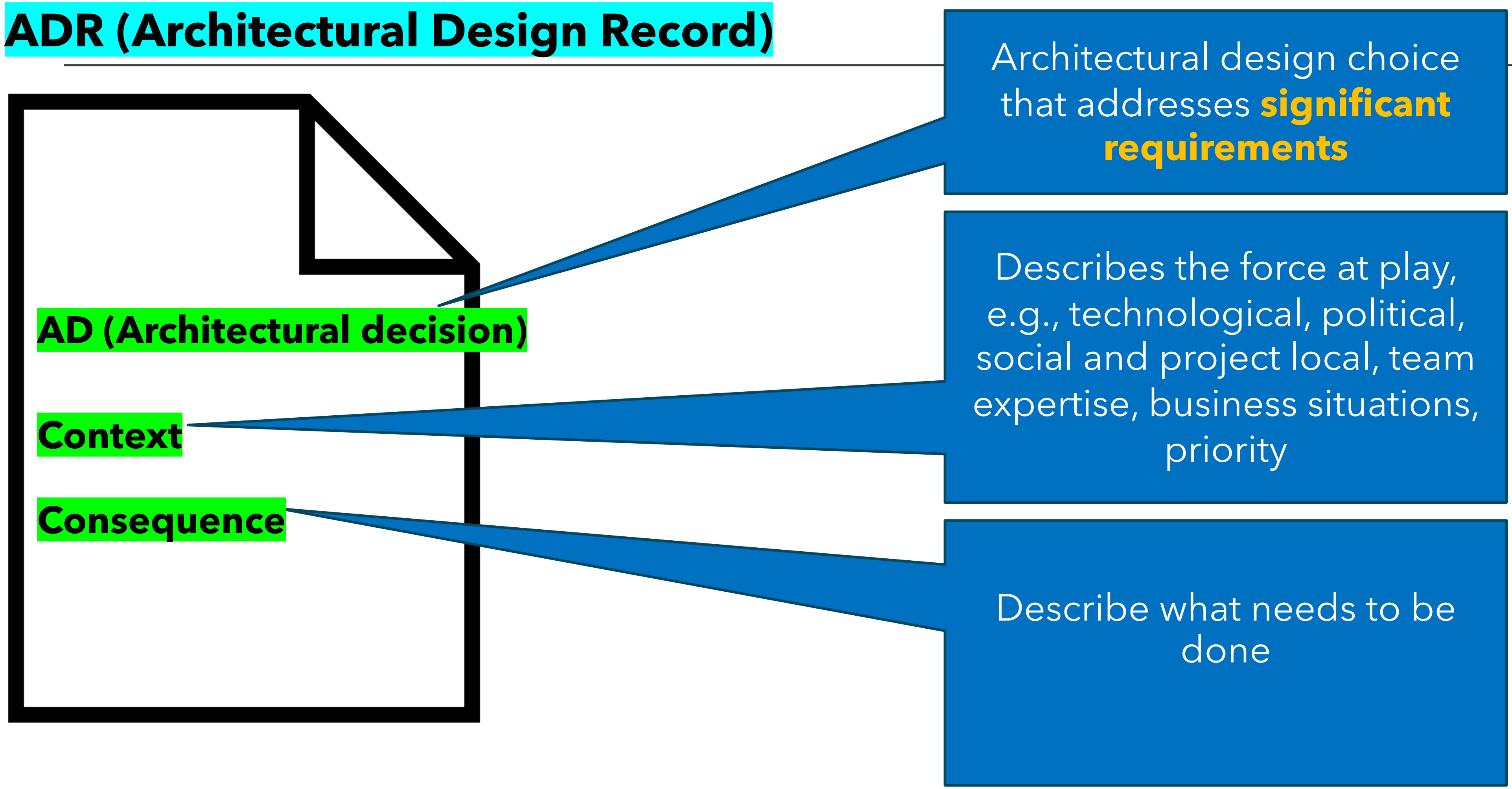
Architectural Decision Record (ADR)

Wiwat Vatanawood
Duangdao Wichadakul
Pittipol Kantavat
Neungwong Tuaycharoen

ADR and related terms

ADR (Architectural Design Record)



AD (Architectural decision)

Architectural design choice that addresses **significant requirements**

Context

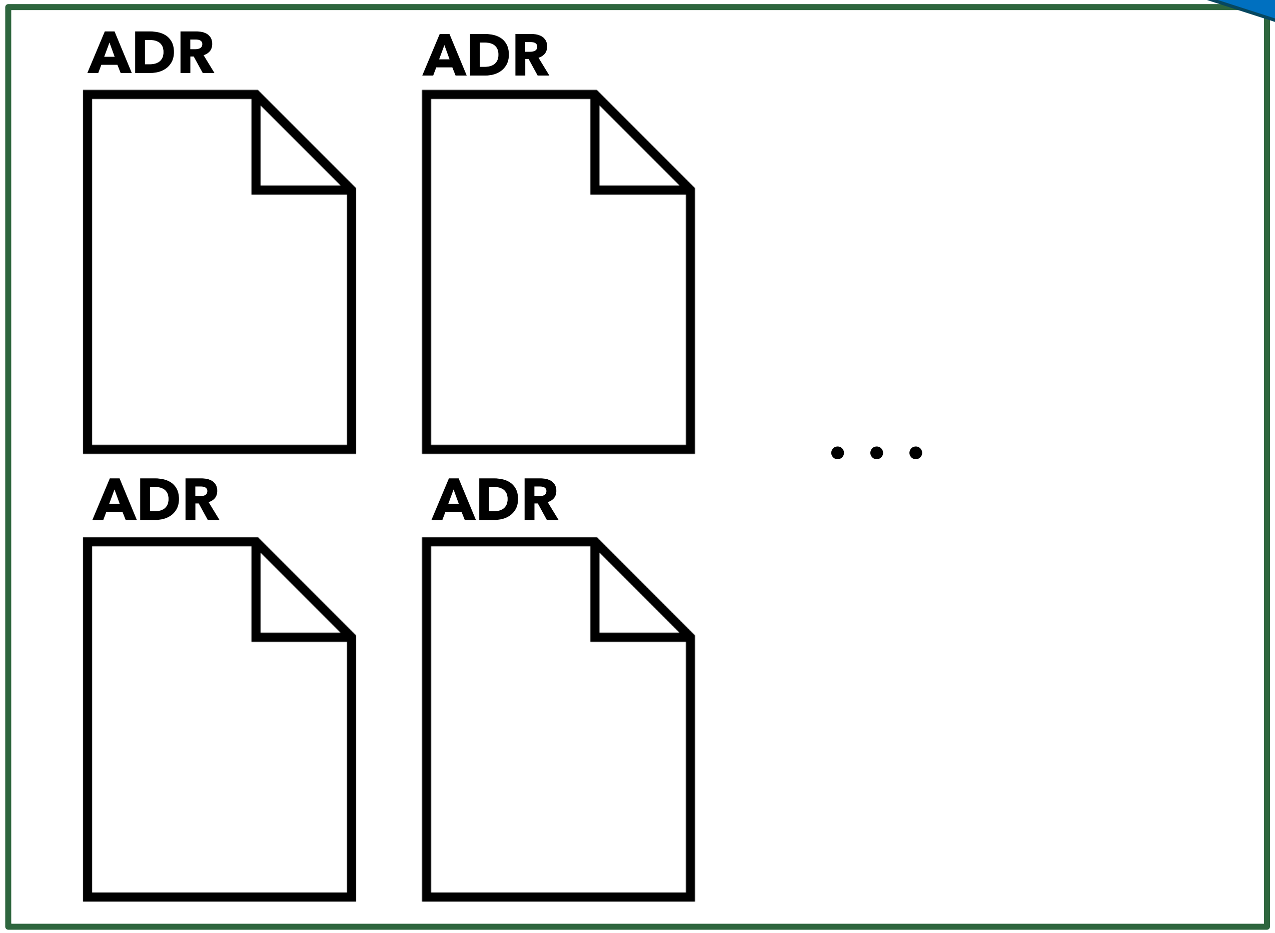
Describes the force at play, e.g., technological, political, social and project local, team expertise, business situations, priority

Consequence

Describe what needs to be done

ADR and related terms (cont.)

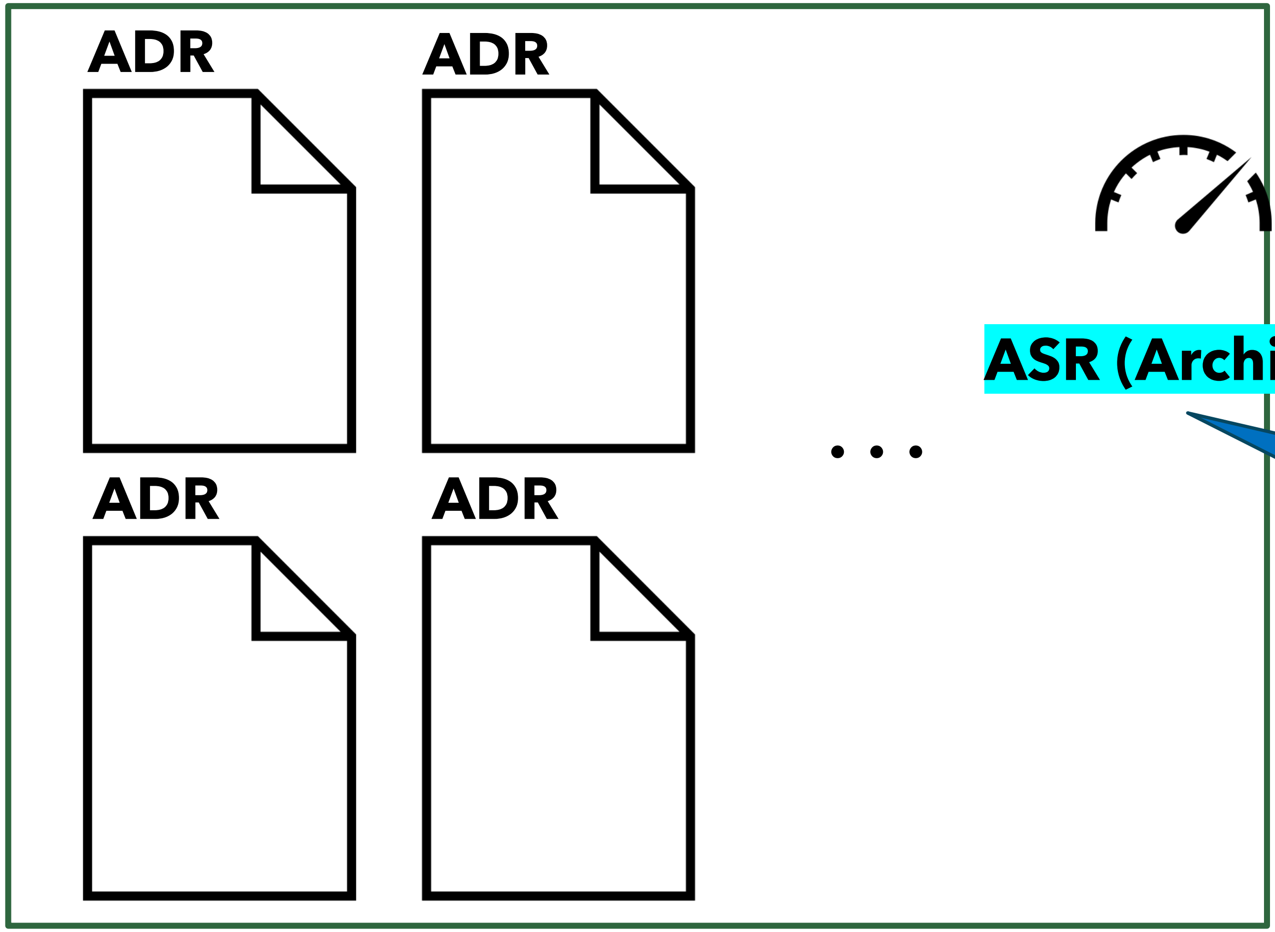
ADL (Architectural Decision Log)



Collected for a particular project or organizaiton

ADR and related terms (cont.)

ADL (Architectural Decision Log)

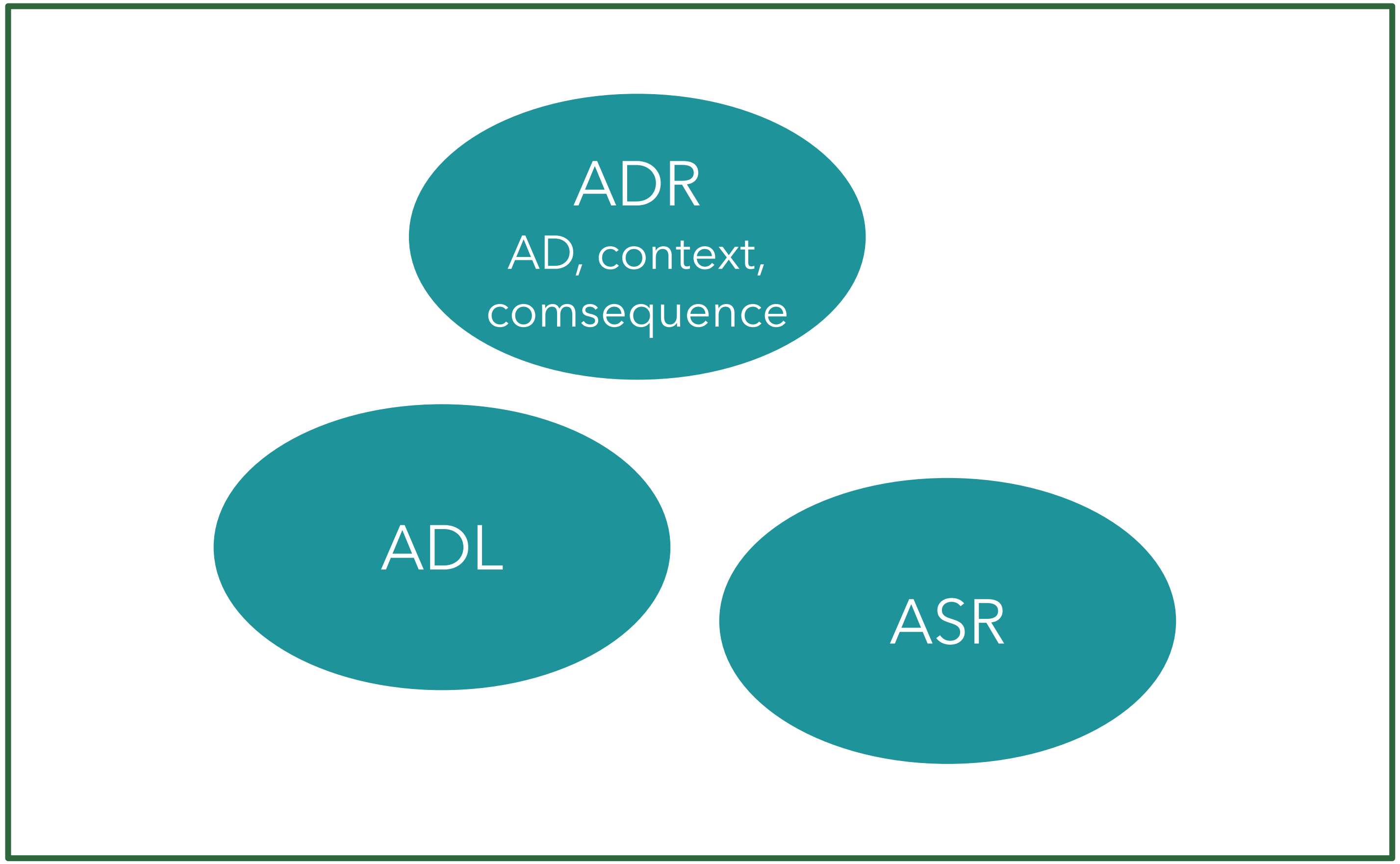


ASR (Architecturally Significant Requirements)

Requirement that has measurable effect on the software system's architect

ADR and related terms (cont.)

AKM (Architecture Knowledge Management)



ADR and related terms

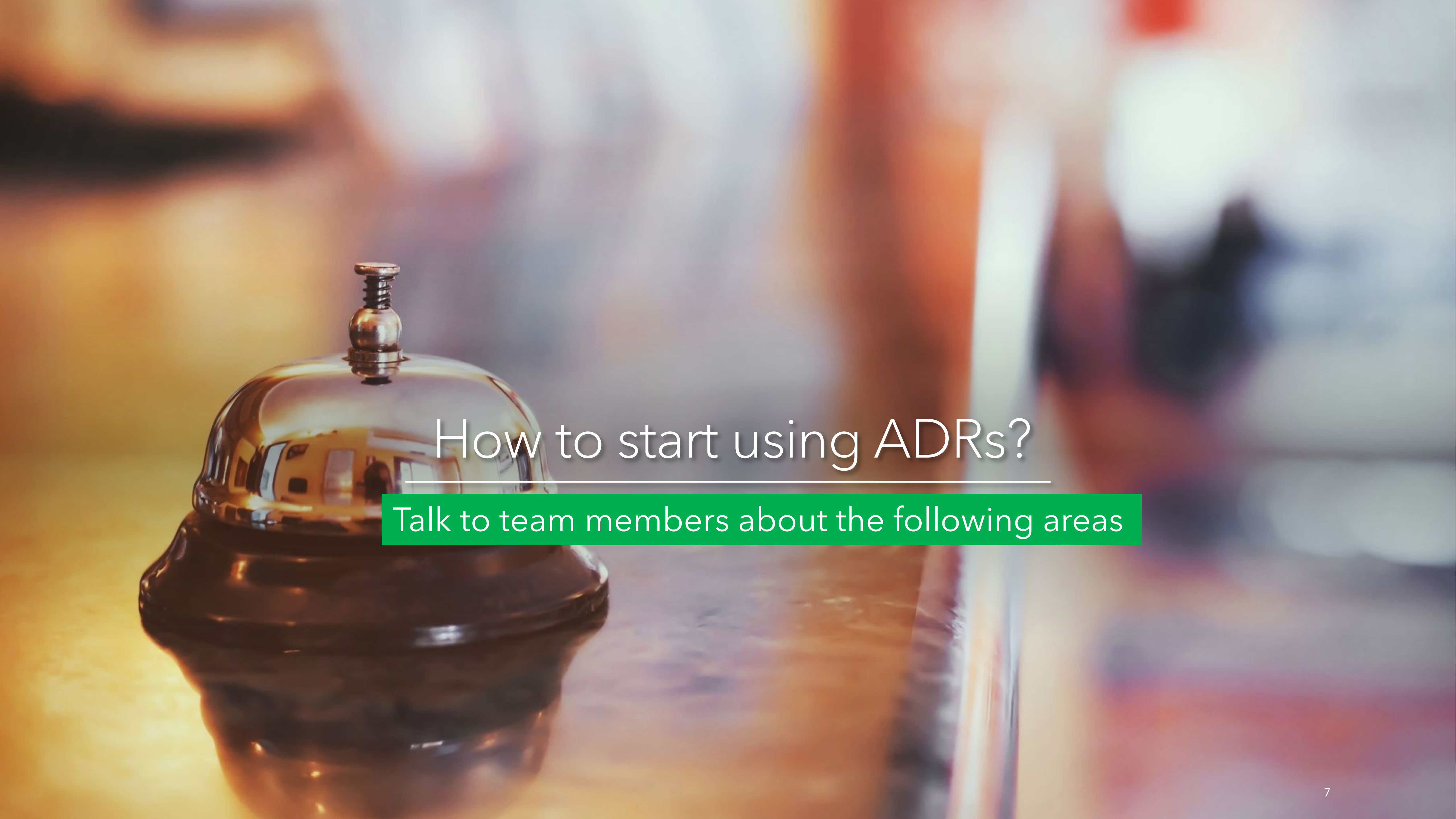
An architecture decision record (ADR) : a document that captures an important **architectural decision (AD)** made along with its **context** and **consequence**.

An architecture decision (AD) is a software **design choice** that addresses a significant requirement.

An architectural decision log (ADL) : **collection of ADRs** created and maintained for a particular project or organization.

An architecturally-significant requirement (ASR) : a requirement that has a measurable effect on a software system's architecture.

Architecture knowledge management (AKM) : the included topics of all above.

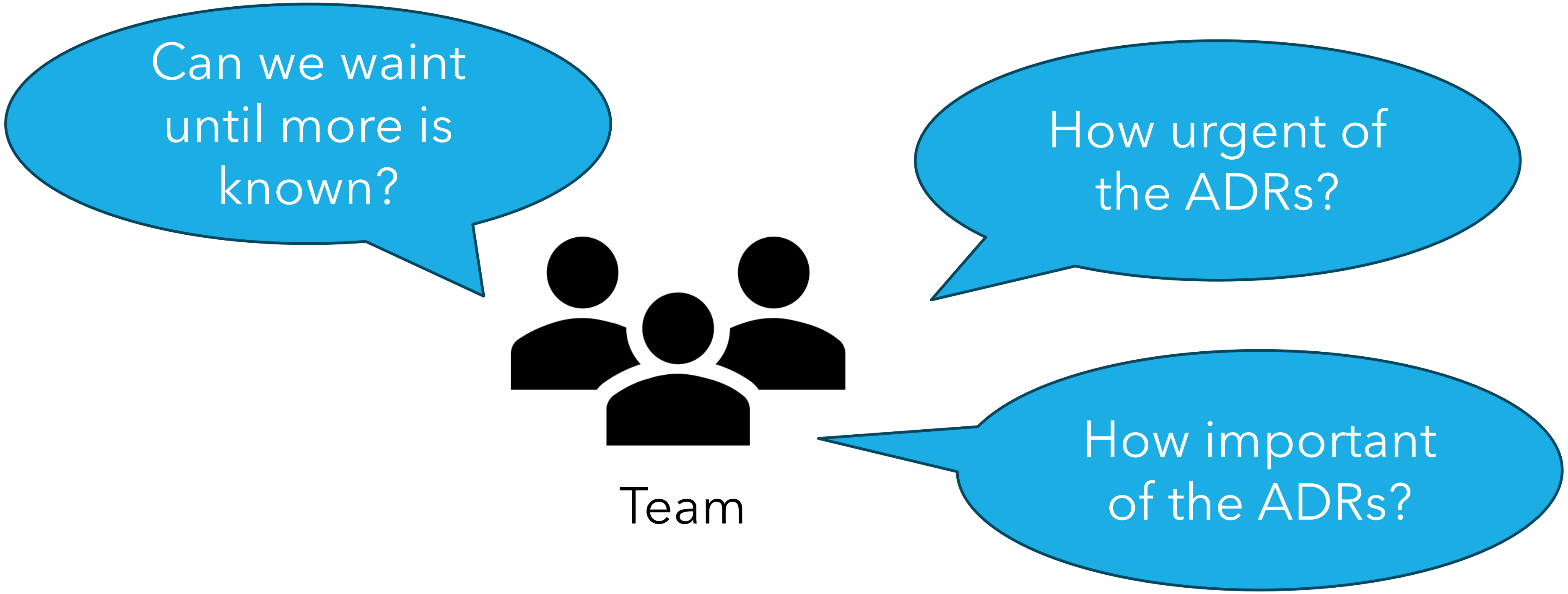


How to start using ADRs?

Talk to team members about the following areas

How to start using ADRs?

1. Decision identification



Can we waint until more is known?

How urgent of the ADRs?

How important of the ADRs?

Team

Maintain a **decision TODO list** that complements the **product TODO list**

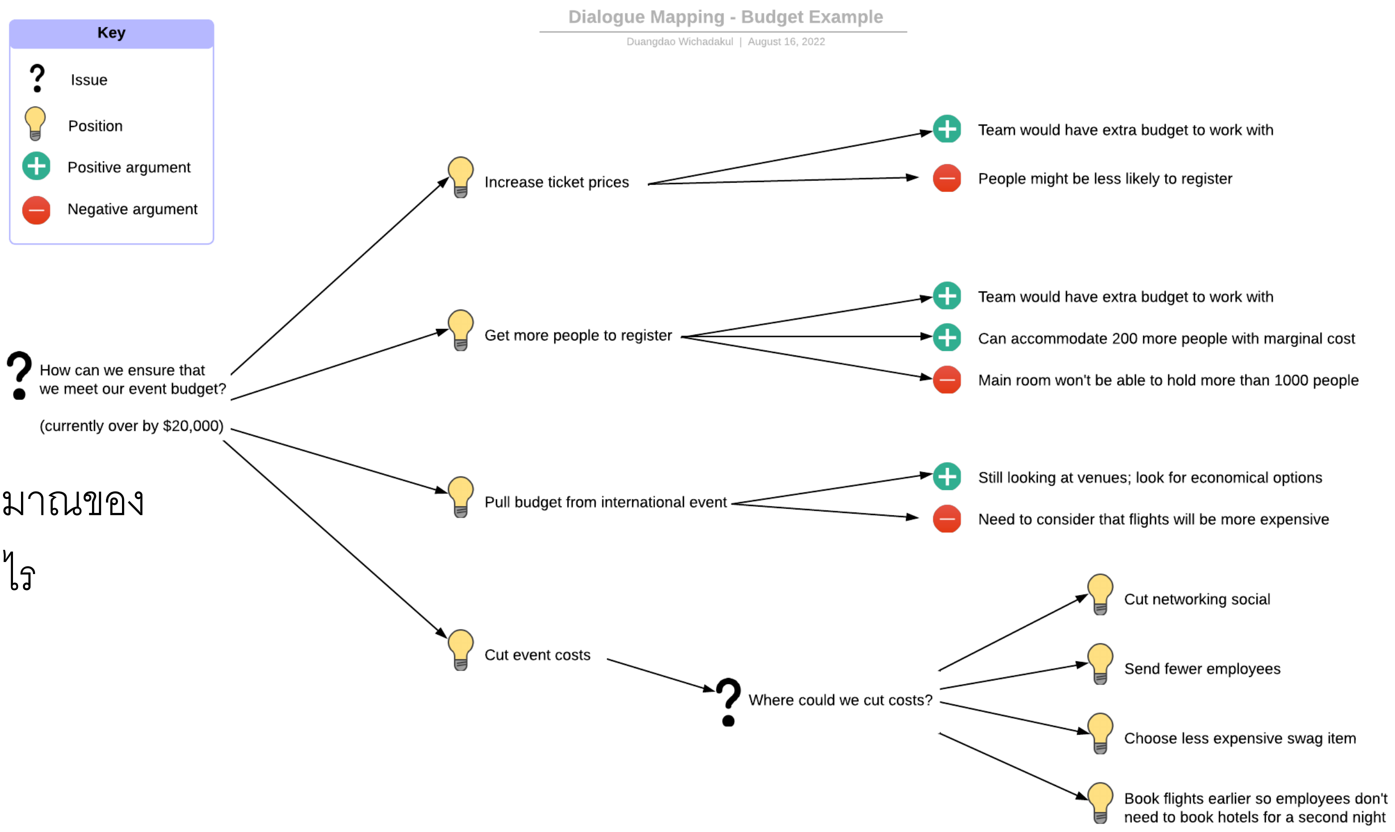
How to start using ADRs

2. Decision making

Apply existing decision-making techniques, e.g., dialogue mapping (https://www.lucidchart.com/blog/what-is-dialogue-mapping)

Group decision making (collaborative decision making) is an active research topic.

Example of dialogue mapping



Dialogue Mapping - Budget Example

Duangdao Wichadakul | August 16, 2022

Key

- ? Issue
- 💡 Position
- ⊕ Positive argument
- ⊖ Negative argument

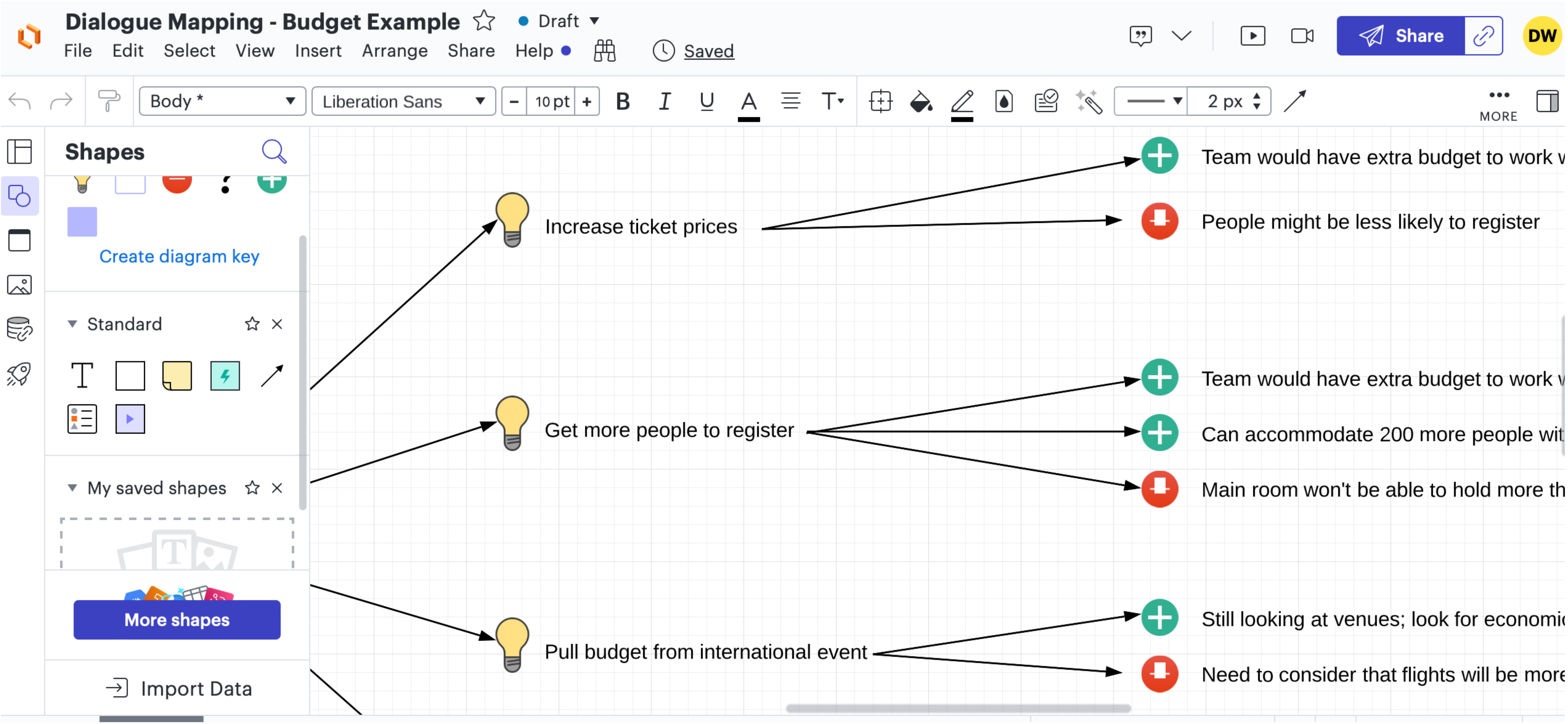
? How can we ensure that we meet our event budget?

(currently over by $20,000)

- 💡 Increase ticket prices
  - ⊕ Team would have extra budget to work with
  - ⊖ People might be less likely to register
- 💡 Get more people to register
  - ⊕ Team would have extra budget to work with
  - ⊕ Can accommodate 200 more people with marginal cost
  - ⊖ Main room won't be able to hold more than 1000 people
- 💡 Pull budget from international event
  - ⊕ Still looking at venues; look for economical options
  - ⊖ Need to consider that flights will be more expensive
- 💡 Cut event costs
  - ? Where could we cut costs?
    - 💡 Cut networking social
    - 💡 Send fewer employees
    - 💡 Choose less expensive swag item
    - 💡 Book flights earlier so employees don't need to book hotels for a second night

จะจัดสรรงบประมาณของ
โครงการได้อย่างไร

https://www.lucidchart.com/blog/what-is-dialogue-mapping



Increase ticket prices

Team would have extra budget to work w

People might be less likely to register

Get more people to register

Team would have extra budget to work w

Can accommodate 200 more people wit

Main room won't be able to hold more th

Pull budget from international event

Still looking at venues; look for economic

Need to consider that flights will be more

https://www.lucidchart.com/blog/what-is-dialogue-mapping

How to start using ADRs

3. Decision enactment and enforcement

ADs are used in **software design**; hence, must be **communicated to** and **accepted by the stakeholders** of the system that fund, develop, and operate.

Architecturally evident coding styles and code reviews that focus on architectural concerns and decisions are two related practices.

ADs must be reconsidered when modernizing a software system in software evolution.

How to start using ADRs

4. Decision sharing (optional)

Many ADs recur across projects

Experiences with past decisions (good or bad) will be valuable/reusable assets with AKM

How to start using ADRs

5. Decision documentation

Many templates and tools for decision capturing exist.

Agile communities, e.g., M. Nygard’s ADRs.

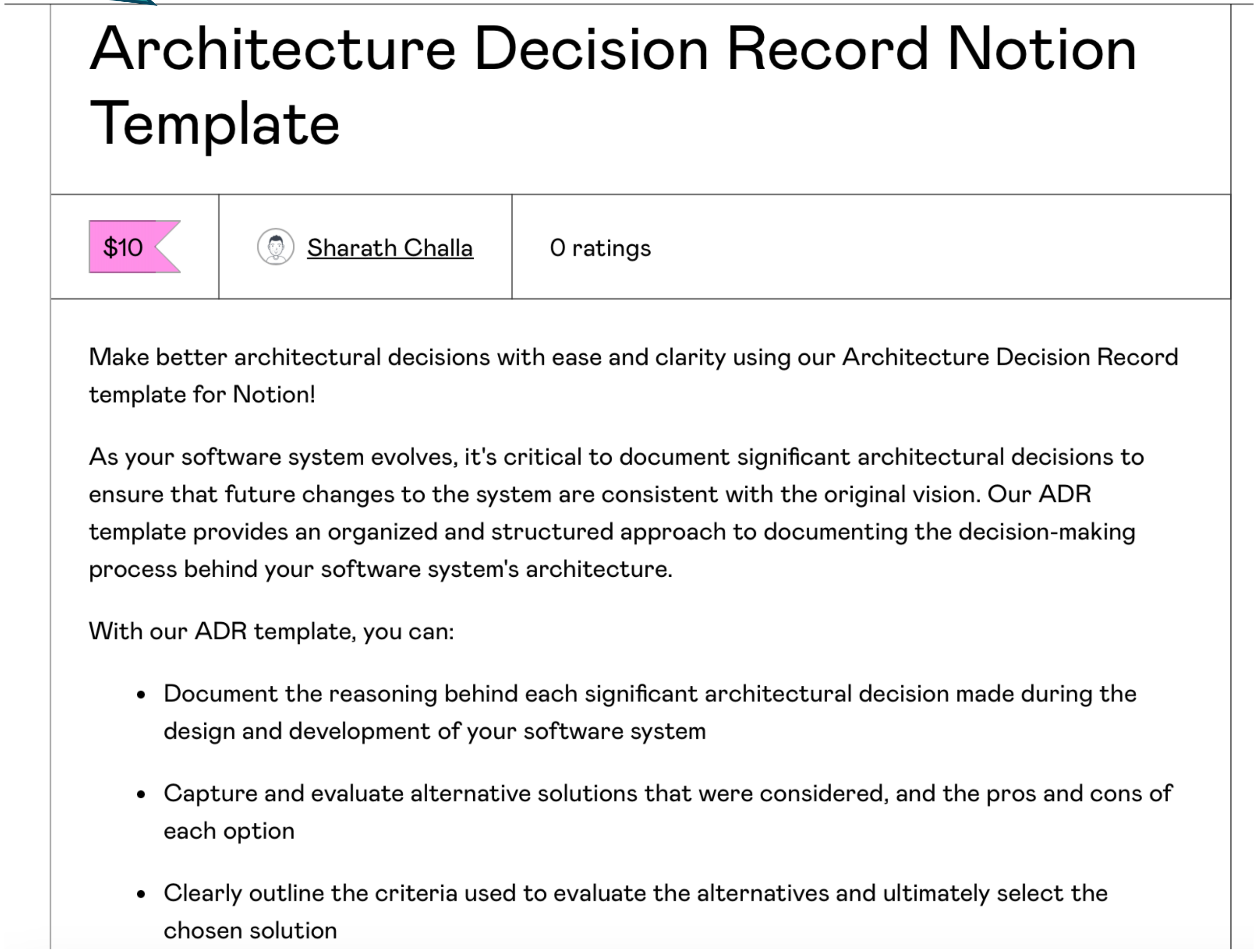
Traditional software engineering and architectural design processes, e.g., table layouts suggested by IBM UMF and by Tyree and Akerman from CapitalOne.

How to start ADRs with tools

- Google drive; google doc, google sheet, w/ online editing among team members
- Version control tools, e.g., git
- Project planning tools, e.g., Atlassian Jira w/ tool planning tracker
- wikis , e.g., MediaWiki

What else?

ตัวอย่าง Template จาก notion



Architecture Decision Record Notion Template

$10 | Sharath Challa | 0 ratings

Make better architectural decisions with ease and clarity using our Architecture Decision Record template for Notion!

As your software system evolves, it's critical to document significant architectural decisions to ensure that future changes to the system are consistent with the original vision. Our ADR template provides an organized and structured approach to documenting the decision-making process behind your software system's architecture.

With our ADR template, you can:

- Document the reasoning behind each significant architectural decision made during the design and development of your software system
- Capture and evaluate alternative solutions that were considered, and the pros and cons of each option
- Clearly outline the criteria used to evaluate the alternatives and ultimately select the chosen solution

HTTPS://SRC200.GUMROAD.COM/L/ADR

How to write a good ADR

Suggestions for writing good ADRs

Characteristics of a good ADR

Point in Time – Identify **when** the AD was made

Rationality – Explain **why** when making the particular AD

Immutable record – The previously published AD should not be altered

Specificity – Each ADR should be about a **single AD**

Suggestions for writing good ADRs

Characteristics of a good “Context” section in ADR

Explain your organization’s **situation** and business **priorities**

Include rationale and considerations based on social and skill makeups of your teams

บอกสถานการณ์ปัจจุบัน ลำดับความสำคัญของงาน รวมทั้งบริบทแวดล้อม เช่น ทีมที่มีอยู่มีทีมอะไรบ้าง มีทักษะอะไรบ้าง

Suggestions for writing good ADRs

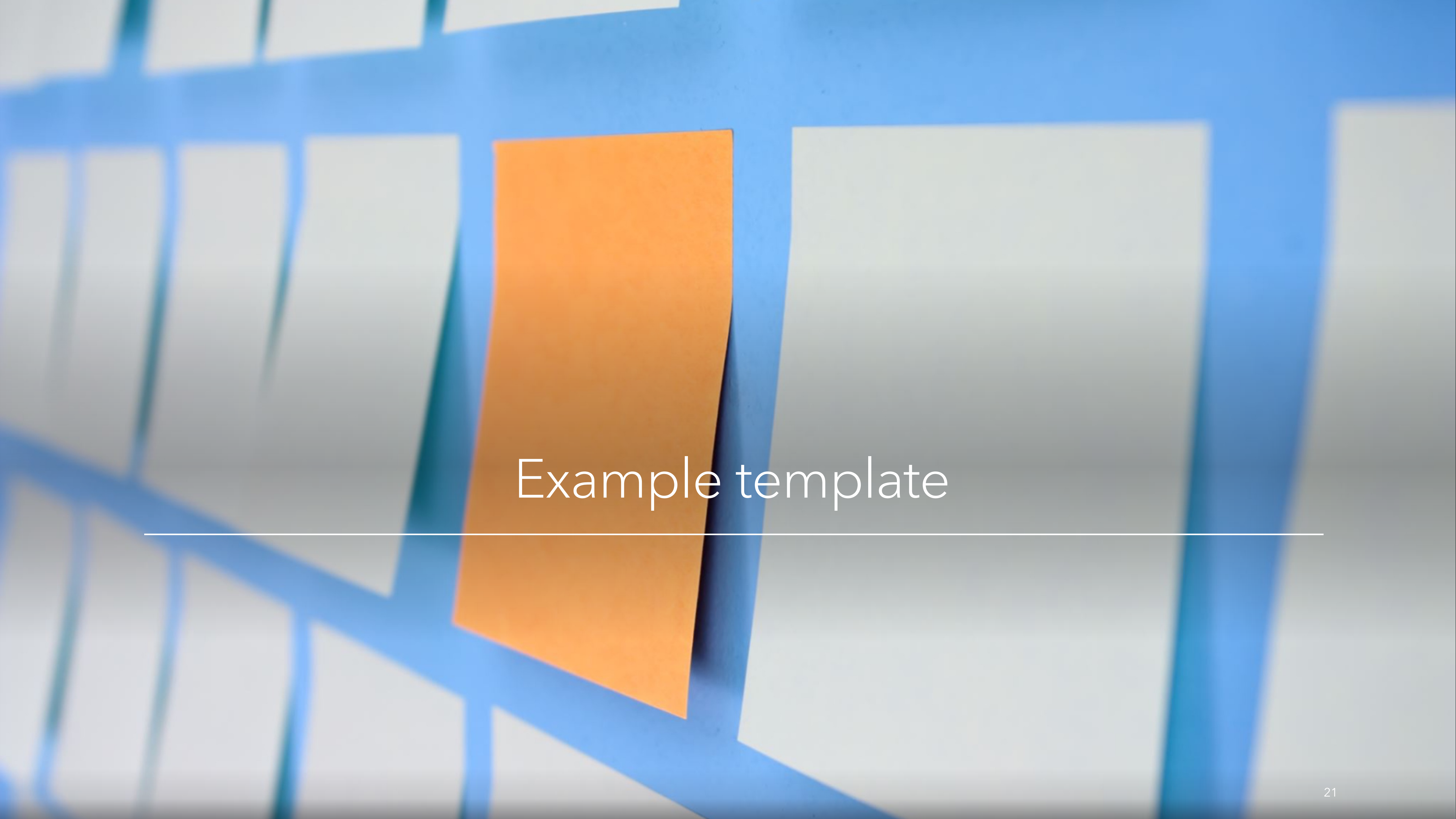
Characteristics of a good "Consequences" section in ADR

✓ Right approach – "We need to start doing X instead of Y"

✗ Wrong approach – Do not explain the AD in terms of "Pros" and "Cons"

อย่าบอกแค่ข้อดี ข้อเสีย แต่ต้องบอกว่าตัดสินใจว่าเลือกอะไร

Example template



ADR example templates

ADR template by Jeff Tyree and Art Akerman (more sophisticated)

ADR template by Michael Nygard (simple and popular)

ADR template for Alexandrian pattern (simple with context specifics)

ADR template for business case (more MBA-oriented, with costs, SWOT, and more opinions)

ADR template MADR (more Markdown)

ADR template using Planguage (more quality assurance oriented)

Decision record template by Jeff Tyree and Art Akerman

Issue : Why do we address this?

Decision : The architecture's direction; the position we selected.

Status : The decision's status; pending, decided, approved

Group : Assigned category for the decision; help organize the set of decisions

Assumptions : Assumptions when the decision is made; schedule, cost, tech,…

Constraints : Capture any additional constraints that the decision might pose.

Positions : Other options / alternatives; to avoid question "Did you think about..?"

Argument : Outlines why you selected a position; implementation cost, total ownership cost, time to market, and required development resources' availability

Implication : A decision may come with many implications; creating new requirements; modifying existing requirements, pose additional constraints, require negotiating scope or schedule with customer, require additional staff training

Decision record template by Jeff Tyree and Art Akerman

Related decisions : List the related decisions here; traceability matrix, decision trees, metamodels

Related requirements : Decisions should be business-driven. To show accountability, map decisions to the objectives or requirements. Assess each architecture decision's contribution to meet each requirement, then, assess how well if a requirement is met across all decisions.

Related artifacts : List the related architecture, design or scope documents that this decision impacts.

Related principles : Make sure the decisions are aligned with the enterprise's principles if available.

Notes : Capture notes during the team discussions and the socialization process.

14 หัวข้อ!!!

Decision record template by Jeff Tyree and Art Akerman (Grouping)

Summary

- **Issue**
- **Decision**
- **Status**
- **Group**

Details

- **Assumptions**
- **Constraints**
- **Positions**
- **Argument**
- **Implications**

Related

- **Related decisions**
- **Related requirements**
- **Related artifacts**
- **Related principles**

Notes

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

Example 1 : Environment variable configuration

Summary

- **Issue**
- **Decision**
- **Status**
- **Group**

Details

- **Assumptions**
- **Constraints**
- **Positions**
- **Argument**
- **Implications**

Related

- **Related decisions**
- **Related requirements**
- **Related artifacts**
- **Related principles**

Notes

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

Example 1 : Environment variable configuration

Summary

Issue

บริบทที่เกี่ยวข้องกับความต้องการ

We want our applications to be configurable beyond artifacts/binaries/source, such that one build can behave differently depending on its deployment environment.

- To accomplish this, we want to use environment variable configuration.
- We want to manage the configuration by using files that we can version control.
- We want to provide some developer experience ergonomics, such as knowing what can be configured and any relevant defaults.

Decision

ผลการตัดสินใจ

Decided on .env files with related default file and schema file.

Status

สถานะของผลการตัดสินใจ

Decided. Open to considering to new capabilities as they come up.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

Example 1 : Environment variable configuration

Details

Assumptions

มีสมมติฐานอะไรบ้าง เกี่ยวกับ
การตัดสินใจข้างต้น

We favor separating the application code and environment code. We assume the app needs to work differently in different environments, such as in a development environment, test environment, demo environment, production environment, etc.

We favor the industry practice of "12 factor app" and even more the related practice of "15 factor app".

Many of our previous projects have used the convention of a .env file or similar .env directory. There's a typical practice of keeping these out of version control, and instead using some other way to deploy them, version them, and manage them.

Constraints

เงื่อนไขที่ต้องพิจารณาเพิ่มเติม

We want to keep secrets out of our source code management (SCM) version control system (VCS).

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

Example 1 : Environment variable configuration

Details

Positions

ทางเลือกที่เป็นไปได้อื่นๆ

We considered a few approaches:

- Store config in the app such as in a file config.js file.
- Store config in the environment such as in a file .env.
- Fetch config from a known location such as a license server.

Argument

เหตุผลที่เลือกใช้ไฟล์.env

We selected the approach of a file .env because:

- It is popular including among experts.
- It follows the pattern of .env files which our teams have successfully used many times on many projects.
- It is simple. Notably, we are fine for now with the significant trade-offs that we see, such as a lack of audit capabilities as compared to an approach of a license server.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

Example 1 : Environment variable configuration

Details

สิ่งที่ต้องพิจารณาเพิ่มเติมสืบเนื่องจากการ
ตัดสินใจข้างต้น

Implications

We need to figure out a way to separate environment variable configuration that is public from any secrets management.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

Example 1 : Environment variable configuration

Related

สิ่งอื่นๆ ที่เกี่ยวข้องกับการตัดสินใจข้างต้น

Related decisions

We expect all our applications to use this approach.

We will plan to upgrade any of our applications that use a less-capable approach, such as hardcoding in a binary or in source code.

We will keep as-is any of our applications that use a more-capable approach, such as a licensing server.

Related requirements

We will add DevOps capabilities for the files, including hooks, tests, and continuous integration.

We need to train all developer teammates on this decision.

Related artifacts

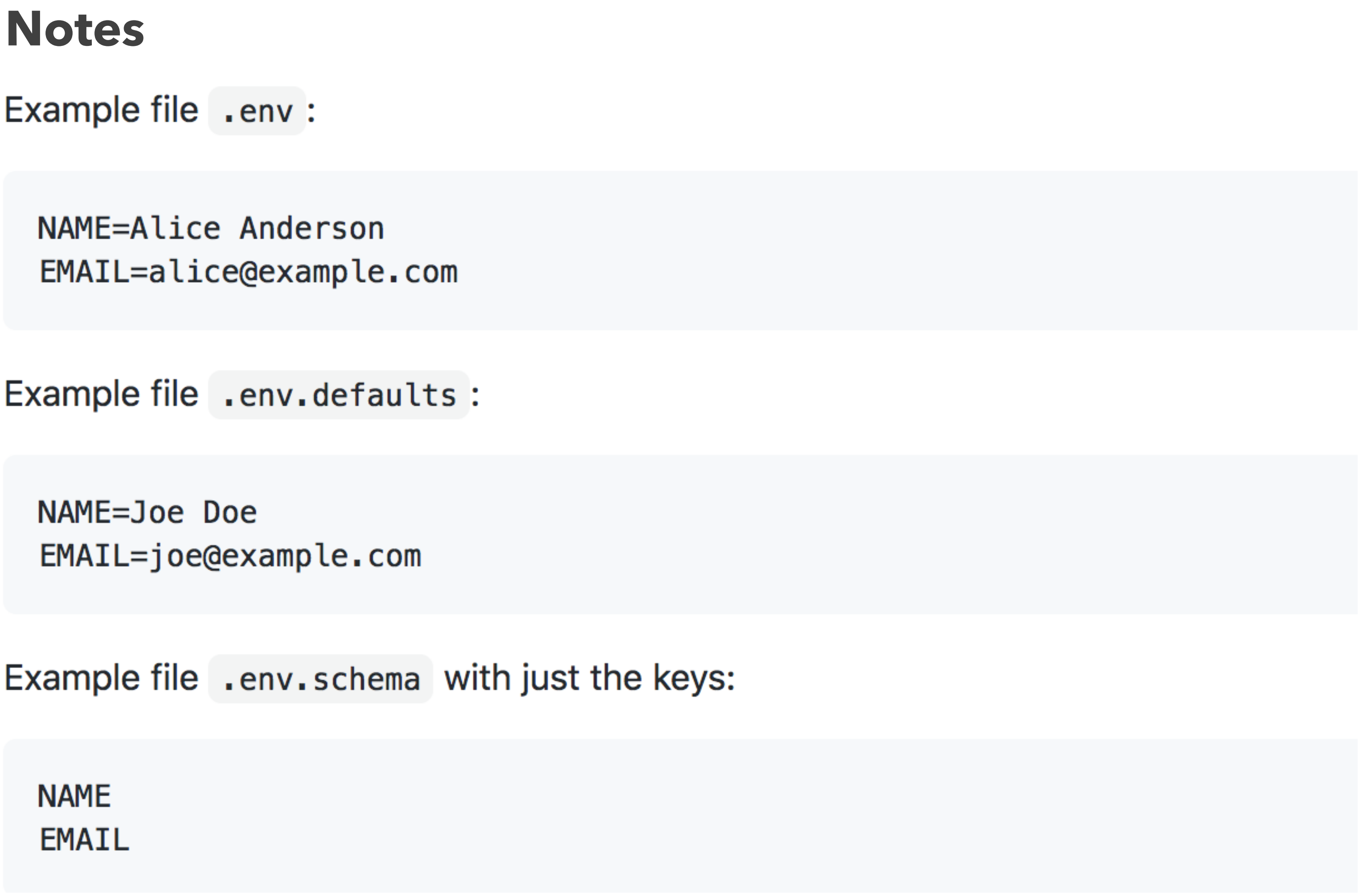
Each area where we deploy will need its own .env file and related files.

Related principles

Easily reversible.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

Example 1 : Environment variable configuration



Notes

Example file `.env` :

```
NAME=Alice Anderson
EMAIL=alice@example.com
```

Example file `.env.defaults` :

```
NAME=Joe Doe
EMAIL=joe@example.com
```

Example file `.env.schema` with just the keys:

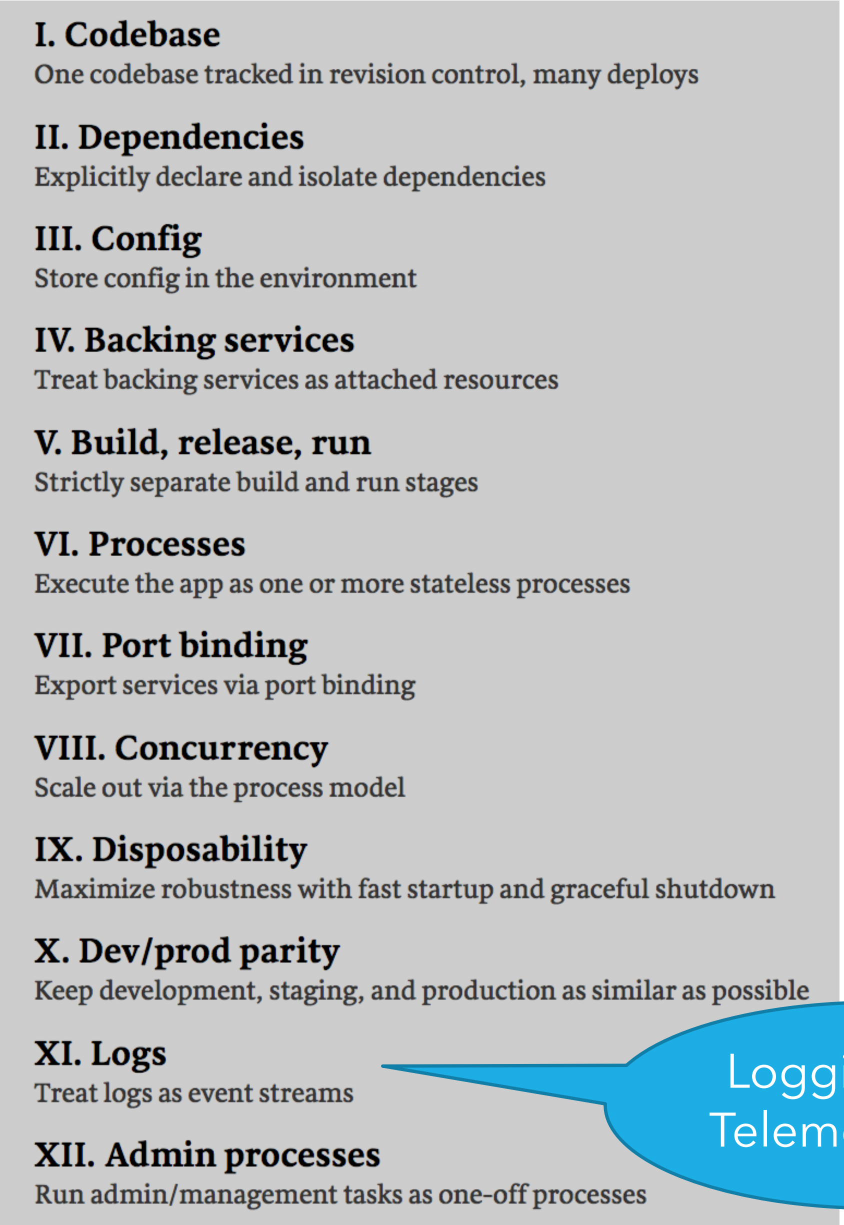
```
NAME
EMAIL
```

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

sidebar

The Twelve-Factor App

The 12-Factor App Framework was created in 2012 by developers at early cloud pioneer Heroku as a set of rules and guidelines for organizations building modern web applications that run "as a service." The framework was inspired by Martin Fowler's work on application architecture and what he saw as suboptimal application development processes.



I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Logging ต่างจาก
Telemetry อย่างไร

https://12factor.net/

Then, what is the Fifteen-Factor App??

XIII. API First
XIV. 14 Telemetry
XV. Authentication and Authorization
by Hoffman

https://github.com/cjudd/15-factor-app-workshop
https://developer.ibm.com/articles/15-factor-applications/

Example 2 : Metrics, monitors, alerts

Summary

- Issue
- Decision
- Status
- Group

Details

- Assumptions
- Constraints
- Positions
- Argument
- Implications

Related

- Related decisions
- Related requirements
- Related artifacts
- Related principles

Notes

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

Example 2 : Metrics, monitors, alerts

Summary

Issue

บริบทที่เกี่ยวข้องกับความต้องการ

We want to use metrics, monitors, and alerts, because we want to know how well our applications are working, and to know when there's a problem.

Decision

ผลการตัดสินใจ

WIP (Work in Progress)

Status

สถานะของผลการตัดสินใจ

Gathering information. We are starting with the plausible ends of the spectrum: the most-recommended older free tool (Nagios) and the most-recommended newer paid tool (New Relic).

https://sematext.com/blog/server-monitoring-tools/

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/metrics-monitors-alerts/index.md

Example 3 : CSS Framework

Summary

- **Issue**
- **Decision**
- **Status**

Details

- **Assumptions**
- **Constraints**
- **Positions**
- **Argument**
- **Implications**

Related

- **Related decisions**
- **Related requirements**
- **Related artifacts**
- **Related principles**

Notes

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/environment-variable-configuration/index.md

Example 3 : CSS Framework

Summary

Issue

บริบทที่เกี่ยวข้องกับความต้องการ

We want to use a CSS framework to create our web applications:

- We want user experience to be fast and reliable, on all popular browsers and screen sizes.
- We want rapid iteration on design, layout, UI/UX, etc.
- We want responsive applications.

Decision

ผลการตัดสินใจ

Decided on Bulma.

Status

สถานะของผลการตัดสินใจ

Decided on Bulma. Open to new CSS framework choices as they arrive.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/css-framework/index.md

Example 3 : CSS Framework

Details

Assumptions

มีสมมติฐานอะไรบ้าง เกี่ยวกับการตัดสินใจข้างต้น

We want to create web apps that are modern, fast, reliable, responsive, etc. Typical modern web apps are reducing/eliminating the use of jQuery because of multiple reasons:

- Modern JavaScript in phasing in many capabilities that jQuery has provided.
- jQuery's broad approach is to do direct DOM manipulation, which is an anti-pattern for modern JavaScript frameworks (e.g. React, Vue, Svelte)
- jQuery interferes with itself if it's loaded twice, etc.

Constraints

เงื่อนไขที่ต้องพิจารณาเพิ่มเติม

If we choose a CSS framework that uses jQuery, then we're stuck importing jQuery. For example, Semantic UI uses jQuery, and Tachyons does not.

If we choose a CSS framework that is minimal, then we forego framework components that we may want now or soon. For example, Semantic UI provides an image carousel, and Tachyons does not.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/css-framework/index.md

Example 3 : CSS Framework

Details

Positions

ทางเลือกที่เป็นไปได้อื่นๆ

We considered using no framework. This still seems viable, especially because CSS grid provides much of what we need for our project..

We considered many CSS frameworks using a quick shortlist triage: Bootstrap, Bulma, Foundation, Materialize, Semantic UI, Tachyons, etc. Our two selections for deeper review are Semantic UI (because it has the most-semantic approach) and Bulma (because it has the lightest-weight approach that provides the components we want now).

We considered Semantic UI. This provides many components, including ones we want for our project: tabs, grids, buttons, etc. We did a pilot with Semantic UI two ways: using typical CDN files and using NPM repos. …

Argument

เหตุผลที่เลือกใช้ Bulma

Specifically, Semantic UI seems to have a caution flag both in terms of technology (i.e. so many jQuery touchpoints) and also in terms of leadership (i.e., jQuery-free was a hard no, rather than attempting a roadmap, or continuous improvement, or donation fundraising, etc.).

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/css-framework/index.md

Example 3 : CSS Framework

Details

Implications

สิ่งที่ต้องพิจารณาเพิ่มเติมสืบเนื่องจากการตัดสินใจข้างต้น

If we find a good non-jQuery CSS framework, this is generally helpful and good overall.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/css-framework/index.md

Example 3 : CSS Framework

Related

สิ่งอื่นๆ ที่เกี่ยวข้องกับการตัดสินใจข้างต้น

Related decisions

The CSS framework we choose may affect testability.

Related requirements

We want to ship a purely-modern app fast.

We do not want to spend time working on older frameworks (esp. Semantic UI) using older dependencies (esp. jQuery).

Related artifacts

Affects all the typical HTML that will use the CSS.

Related principles

Easily reversible.

Need for speed.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/css-framework/index.md

Example 4 : Monorepo or multirepo

Summary

- **Issue**
- **Decision**
- **Status**
- **Group**

Details

- **Assumptions**
- **Constraints**
- **Positions**
- **Argument**
- **Implications**

Related

- **Related decisions**
- **Related requirements**
- **Related artifacts**
- **Related principles**

Notes

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/monorepo-vs-multirepo/index.md

Example 4 : Monorepo or polyrepo

Summary

Issue

บริบทที่เกี่ยวข้องกับความต้องการ

Our project involves developing three major categories of software:

(1) Front-end GUIs, (2) Middleware services, (3) Back-end servers

Our source code management (SCM) version control system (VCS) is git.

We need to choose how we use git to organize our code.

The top-level choice is to organize as a "monorepo" or "polyrepo" or "hybrid"

Decision

ผลการตัดสินใจ

Monorepo when an organization/team/project is relatively small, and rapid iteration is higher priority than sustaining stability.

Polyrepo when an organization/team/project is relatively large, and sustaining stability is higher priority than rapid iteration.

Status

สถานะของผลการตัดสินใจ

Decided.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/monorepo-vs-multirepo/index.md

Example 4 : Monorepo or polyrepo

Details

Assumptions

มีสมมติฐานอะไรบ้าง เกี่ยวกับการตัดสินใจข้างต้น

All the code that we're developing is for one organization's offerings, and not for the general public. i.e. the Broker-Dealer isn't aiming to have anything like general public volunteer developers.

Constraints

เงื่อนไขที่ต้องพิจารณาเพิ่มเติม

Constraints are well-documented at https://github.com/joelparkerhenderson/monorepo-vs-polyrepo

Positions

ทางเลือกที่เป็นไปได้อื่นๆ

We considered monorepos in the style of Google, Facebook, etc. We think any monorepo scaling issues are so far in the future that we will be able to leverage the same practices as Google and Facebook, by the time we need them.

We considered polyrepos in the style of typical Git open source projects, such as Google Android, Facebook React, etc. We think these are the best choice for general public participation

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/monorepo-vs-multirepo/index.md

Example 4 : Monorepo or polyrepo

Details

Argument

เหตุผลที่เลือกใช้ monorepo / polyrepo

When an organization/team/project is relatively small, we choose monorepo, because rapid iteration is significantly higher in priority than sustaining stability.

When an organization/team/project is relatively large, we choose polyrepo, because sustaining stability is significantly higher in priority than rapid iteration.

Implications

สิ่งที่ต้องพิจารณาเพิ่มเติมสืบเนื่องจากการตัดสินใจข้างต้น

If there's an existing pipeline for CI+CD, then we may need to adjust it for testing multiple projects within one repo. CI+CD could take more time for a full build for a monorepo, because CI+CD could build all the projects in the monorepo.

If an organization/team/project grows, then a monorepo will have scaling issues. Monorepo scaling issues may make it increasing valuable to transition to a polyrepo. Transition from monorepo to polyrepo is a significant devops task, and will need to be planned, managed, and programmed.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/monorepo-vs-multirepo/index.md

Example 4 : Monorepo or polyrepo

Related

สิ่งอื่นๆ ที่เกี่ยวข้องกับการตัดสินใจข้างต้น

Related decisions

We will create decisions for related tooling to manage monorepos (e.g. Google Bazel) and polyrepos (e.g. Lyft Refactorator).

Related requirements

We need to develop the CI+CD pipeline to work well with git.

Related artifacts

We expect the repo organization to have related artifacts for provisioning, configuration management, testing, and similar DevOps areas.

Related principles

Easily reversable. If the monorepo doesn't work in practice, or isn't wanted by leadership, it's simple to change to polyrepo.

Think big. Google and Facebook are very strong advocates for monorepos over polyrepos, because all the core offerings can be developed/tested/deployed in concert.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/monorepo-vs-multirepo/index.md

Example 5 : Programming languages

Summary

- **Issue**
- **Decision**
- **Status**
- **Group**

Details

- **Assumptions**
- **Constraints**
- **Positions**
- **Argument**
- **Implications**

Related

- **Related decisions**
- **Related requirements**
- **Related artifacts**
- **Related principles**

Notes

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/programming-languages/index.md

Example 5 : Programming languages

Summary

Issue

บริบทที่เกี่ยวข้องกับความต้องการ

We need to choose programming languages for our software. We have two major needs: a front-end programming language suitable for web applications, and a back-end programming language suitable for server applications.

Decision

ผลการตัดสินใจ

We are choosing TypeScript for the front-end.

We are choosing Rust for the back-end.

Status

สถานะของผลการตัดสินใจ

Decided. We are open to new alternatives as they arise.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/programming-languages/index.md

Example 5 : Programming languages

Details

มีสมมติฐานอะไรบ้าง เกี่ยวกับการตัดสินใจข้างต้น

Assumptions

The front-end applications are typical users and interactions, browsers and systems, development and deployment. The front-end applications is likely to evolve quickly.

The back-end applications are higher-than-typical goals for quality, especially provability, reliability, security, near-real-time, functional programming, especially for parallelization, multi-core processing, and memory safety.

We accept lower compile-time speeds in favor of compile-time safety and runtime speeds.

เงื่อนไขที่ต้องพิจารณาเพิ่มเติม

Constraints

We have a strong constraint on languages that are usable with major cloud provider services for functions, such as Amazon Lambda.

ทางเลือกที่เป็นไปได้อื่นๆ

Positions

We considered these languages: C, C++, Erlang, Go, Haskell, Java, Javascript, Kotlin, Python, Ruby, Rust, TypeScript, etc.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/programming-languages/index.md

Example 5 : Programming languages

Details

Arguments

เหตุผลที่เลือกใช้ TypeScript / Rust

Summary per language:

- C/C++ : rejected because of low safety; Rust can do nearly everything better.
- Go: excellent developer experience; excellent concurrency; but a track record of bad decisions that cripple the language.
- Java: excellent runtime; excellent ecosystem; sub-par developer experience.
- JavaScript: most popular language ever; most widespread ecosystem.
- Python: most popular language for systems administration; great analytics tooling; good web frameworks; but abandoned by Google in favor of Go.
- Ruby: best developer experience ever; best web frameworks; nicest community; but very slow; somewhat hard to package.
- Rust: best new language; zero-abstraction emphasis; concurrency emphasis; however relatively small ecosystem; and has deliberate limits on some kinds of compiler accelerations, e.g. ,direct memory access needs to be explicitly unsafe.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/programming-languages/index.md

Example 5 : Programming languages

Details

Arguments

สรุป เหตุผลที่เลือกใช้ TypeScript / Rust

Summary per language:

- Rust: best new language; zero-abstraction emphasis; concurrency emphasis; however relatively small ecosystem; and has deliberate limits on some kinds of compiler accelerations, e.g. ,direct memory access needs to be explicitly unsafe.
- TypeScript: adds types to JavaScript; great transpiler; growing developer emphasis on porting from JavaScript to TypeScript; strong backing from Microsoft.

We believe that our core decision is driven by two cross-cutting concerns:

For fastest runtime speed and tightest system access, we would choose JavaScript and C.

For close-to-fastest runtime speed and close-to-tightest system access, we choose TypeScript and Rust.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/programming-languages/index.md

Example 5 : Programming languages

Details

Implications

สิ่งที่ต้องพิจารณาเพิ่มเติมสืบเนื่องจากการ
ตัดสินใจข้างต้น

Front-end developers will need to learn TypeScript. This is likely an easy learning curve if the developer's primary experience is using JavaScript.

Back-end developers will need to learn Rust. This is likely a moderate learning curve if the developer's primary experience is using C/C++, and a hard learning curve if the developer's primary experience is using Java, Python, Ruby, or similar memory-managed languages.

TypeScript and Rust are both relatively new. This means that many tools do not yet have documentation for these languages. For example, the DevOps pipeline will need to be set up for these languages.

Compile times for TypeScript and Rust are quite slow.

IDE support for these languages is not yet ubiquitous and not yet first-class.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/programming-languages/index.md

Example 5 : Programming languages

Related

สิ่งอื่นๆ ที่เกี่ยวข้องกับการตัดสินใจข้างต้น

Related decisions

We will aim toward ecosystem choices that align with these languages.

For example, we want to choose an IDE that has good capabilities for these languages.

For example, for our front-end web framework, we are more-likely to decide on a framework that tends to aim toward TypeScript.

Related requirements

Our entire toolchain must support these languages.

Related artifacts

We expect we may export some secrets to environment variables.

Related principles

Measure twice, build once. We are prioritizing some safety over some speed.

Runtime is more valuable than compile time. We are prioritizing customer usage over developer usage.

https://github.com/joelparkerhenderson/architecture-decision-record/blob/main/examples/programming-languages/index.md

Decision record template by Jeff Tyree and Art Akerman

Summary

- Issue
- Decision
- Status
- Group

Details

- Assumptions
- Constraints
- Positions
- Argument
- Implications

Related

- Related decisions
- Related requirements
- Related artifacts
- Related principles

Notes

Decision record template by Michael Nygard

Title

Context

Describes the forces at play, including technological, political, social, and project local. It is simply describing facts.

Decision

Describes our responses to these forces. State it in full sentences, with active voices. “We will…”

Status

proposed : if the project stakeholders haven’t agreed with it yet.
accepted: if it is agreed.
deprecated or superseded : if it is replaced by a new ADR.

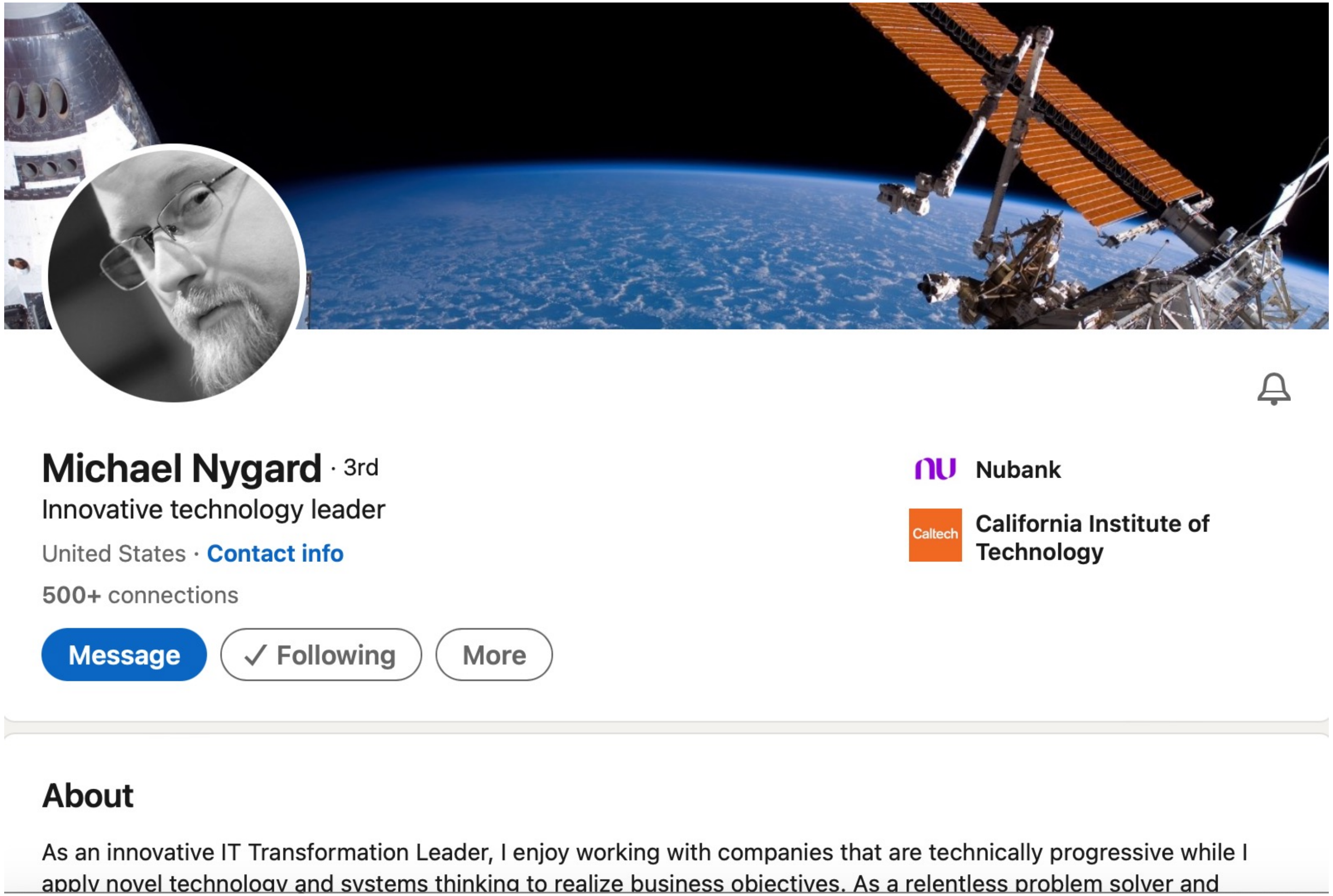
Consequences

describes the resulting context, after applying the decision. All consequences: positives

1-2 pages long; write an ADR as if it is a conversation with a future developer.

https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions

https://github.com/npryce/adr-tools



Michael Nygard · 3rd

Innovative technology leader

United States · **Contact info**

500+ connections

Nubank

California Institute of Technology

About

As an innovative IT Transformation Leader, I enjoy working with companies that are technically progressive while I apply novel technology and systems thinking to realize business objectives. As a relentless problem solver and

HTTPS://WWW.LINKEDIN.COM/IN/MTNYGARD/

Decision record template for **Alexandrian pattern**

Prologue (Summary)

In the context of (use case), facing (concern), we decided for (option) to achieve (quality) accepting (downside)

Discussion (Context)

Explains the force at play (technical, political, social, project).

Solution (Decision)

Explains how the decision will solve the problem

Consequences (Results)

Explains the results of the decision over the long term.
Did it work, not work, was changed, upgraded, etc.

Other examples

- Database systems for managing 50,000 genome data, which allows availability and consistency / query speed and data versioning
- Which message broker should we use to get messages; bus speed, location, temperature, PM2.5 from busses ? Which component can we use to get messages?
- Which could be used to protect backend APIs?, session-based cookies or token-based authentication such as OAuth, JWT, Random token with Memcached/ Redis?
- Database systems for network traffic management and monitoring, time series database, graph database?
- Which backend frameworks, e.g., spring boot, Django, Rails, JS, Flask, Kao, ASP.Net, Laravel, etc. that we will choose for our project?
- Server-side or client-side discovery / self registration or 3rd-party registration

WHEN SHOULD I WRITE AN ADR?

ALMOST ALWAYS

https://engineering.atspotify.com/2020/04/14/when-should-i-write-an-architecture-decision-record/

Activity in class

Write an ADR, documenting the AD for API design, choosing between REST API and gRPC.

Write an ADR, documenting the AD for selecting database system(s) for your term project.

Write an ADR, documenting the AD for implementing Zero Trust security.

References

https://github.com/joelparkerhenderson/architecture-decision-record

https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions