Continuous Architecture Framework

Table of Contents

Chapter 1. Overview

By Frédéric Lé <fle@agileddd.com> and Jean-Pierre Le Cam <jean-pierre.le-cam@wanadoo.fr>

The Continuous Architecture Framework is for enterprises wishing to architect their enterprise for digital. Business leaders have long taken responsibility for creating new business models and new operating models. Too often technology is casted as an after-thought, an enabler which is not seen as a source of innovation. We argue that in a digital world, business leaders can no longer delegate technology decisions.

The old fashioned way of aligning business and IT tends to neglect the transformative potential of digital technologies. Even if it does, the waterfall nature of such business/technology interlock does not take into account the need to continuously adapt to an evolving ecosystem characterized by new competition and changing customer expectations.

The figure Dual Business/Technology Transformation advocates a continuous business/technology change journey that is led by leaders who deeply understand their business and the transformative power of digital technologies:

- Deliver a superior experience to clients and employees, and
- Provide a sustainable competitive advantage

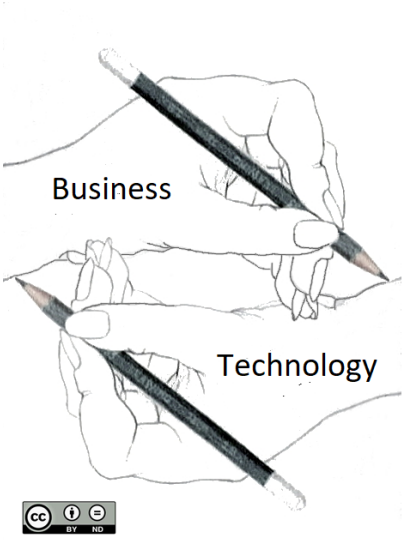


Figure 1. Dual Business/Technology Transformation

Chapter 2. Navigating the Continuous Architecture Framework

By Frédéric Lé <fle@agileddd.com> and Jean-Pierre Le Cam <jean-pierre.le-cam@wanadoo.fr>

The Navigator table contains hyperlinks to chapters and sections of the Continuous Architecture Framework.

Table 1. Navigator

	Business Unit	**Value Area**	**Product Family**
Agile Strategy	• [a]	• [b]	• [c]
Experience	• [a]	• Jobs To Be Done Analysis	• [c]
Product	• [a]	• [b]	• [c]
Operations	• [a1] • [a2]	• [b]	• [c]
Software	• [a]	• [b]	• [c]
Platform	• [a]	• [b]	• [c]
Organization	• [a]	• [b]	• [c]
Roadmap	• [a]	• [b]	• [c]
Cross-Cutting Concerns: • [data-architecture] • [security] •			
Cross-Cutting Rituals, Practices and Techniques: • Architecture Decision Record • Architecture Kata •			

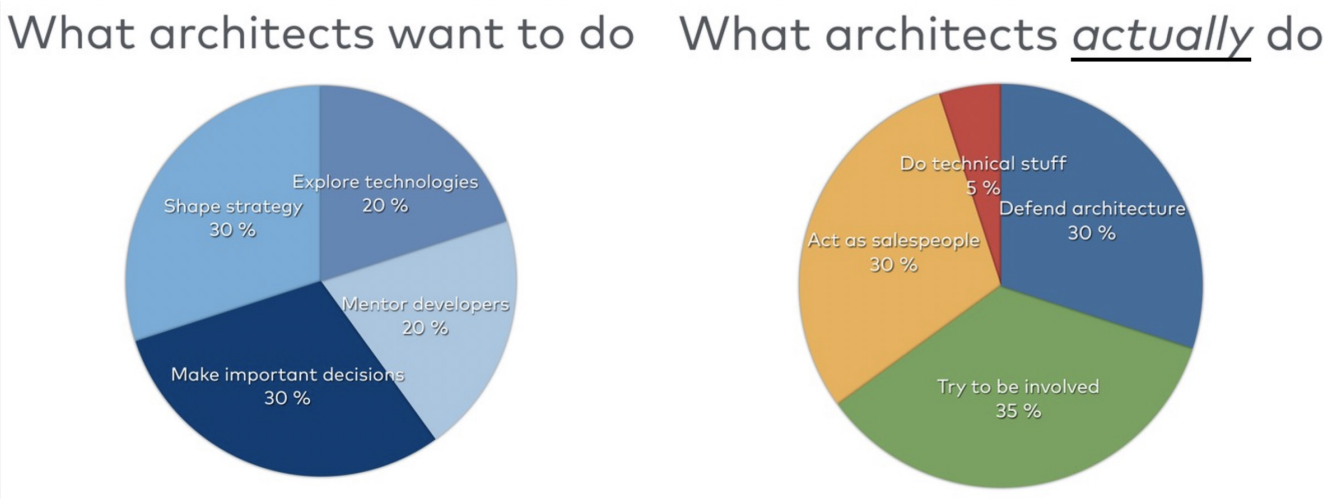
Chapter 3. Rituals

3.1. Architecture Kata

By Olivier Jauze <olivier.jauze@michelin.com> and Nicolas Chevalier <nicolas.chevalier@gluendo.com>

3.1.1. Can we help architects to practice?

Unless you're practicing architecture every day, we often spend our time on very different activities as depicted on the below charts.



(courtesy from Stefan Tilkov).

So if you find yourself in this situation, we have a ritual that was created to help you: it's the architecture kata and we want to give credits to Ted Neward who formalized it several years ago.

3.1.2. Why having a ritual to practice architecture?

The two below quotes are self explanatory we believe:

- "How do we get great designers? Great designers design, of course." --Fred Brooks
- "So how are we supposed to get great architects, if they only get the chance to architect fewer than a half-dozen times in their career?" --Ted Neward

Naming this ritual kata is a direct analogy to the Japanese Kata word (□, □) where it literally means "form referring to a detailed choreographed pattern of martial arts movements made to be practised aloneor within groups and in unison when training. It is practised in Japanese martial arts as a way to memorize and perfect the movements being executed." — Wikipedia

The idea of an architecture kata is then to repeat the architecture "movement" often enough to memorize it and perfect it over time.

3.1.3. How the ritual is conducted?

There are only two pre-requisites to organize an architecture kata: you need a topic / problem to work on and you need to have at least 2 hours ahead of you.

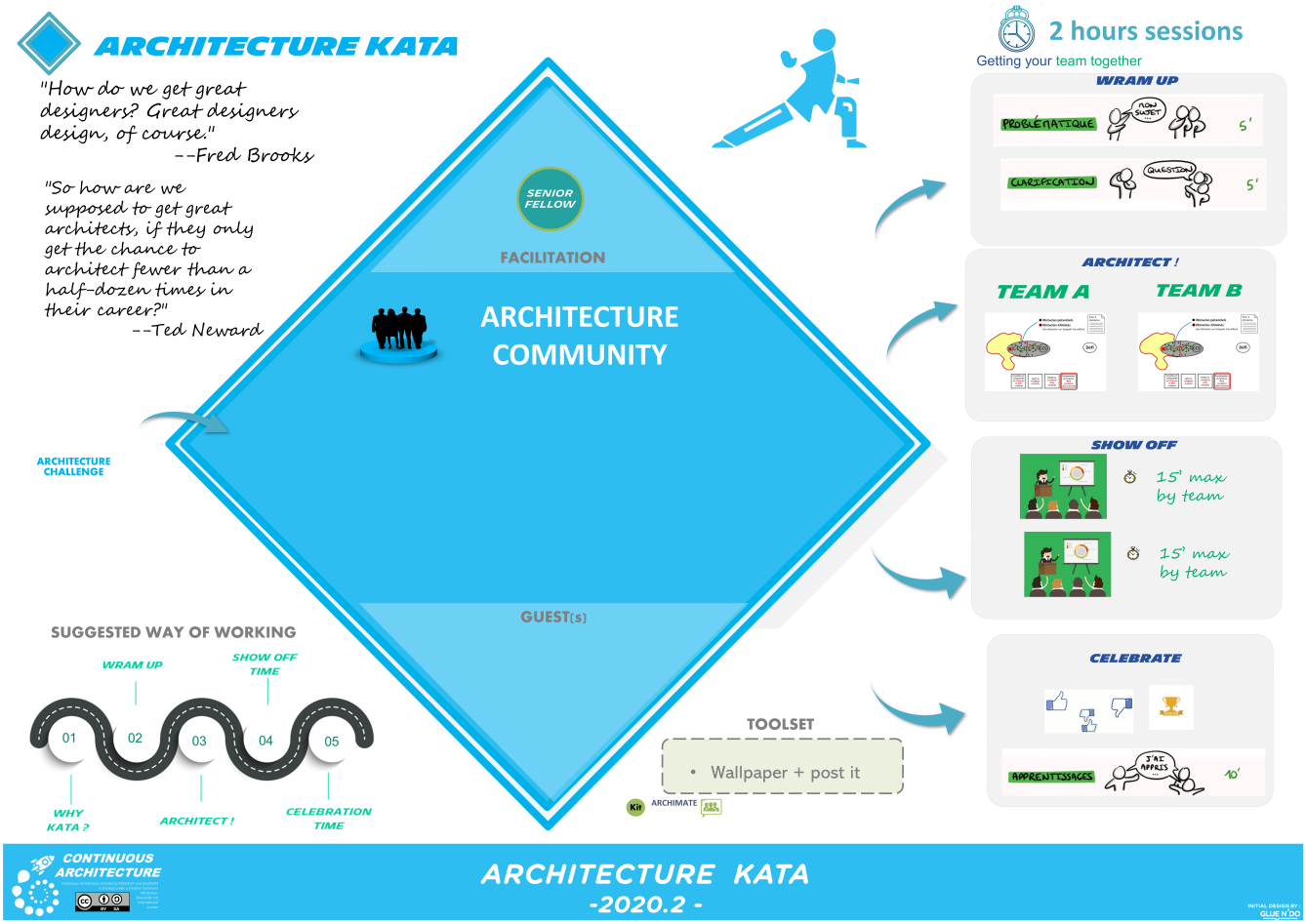
A kata is organized is 4 stages

1. Warm-up phase: the moderator (who can be seen as the customer) presents the topic / problem he wants the different groups to work on. He takes time to answer any clarification questions so attendees discovers the topic. Then we assemble the groups / teams. Depending on the number of persons attending the kata, we can create one or several teams of 5 people. If you have multiple teams, there will be another benefit to this practice: having several teams solving your problem, you may end up finding one solution or more ;)
2. Architect phase: each team is trying to outline an architecture that fix the given problem. And there are very few rules to respect
    - You may ask the Moderator any questions you have about the topic/problem.
    - Any technology is fair game. Customers really don't care, most of the time, what kind of technology you use. Just be prepared to defend it use during the show off Phase.
    - You may safely make assumptions about technologies you don't know well. However, any assumptions you make under this rule must be clearly defined and described during the show off phase.
    - You may not assume you have hiring/firing authority over the development team. No assumptions of developer All-Stars; assume that a development team roughly as skilled as the one you work with on a daily basis will be doing the implementation.
3. Show off phase (or peer review): During this phase, your team will be either presenting to other groups, or listening to other groups. If you are presenting, you have to explain your proposal and answer questions on it. If you are listening to the other groups, your job is to ask questions and please try to keep the questions constructive. But feel free to openly question any choice or decision that you think might not have been carefully examined or thought out. The moderator here will help to lower down the tension if a kind of back-and-forth passionate discussion starts. Remember that teams may assume anything about a technology they don't know well, so long as that assumption is clearly spelled out; if they assumed something that you know to be false, by all means inform them of that, but bear in mind that someone else in the room may have different experience with it than you, and keep an open mind. Also keep in mind that you'll be presenting your solution soon ;)
4. Vote and celebration phase: after each team has finished their presentation, we move to the voting phase. The Moderator will call out a "1-2-3", and you will each individually give the team an overall feedback vote:
    -  : You thought they nailed it, answered all obvious questions, chosen credible and feasible tech that at least seems credible and feasible, and they have a basic vision of solution. This doesn't mean they have every answer, or that they have already produced a UML diagram, it means that you have confidence they could produce them given enough time.
    -  : You thought they missed a few things, enough to make you a bit uncomfortable

that they really have a clear vision of what they're trying to build. They forgot some key questions to ask the customer, they forgot some important aspect, or they just in general didn't give you the feeling that they really "got it".

- 👎 : You thought they missed it. They made major assumptions that you think had no validity to it. They never asked the customer any questions and got burned on a few bad assumptions as a result. They really bungled the job, and you would never want to work with them on this topic.
- But all is not finished! Once the voting is complete, the ancient Klingon proverb must be heeded: "Revenge is a dish best served cold". The team departing the stage chooses the next one, and we repeat again by going back to the Vote and celebration Phase for them.

The below visual recaps how an architecture kata is conducted:



Chapter 4. Practices

4.1. Architecture Decision Record

By Olivier Jauze <olivier.jauze@michelin.com> and Nicolas Chevalier <nicolas.chevalier@gluendo.com>

4.1.1. Decisions, you said decisions?

Taking decisions plays a big role in a life of an architect. Almost every day we take decisions. Instead of trying to define what a decision is and on what they should be taken, we prefer to refer to Martin Fowler who said "Architecture is about the important stuff. Whatever that is". So decisions have to be made for important topics ;) The rule of thumb here could be: a decision is needed when the cost of change / rework will be important.

The last responsible moment

Our creed is to wait for the last responsible moment to take decisions. There are two main reasons for this approach:

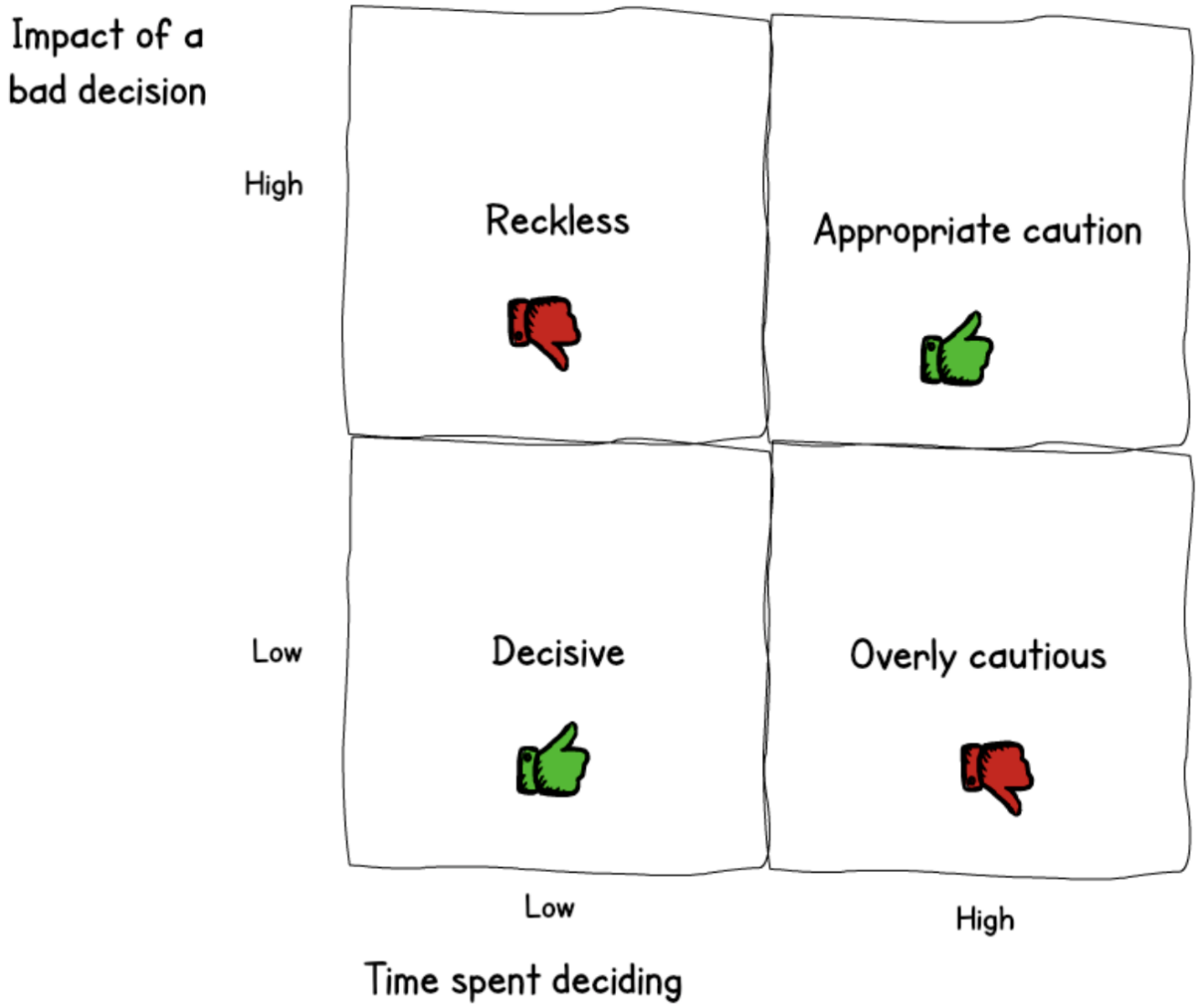
1. it will be easier to take decisions once you have more information than upfront
2. the decision process will not be a bottleneck in the delivery process.

There are two main caveats with this approach. The first one is to determine when you think you have enough information to decide. The last thing we want is to have architects waiting indefinitively to have all the informations they think they need. Some times you need to make hypothesis instead of waiting the information. The second one is that it can generate some inconfort within your team. Indeed, quite often team members are waiting the approval (or blessing would be more appropriate) from "the archtiects" before proceeding.

4.1.2. How much time should I spend on this decision?

Jason Yip in Impact of a bad decision vs time spent deciding came up with an interesting perspective on the time we should spend to take decision. To determine it, he combined the time spent deciding with the impact of a bad decision. He ended up with 4 quarands:

- Reckless: High impact of a bad decision, low time spent deciding;
- Appropriate caution: High impact of a bad decision, high time spent deciding;
- Decisive: Low impact of a bad decision, low time spent deciding;
- Overly cautious: Low impact of a bad decision, high time spent deciding.



You should try to position your decisions in these quadrants to make sure you're spending the appropriate time on them.

4.1.3. Who contribute to this decision making process?

Should all decisions be made by architects? no we don't think so but architects are clearly leading this process. Once, we've said that, it's implicit that whoever in the team have something to say on this decision should be contributing to this process. We also want to make it clear that unless your team is completely autonomous on everything, external contributors can also be involved: it could be subject matter experts, other architects, CTOs …

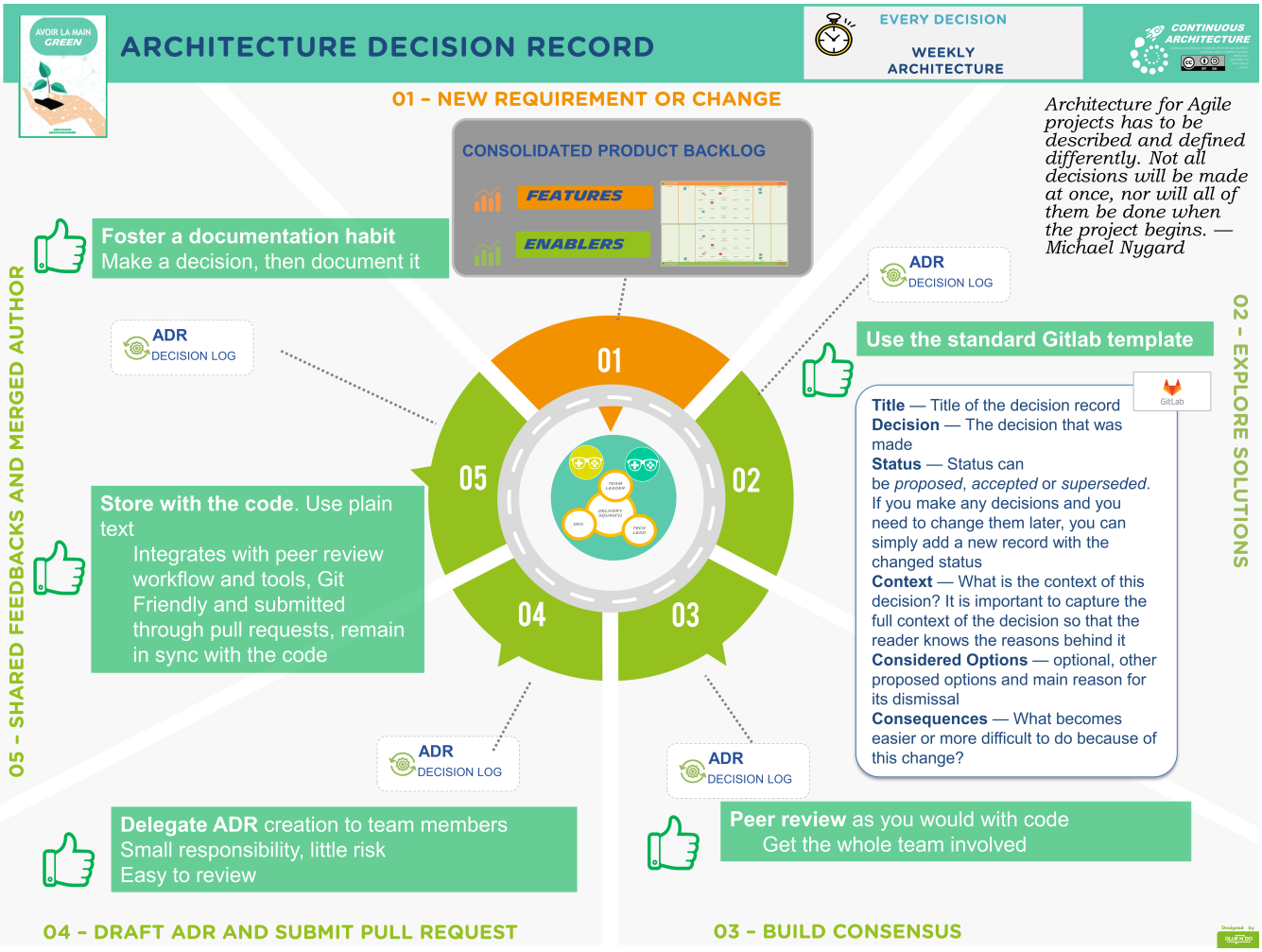
4.1.4. The decision making process

Speaking of processes these days seems a bit awkward maybe. But we think we need a small dose of process here as it can really help to produce high quality decisions. Making decisions goes through 5 different steps:

1. It starts with a trigger: why the hell should we make a decision? Is there something in the backlog of your product that requires the team to make a decision. It could be a business feature, a technical constraints, a problem coming from the run or an intentional change.

2. You need to document your decision to be factual and accurate. What is the problem you're trying to solve? what are the different options you are considering? what are your criterias to decide? what are the consequences of this decision? At this stage, we do recommend to log this decision as an artefact in your product repository. Why? because we think this decision, which is a documentation asset, is attached to the product itself and as such should be managed and versionned along with the other assets like source code for instance. If your using `Git`, you can look at this repository to get some inspiration.
3. You need to build a consensus on that decision to avoid the ivory tower architect and top down decisions syndroms. A good way to achieve it is to make proof of concepts and making sure your ADR is updated with its results. If you don't have the time to execute a PoC, brainstorming sessions or workshops can also be used.
4. Once you've reached a good maturity level, you can submit your ADR to either you team (through the weekly architecture ritual) or to your peers (through a peer review). The idea here is to get as much feedback as possible and ensure alignment on your decision.
5. At this stage, you do normally have enough material and reached a consensus to approved the decision. The `pull - merge` request process from `{Git}` is a good way to support this step. Merging your architecture decision (which is a markdown file for instance) into your product repository is the formal validation.
6. Do not forget to communicate your decision to your community (other product teams, infrastructure, security, enterprise architecture, stake holders ...). Too often we forget this last step and it can affect the efficiency and impact of your decision. If nobody knows about it, do you believe that it will be respected? Sometimes, it requires to create special documents to make the decision understandable by different audiences.

We provide a visual that illustrate this practice. You'll find it below:


ARCHITECTURE DECISION RECORD
EVERY DECISION
WEEKLY ARCHITECTURE
CONTINUOUS ARCHITECTURE
01 - NEW REQUIREMENT OR CHANGE
CONSOLIDATED PRODUCT BACKLOG
FEATURES
ENABLERS
Architecture for Agile projects has to be described and defined differently. Not all decisions will be made at once, nor will all of them be done when the project begins. — Michael Nygard
Foster a documentation habit
Make a decision, then document it
ADR DECISION LOG
ADR DECISION LOG
Use the standard Gitlab template
Title — Title of the decision record
Decision — The decision that was made
Status — Status can be proposed, accepted or superseded. If you make any decisions and you need to change them later, you can simply add a new record with the changed status
Context — What is the context of this decision? It is important to capture the full context of the decision so that the reader knows the reasons behind it
Considered Options — optional, other proposed options and main reason for its dismissal
Consequences — What becomes easier or more difficult to do because of this change?
02 - EXPLORE SOLUTIONS
01
02
03
04
05
05 - SHARED FEEDBACKS AND MERGED AUTHOR
Store with the code. Use plain text
Integrates with peer review workflow and tools, Git Friendly and submitted through pull requests, remain in sync with the code
ADR DECISION LOG
ADR DECISION LOG
Delegate ADR creation to team members
Small responsibility, little risk
Easy to review
Peer review as you would with code
Get the whole team involved
04 - DRAFT ADR AND SUBMIT PULL REQUEST
03 - BUILD CONSENSUS

Chapter 5. Techniques

5.1. Jobs To Be Done Analysis

By Frédéric Lé <fle@agileddd.com>.

Jobs To Be Done analysis is a customer research technique that aims at understanding the goals that people want to accomplish under given circumstances. Jobs the motivators and drivers of behavior: they predict why people behave the way they do.

The figure Jobs To Be Done Documentation template provide a structure way of documenting JTBD.

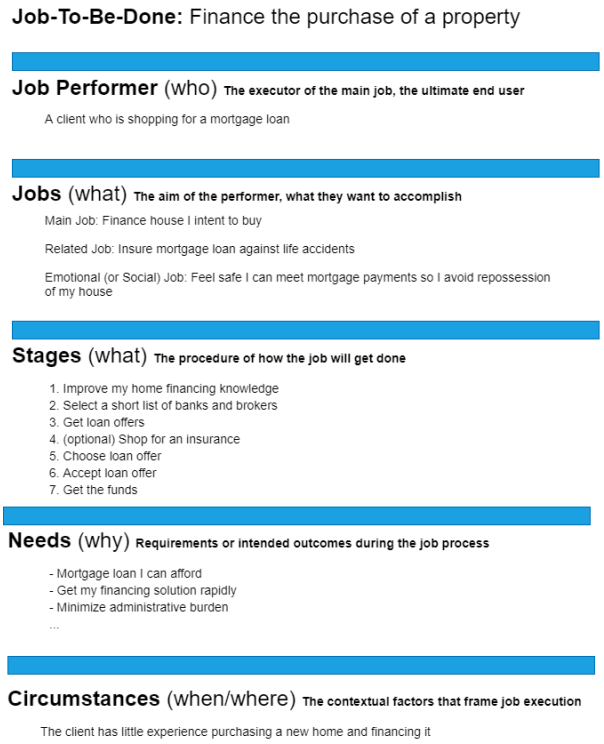


Figure 2. Jobs To Be Done Documentation template