead time, with contained change
Identified market opportunity	Architecture revision
Too big to fail project with high impact of decisions	Technology as primary driver (also in existing product)
Top-Down redesign with lessons-learned	Alignment of communication and culture (in case of a merger or acquisition)
Product adaptation to new markets	Long-lasting principles (e.g. internet enabled explosion of apps and functions)

A key element in the Top-Down success factors was the availability of domain knowledge.

Domain knowledge is often not explicit initially, especially not in start-up situations, or when introducing new technologies, or when entering new markets. Yet, a key understanding of a domain and its driving concepts enables long-lasting architectures to be designed.

On the customer and market side, Design Thinking and Story-Telling offer new ways to capture the essence of a domain in a condensed, nicely illustrated, way. Architects may consider getting to know customers themselves, instead of relying on product managers. Doing an internship at a customer location can make them see the obvious that nobody bothers to write down.

On the product side, a good way to make domain know-how explicit is the creation of a conceptual, or domain architecture. As abstraction of the physical architecture, such a conceptual architecture can guide updates of the physical architecture and product realizations over technology and feature updates. Without it, the physical architecture will become polluted over time.

Principle 19.2: Create understanding of a domain Bottom-Up; and use, check, refine and abstract this domain understanding Top-Down; keep iterating for mutual validation and inspiration.

Availability of domain knowledge is also key in architecting for change. While change can't be planned for, question is whether it should be prepared for. In certain fields, such as Internet-of-Things and Cloud Services, architecting for change is essential. Services change, APIs change.

5. Where do Top-Down and Bottom-Up meet, and how to manage?

The forum concurred that no systematic, Top-Down only, recipe for architectural success exists. So how to mix and match Top-Down and Bottom-Up architecting? Where do they meet, and how to manage?

Figure 3 shows the process of interaction between Top-Down and Bottom-Up architecting. At the interface, questions are raised and answers given (in both directions). Before such

communication can start however, it is necessary to create a common language first. The ‘normal’, untrained, way of working of the mind is Bottom-Up, Top-Down requires some training.

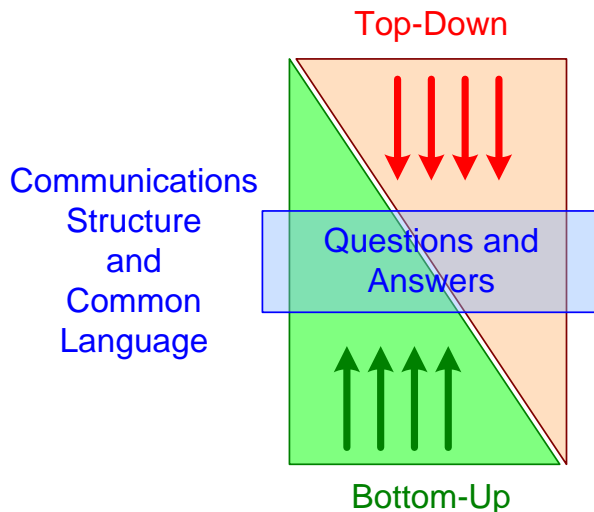


Figure 3: The interaction between Top-Down and Bottom-Up architecting

At first, Bottom-Up may dominate to fill in domain uncertainties, and explore issues. In later stages, Top-Down architecting may take most of the architecting effort.

Often, the interface is not immediate. A gap may exist, prohibiting to connect Top-Down with Bottom-Up directly. How to recognize such a gap and how to close such a gap? Two example situations are as follows:

- While working Top-Down, you start feeling insecure given an observed lack of knowledge with an assessed high level of risk. Then it is time to switch to Bottom-Up to assess these risks and obtain required knowledge.
- You are working Bottom-Up, and you realize that your topic will fit in a bigger context. Then it is time to incorporate a Top-Down architecture effort.

Architectural gaps may be caused by a lack of a shared picture, a lack of a common language, or a lack of sufficient domain or technology knowledge. To close such gaps, always two-way communication is needed. Both sides should be open to listen, and interact, with intention to seek alignment.

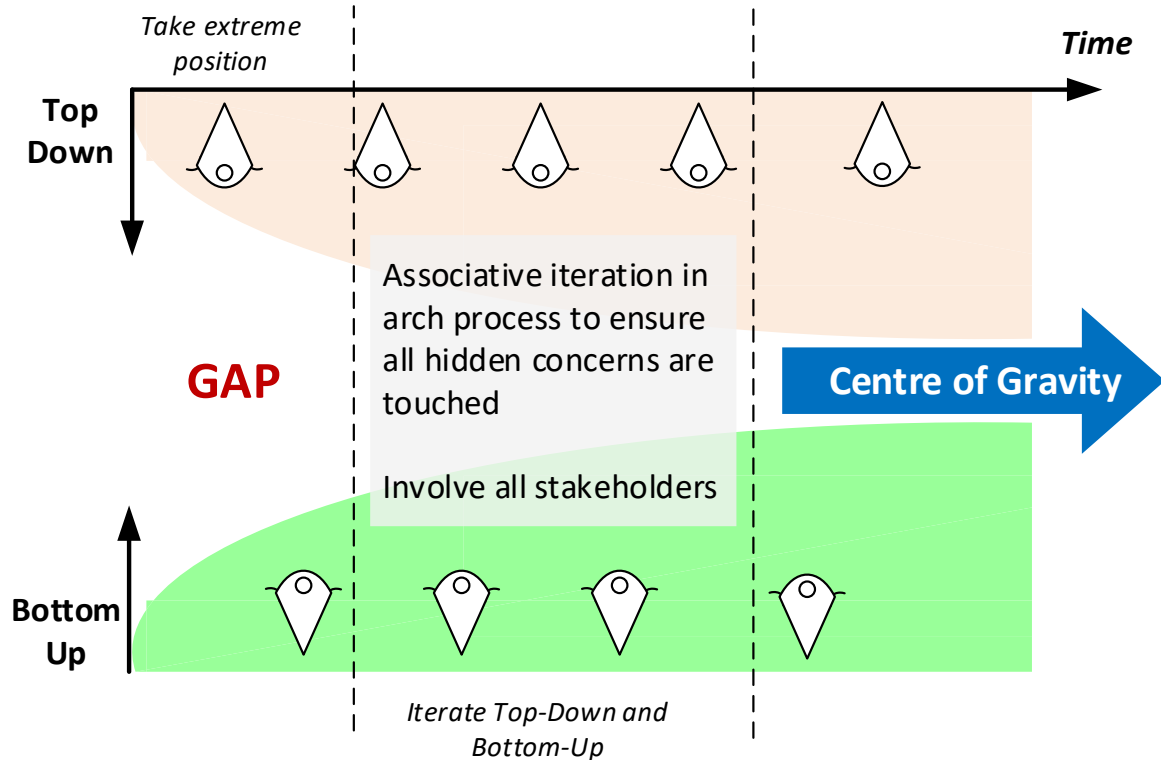


Figure 4: Process to connect Top-Down to Bottom-Up

A suitable way of working to connect Top-Down and Bottom-Up, is to start from one side (Bottom-Up or Top-Down) and then switch to the other side (see Figure 4). Keep iterating to ensure that all uncertainties are touched, and narrow these down to arrive at the 'centre of gravity' i.e. key problems to address. This centre of gravity then should drive the main effort.

Thus, to make Top-Down and Bottom-Up meet, establish two-way communication; and switch positions regularly. Both directions yield valuable insights, the crux is to find ways to connect them. Suitable methods to arrange/facilitate two-way communication between Top-Down and Bottom-Up are e.g. simulation, or virtual prototyping models. Not all questions may need the same level of detail, thus such models can be very coarse or refined, depending on the topic or architectural view addressed.

6. Principles and guidance for orchestrating Top-Down and Bottom-Up

How to orchestrate this process for closing the gap between Top-Down and Bottom-Up? Here perhaps architects can learn from other disciplines dealing with uncertainty and dynamic situations. On such discipline is Warfighting, a very old discipline with lots of experience. The U.S. Marine Corps has developed methods and doctrines to handle such situations (see Table 4 for a summary, and the appendix for more details and applicability).

Key objective of these doctrines is to ensure that military actions are aligned with mission and objectives, yet that these missions still benefit from soldiers at the front lines having, and acting on, the most up-to-date information as it unfolds. Architects similarly need to ensure architecture alignment, while also needing to make use of the most up-to-date technology and market know-how of subject experts.

Table 4: Summary of the “Reinke Principles”: Ten Principles from Warfighting, based on Reinertsen (2009) and U. S. Marine Corps Staff (2012)

1	Boundaries	6	Authoritative but not prescriptive
2	Commander’s Intent	7	Planning is invaluable, Plans are Useless
3	Implicit Communication	8	Centre of Gravity
4	Early Contact	9	Layered Control
5	Large versus Small Fires	10	Main Effort

The forum discussed these principles and their applicability on systems architecting. The appendix provides an overview of the principles and their applicability. Here the main findings are presented.

Setting and validating boundaries

Architects shall aim to set clear boundaries. This limits the amount of “random walking” to be done. Such boundaries still need to be validated Bottom-Up: do the boundaries stand when confronted with the details, are they feasible?

The level of boundary setting can lead to tension: dictating vs. designers' freedom. The level of boundary setting needs to reflect the capability of the engineering staff. Less experienced staff need more detailed boundaries – more experienced people should be trusted to do part of the thinking. The latter may also lead to innovation – the learning organization.

Commander's Intent and (Implicit) Communication

Communication is key to architecting. Architects need to understand the psychology of communication. It is essential to communicate the intent of the architecture, (also) in a more informal, less technical way. Hence, architects must be able to grasp and communicate the essence and intent of the architecture in easy to understand stories and pictures.

To achieve organisation buy-in, architects need to inform and enrol people by spreading such informal communication (as if it were *gossip*); and in larger organizations spread these stories and pictures around too as *myths* (Bar-Niv, and Shenhav 2016). A good myth is simple, abstract, yet close to reality, and provides rationale. Myths can cover blueprints, abstractions, architectural decisions and guidelines, or the architectural vision. A good example of a myth is the simple picture and name (“Skyhook”) used by the lead architect of ASML to communicate the essence of a new way of framing the lithography system, (Bonnema 2014, p.4, figure 6).

Shared experience or training in an architecture doctrine provides for shared thinking patterns (“Implicit communication”). Architects having such shared experience or trained upfront in an architecture doctrine will take decisions more coherently, and achieve better result under time pressure; they need much less explicit communication and alignment.

Making Early Contact

Making early contact, i.e. grounding the architectural work early on with Bottom-Up insights, helps to prevent the “big architecture upfront” pitfall. Architects thinking too much upfront in their own confined space, will not be effective. Contact with the “enemy”, i.e. “the real world” (not your organization) grounds the work, reduces uncertainty, and provides direction.

Determining the Centre of Gravity

Making early contact is part of the strategy to determine the centre of gravity, i.e. the key architectural issues to address. Two-way communication between Top-Down and Bottom-Up (see section 5) shall ensure that all risks and uncertainties are touched. Through iteration and viewpoint hopping these then can be narrowed down to arrive at the key architectural issues to address in combination with a key overview on available assets.

Main Effort

Architects shall make clear what the main effort is; the main activities and key architectural issues to be worked on. These main activities may shift over time due to viewpoint hopping and the phase with respect to the product development life cycle. This is logical, but needs to be made clear for the organization.

When to stop architecting and delegate

(Large vs Small Fires; Authoritative versus Prescriptive)

When do you stop architecting and delegate? This moment arrives, when, as an architect, you do not care anymore what type of solutions designers would choose, e.g. because it may not be needed to meet architectural requirements. This is a case of “large fires versus small fires”.

Even when a requirement has architectural scope, it may be better for acceptance to delegate the creation of a solution to representatives of affected engineering teams (“Authoritative, not Prescriptive”). For instance, the creation of engineering directives for restart/recovery, diagnostics, etc. may see better acceptance if implementation teams are (part-time) involved to define together a single, common solution that works for all of them.

7. Conclusions

The forum came to the conclusion that guidance can be given for the process of systems architecting. While no simple recipe exists, systems architecting is not by necessity a “messy, random walk” process either.

Principles provide guidance to the architect such that the way-of-working can be “associative” rather than “random”. Architects can provide scope by setting clear boundaries, and can enrol and inform people by communicating intent and spreading myths and gossips. They may consider using different approaches for large versus small fires. Iterating between Top-Down and Bottom-Up and viewpoint hopping addresses risks and uncertainties to close the gap between technology and market drivers. In doing so, architects may sometimes consider acting authoritatively, and sometimes prescriptively (see Figure 5).

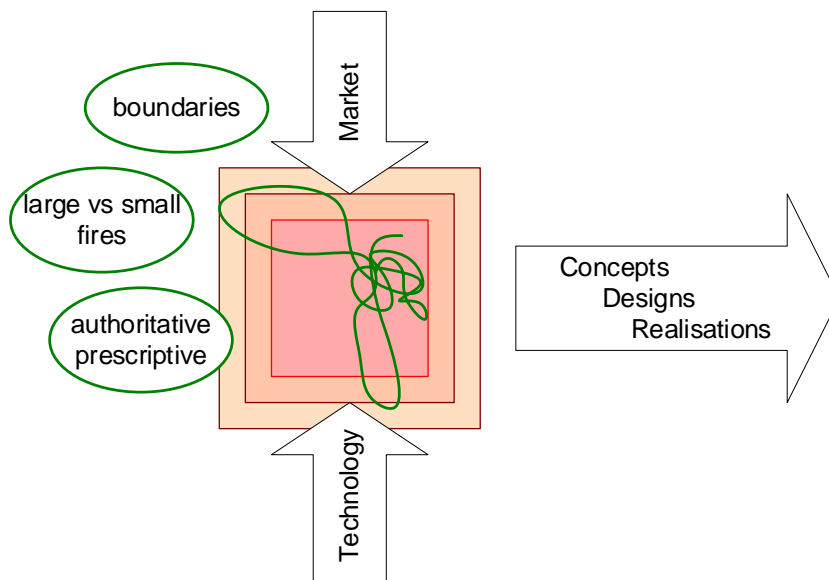


Figure 5: System architecting: a guided "associative walk" process to scope and architect system solutions

Revisiting the questions of the introduction, the forum summarized its findings as follows.

When and why do we choose to work Top-Down versus Bottom-Up?

Top-Down versus Bottom-Up has multiple dimensions. Only seldom will a solely Top-Down or a solely Bottom-Up approach be effective. A more Top-Down architecting approach works well when ample domain and technology know-how is available, e.g. in redesigns, or in case of long lead-times. A Top-Down approach is simply needed in “too big to fail” systems and projects. A more Bottom-Up effort may be needed to gain an understanding of new domains and technologies before they can be integrated. In such cases making “early contact” is invaluable to reduce uncertainty quickly.

To scope and direct New Product Development (NPD), it is recommended to provide a top-level architecture early in the NPD process, together with e.g. a validated business case and top-level requirements. A top-level architecture should at least contain a skeleton structure with a top-level decomposition that achieves separation of concerns.

How do we balance between Top-Down and Bottom-Up?

The balance between Top-Down and Bottom-Up depends in large part on whether the necessary domain and technology know-how is available or not. More important is to establish a dialog between Top-Down and Bottom-Up, such that risks and uncertainties can be identified and narrowed down to the main architectural issues to address. Through this dialog, gaps in know-how will become quickly visible, and will direct the balance in effort. Target is to reach mutual understanding on the main architectural issues. For topics with system-wide impact, (the large fires) Top-Down common approaches and standards should be set, whereas smaller issues are better delegated.

How do we ensure that Bottom-Up results in viable solutions?

Bottom-Up results need to be validated Top-Down and be shown to fit in an overall architecture. The overall architecture should consider sufficient views (order 8-10) to perform this fitness check. Viewpoint hopping and iteration between Top-Down and Bottom-Up are crucial as Top-Down results equally need validation Bottom-Up for feasibility.

How can Top-Down benefit from (technical) opportunities and strengths?

The dialog between Top-Down and Bottom-Up must be a two-way communication with questions and answers both ways. Architects should be conscious how much “Designers Freedom” to give and how much to stay in control. In fast moving technology areas, it may pay off to take an Authoritative rather than Prescriptive approach towards designers to take benefit of the latest innovations and strengths.

How does scope impact the balance between Top-Down and Bottom-Up?

Too big to fail systems and projects simply require a large Top-Down effort to divide and conquer, by providing the necessary structure and directives. Nonetheless, experience levels of personnel, and experience or training with a shared architecture doctrine influence whether all this effort needs to be expended centrally and explicitly coordinated, or whether some parts can be delegated and handled decentralized based on confidence in implicit communication and stated intent.

Who decides on the approach to use?

Ideally, a (lead) architect should orchestrate the approach, and communicate the main effort and intent. In many organizations however, the business side tends to make Top-Down decisions without technical input. In such organizations, architects first need to earn trust by asking the pertinent questions to move the business in the right direction. Patience is needed, as experience shows that it takes years of constructive collaboration to grow acceptance of the value of architecting.

Literature

- Bar-Niv, A., Shenhav, A., Can't Find Superheroes to Help You Out of a Crisis? How About Some Architecture and Lots of Superglue? (were gossip and myths are talked about), SEI, 2016, <http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=454541> (incl. PDF link)
- Bonnema, G. M., Communication in Multidisciplinary Systems Architecting, Procedia CIRP, Volume 21, 2014, Pages 27-33, ISSN 2212-8271, <http://www.sciencedirect.com/science/article/pii/S2212827114007549>
- D.L. Parnas and P.C. Clements. A rational design process: How and why to fake it. IEEE Transactions on Software Engineering, SE-12., No. 2:251- 257, February 1986
- Reinertsen, D. G., The Principles of Product Development Flow: Second Generation Lean Product Development, Celeritas Publishing, Redondo Beach CA, 2009. ISBN 978-1-935401-00-1.
- U. S. Marine Corps Staff, Warfighting, Renaissance Classics, 2012. ISBN 978-1-48123-47-02.

Appendix: The “Reinke Principles”: Ten Principles from Warfighting

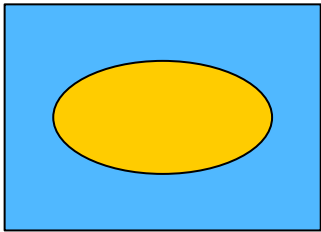
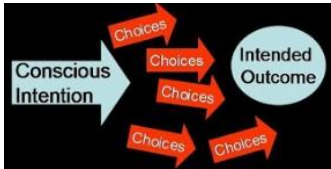
Warfighting is a very old discipline with lots of experience to handle uncertainty, incomplete information and changing dynamics. Architects can learn from such other disciplines how to cope with/manage inherent complexity of product development.





In the U.S Marine Corps, warfighting methods and doctrines have been developed and documented to ensure actions are aligned, yet still benefit from the fact that soldiers at the front lines have the most information.




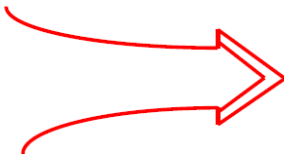
As seed presentation for inspiration for the Forum, Bob Reinke distilled ten principles from the U.S Marine Corps doctrines from two books (U.S. Marine Corp: Warfighting, 2012, and Reinertsen: The Principles of Product Development Flow, 2009).

These “Reinke principles” are reproduced in the table below (Table 5).

Table 5: The “Reinke Principles”: Ten Principles from Warfighting

1		Boundaries Establish clear roles and boundaries to avoid gaps and overlaps. (Reinertsen, pg. 253)
2		Mission and Commander’s Intent Specify the end state, its purpose, and the minimum possible constraints (Reinertsen, pg. 252). “We seek unity not principally through imposed control, but through <i>harmonious initiative</i> ...we achieve this...in large part through the use of commander’s intent.” (Warfighting, pg. 57)

3		<p>Implicit Communication</p> <p>“...exploit the human ability to communicate implicitly through mutual understanding...or even anticipating one another’s thoughts...is a faster and more effective way to communicate than through the use of detailed, explicit instructions.”</p> <p>(Warfighting, pg. 51)</p>
4		<p>Early contact</p> <p>“Contact with the enemy resolves a great deal of uncertainty...”</p> <p>(Reinertsen, pg. 260)</p>
5		<p>Large vs. Small Fires</p> <p>Centralize control for problems that are infrequent, large, or that have significant economies of scale. Example: fighting large fires.</p> <p>(Reinertsen, pg. 253)</p> <p>Decentralize control for problems and opportunities that age poorly (e.g., fire extinguishers everywhere to stop small fires from growing). Must decentralize responsibility AND authority AND resources necessary.</p> <p>(Reinertsen, pp. 246-247)</p>
6		<p>Authoritative but not prescriptive</p> <p>“Our doctrine does not consist of procedures to be applied in specific situations so much as it sets forth general guidance that requires judgment in application.”</p> <p>(Warfighting, pg. 35)</p>

7		<p>“Planning is invaluable, plans are useless”</p> <p>Plans are only as good as the information available when the plan was formulated. Plans are for alignment, not compliance. (Reinertsen, pg. 246).</p>
8		<p>Centre of Gravity</p> <p>Essential to manoeuvre warfare is understanding both your and your enemy’s centres of gravity, defined as key factors in your/your enemy’s strength. (Warfighting, pg. 28)</p>
9		<p>Layered Control</p> <p>Adapt the control approach to emerging information about the problem. (Reinertsen, pp. 248-249)</p>
10		<p>Main Effort</p> <p>Identify a main effort and subordinate other activities to it, but be ready to shift the main effort based on new information. (Reinertsen, pg. 254)</p>

Relevance for system architecting

The forum found these ten warfighting principles from the U.S. Marine Corps are also relevant for architecting. A summary of the discussion is as follows.

- **Boundaries:** Establish boundaries needs to be done centrally.
- **Commander’s intent:** Everybody needs to understand the mission and the business intent (the why, not the what / how).
- **Implicit communication:** Shared experience, or shared training enable people to think the same way, and make consistent decisions when confronted with the same information. For systems engineering, this means that architects trained upfront in

the architecture doctrine may need less explicit communication and coordination, which is beneficial in time-pressure situations.

- **Early contact:** Early grounding of architectural work in the “real” world is a fast way to reduce uncertainty. This is the value of Bottom-Up architecting, and may be achieved through early contact with customers, product management, contact with implementation layer.
- **Large versus small fires:** Centralize solutions for big (infrequent, large) problems, delegate solutions for small problems (cf. centralized fighting stations for big fires vs. fire extinguishers placed everywhere for small fires).
- **Be authoritative but not prescriptive:** Prescribe what and why, but not how, such that innovation can take place based on “front-line” information.
- **Planning is invaluable, plans are useless:** Plans are only as good as the input they are based on. Plans are for alignment, not compliance. When a plan is not OK anymore, you need a new one.
- **Centre of gravity:** For systems architects: find out what the essence of the architectural problem is and address only that.
- **Layered control:** Do triage: determine what architecture problems to solve, and which issues to accept.
- **Main effort:** Make clear what the main effort is and align work towards the main effort. In reality, the main effort may change; then also realign the work.