Software Architecture for Developers

Simon Brown
@simonbrown
Simon Brown
Independent consultant specialising in software architecture, plus the creator of the C4 model and Structurizr

@simonbrown
Five things every developer should know about software architecture

1. Software architecture isn't about big design up front
2. Every software team needs to consider software architecture
3. The software architecture role is about coding, coaching and collaboration
4. You don't need to use UML
5. A good software architecture enables agility

https://www.infoq.com/articles/architecture-five-things/
What is software architecture?
Structure

The definition of software in terms of its building blocks and their interactions
Vision

The process of architecting; making decisions based upon business goals, requirements and constraints, plus being able to communicate this to a team.
Enterprise Architecture
Structure and strategy across people, process and technology

System Architecture
High-level structure of a software system
software and infrastructure

Application Architecture
The internal structure of an application
As a noun, design is the named structure or behaviour of a system whose presence resolves ... a force on that system. A design thus represents one point in a potential decision space.

Grady Booch
All architecture is design, but not all design is architecture.

Grady Booch
Architecture represents the significant decisions, where significance is measured by cost of change.

Grady Booch
As architects, we define the significant decisions
What happens if a software development team doesn’t think about architecture?
Chaos

Big ball of mud, spaghetti code, inconsistent approaches to solving the same problems, quality attributes are ignored, deployment problems, maintenance issues, etc
Big design up front

Software Architecture Document

vs

No design up front
Big design up front is dumb. Doing no design up front is even dumber.

Dave Thomas
Software architecture helps us avoid chaos
Architectural drivers
Requirements
drive architecture
(use cases, user stories, features, etc)
Requirement

"a thing that is needed or wanted"

(this includes experiments and hypotheses too)
Don’t start designing software if you have no inputs
Quality attributes
(also known as non-functional requirements, cross-cutting concerns, service-level agreements, etc)
Create a **checklist** of quality attributes you regularly encounter:

- Performance
- Scalability
- Availability
- Security
- Disaster Recovery
- Accessibility
- Monitoring
- Management
- Audit
- Flexibility
- Extensibility
- Maintainability
- Interoperability
- Legal
- Regulatory
- Compliance
- i18n
- L10n
Understand how to capture, refine and challenge quality attributes
Software lives in the real world, and the real world has constraints.
Typical constraints include time and budget, technology, people and skills, politics, etc.
Constraints can sometimes be prioritised
Principles are selected by the team.
Development principles include coding conventions, naming guidelines, testing approaches, review practices, etc.
Architecture and design principles typically relate to modularity or crosscutting concerns

(architectural layering, separation of concerns, stateless vs stateful, rich vs anaemic domain, security, error handling, logging, etc)
Ensure you have a good understanding of the requirements, quality attributes, constraints and principles to create sufficient foundations
What about agile, and agility?
Agile is about moving fast, embracing change, releasing often, getting feedback, ...
Agile is about a mindset of continuous improvement
Inspect and adapt
Continuous attention to technical excellence and good design enhances agility.

Principle 9 of the Manifesto for Agile Software Development
A good architecture enables agility
A good architecture rarely happens through architecture-indifferent design
Monolithic
big ball of mud

Modular
monolith

Microservices

Distributed
big ball of mud

Number of deployment units

Modularity
Agility is a quality attribute
The software architecture role
Software development is not a relay sport
AaaS
Architecture as a Service
The software architecture role is about the “big picture” and, sometimes, this means stepping away from the code.
Do software development teams need "architects"? No
Do software development teams need technical leadership?

Yes
Every team needs technical leadership
Continuous technical leadership
(somebody needs to continuously steer the ship)
Should software architects write code?
Production code, prototypes, frameworks, foundations, code reviews, experimenting, etc
Don’t code all of the time!
There is often a tension between being “senior” and writing code...
Software architects should be master builders
Progress Toward an Engineering Discipline of Software

Mary Shaw
I am a senior developer. Recently, I was promoted to the position as architect. Could anyone please let me know which tools/software an architect should master/be familiar with. Thank you

Experience is important ... software architecture is not a rank!
Software architecture is not a “post-technical” career option!
Technology skills
(depth and breadth)
Good software architects are good software developers
The people designing software must understand technology ...
all decisions involve trade-offs
Soft skills

(leadership, communication, presentation, influencing, negotiation, collaboration, coaching and mentoring, motivation, facilitation, political, etc)
Collaborate or fail
Domain knowledge
(or the ability to learn quickly)
The software architecture role is multi-faceted (technology, soft skills, domain knowledge)
The software architecture role
(technical leadership, and responsible for the technical success of the project/product)

Architectural drivers
Understanding the goals; capturing, refining and challenging the requirements and constraints.

Technical risks
Identifying, mitigating and owning the technical risks to ensure that the architecture "works".

Designing software
Creating the technical strategy, vision and roadmap.

Technical leadership
Continuous technical leadership and ownership of the architecture throughout the software delivery.

Quality assurance
Introduction and adherence to standards, guidelines, principles, etc.
Software architects, solution architects, tech leads, principal engineers?
Hierarchies of architects, and central architecture groups?
Introducing control?
Avoiding chaos?
How much control do you need?
Different types of teams need different leadership styles
Pair architecting
Collaborative technical leadership is not easy
Complete all steps of part 1 at your online course URL

90 minutes for part 1
Visualising software architecture
Complete all steps of part 2 at your online course URL

10-15 minutes for part 2
Information is likely still stuck in your heads
This doesn’t make sense, but we’ll explain it.
• What is this shape/symbol?
• What is this line/arrow?
• What do the colours mean?
• What level of abstraction is shown?
• Which diagram do we read first?
Diagram:
- Logging Service
- Parameter Manager
- Risk Calculation
- Report Generator
- Data Import
- Auditing
- Validation
- TOS
- ROS
- NERDE Data
- Params
- Security
- Audit
<table>
<thead>
<tr>
<th>File Retriever</th>
<th>Scheduler</th>
<th>Auditing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference Archiver</td>
<td>Risk Assessment Processor</td>
<td>Risk Parameter Configuration</td>
</tr>
</tbody>
</table>

**FUNCTIONAL VIEW**
The producer-consumer conflict of software architecture diagrams

I don’t want to put technology choices on the diagrams...

Software design should be technology independent...

I wish these diagrams included technology choices...
If you’re going to use “boxes & lines”, at least do so in a **structured way**, using a **self-describing notation**
Moving fast in the same direction as a team requires good communication.
Do you use UML?
In my experience, optimistically, 1 out of 10 people use UML.
“Not everybody else on the team knows it.”
“I’m the only person on the team who knows it.”
“You’ll be seen as old.”
“You’ll be seen as old-fashioned.”
“The tooling sucks.”
“It’s too detailed.”
“It’s a very elaborate waste of time.”
“It’s not expected in agile.”
“The value is in the conversation.”
If you’re using UML, ArchiMate, SysML, BPML, DFDs, etc and it’s working ... keep doing that!
Who are the stakeholders that you need to communicate software architecture to; what information do they need?
There are many **different audiences** for diagrams and documentation, all with **different interests**

(software architects, software developers, operations and support staff, testers, Product Owners, project managers, Scrum Masters, users, management, business sponsors, potential customers, potential investors, …)
The primary use for diagrams and documentation is communication and learning.
To describe a software architecture, we use a model composed of multiple views or perspectives.

Architectural Blueprints - The “4+1” View Model of Software Architecture
Philippe Kruchten
The description of an architecture—the decisions made—can be organized around these four views, and then illustrated by a few selected use cases, or scenarios which become a fifth view. The architecture is in fact partially evolved from these scenarios as we will see later.

![Diagram of the “4+1” view model]

**Figure 1 — The “4+1” view model**
“Viewpoints and Perspectives”
Why is there a separation between the logical and development views?
Our architecture diagrams don’t match the code.
Model-code gap. Your architecture models and your source code will not show the same things. The difference between them is the model-code gap. Your architecture models include some abstract concepts, like components, that your programming language does not, but could. Beyond that, architecture models include intensional elements, like design decisions and constraints, that cannot be expressed in procedural source code at all.

Consequently, the relationship between the architecture model and source code is complicated. It is mostly a refinement relationship, where the extensional elements in the architecture model are refined into extensional elements in source code. This is shown in Figure 10.3. However, intensional elements are not refined into corresponding elements in source code.

Upon learning about the model-code gap, your first instinct may be to avoid it. But reflecting on the origins of the gap gives little hope of a general solution in the short term: architecture models help you reason about complexity and scale because they are abstract and intensional; source code executes on machines because it is concrete and extensional.
1 Introduction

Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers’ whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because the models are almost always inaccurate with respect to the system’s source.

Current reverse engineering systems derive high-level models from the source code. These derived models are useful because they are, by their very nature, accurate representations of the source. Although accurate, the models created by these reverse engineering systems may differ from the models sketched by engineers. An example of this is reported by Wong et al. [WTMS95].

Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers’ whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because the models are almost always inaccurate with respect to the system’s source.

Current reverse engineering systems derive high-level models from the source code. These derived models are useful because they are, by their very nature, accurate representations of the source. Although accurate, the models created by these reverse engineering systems may differ from the models sketched by engineers; an example of this is reported by Wong et al. [WTMS95].
We lack a common vocabulary to describe software architecture
Figure 48. Diagram of a basic circuit.
https://en.wikipedia.org/wiki/Component_diagram
Simple Definition of **component**

: one of the parts of something (such as a system or mixture): an important piece of something

Source: Merriam-Webster's Learner's Dictionary
Ubiquitous language
Would you code it that way?

(ensure that your diagrams reflect your implementation intent)
When drawing software architecture diagrams, think like a software developer.
If software developers created building architecture diagrams...
A common set of abstractions is more important than a common notation.
Abstractions
A **software system** is made up of one or more **containers** (web applications, mobile apps, desktop applications, databases, file systems, etc), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (e.g. classes, interfaces, objects, functions, etc).
Static structure diagrams
The C4 model for visualising software architecture

c4model.com
Diagrams are maps that help software developers navigate a large and/or complex codebase.
1. **System Context**
The system plus users and system dependencies.

2. **Containers**
The overall shape of the architecture and technology choices.

3. **Components**
Logical components and their interactions within a container.

4. **Code (e.g. classes)**
Component implementation details.
Example

(Internet Banking System)
Level 1

System Context diagram
Internet Banking System

Allows customers to view information about their bank accounts, and make payments.

System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Level 2

Container diagram
The container diagram shows the containers that reside inside the software system boundary.
Container diagram for Internet Banking System
The container diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Container diagram for Internet Banking System

The container diagram for the Internet Banking System.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Personal Banking Customer

A customer of the bank, with personal bank accounts.

-, Visits bigbank.com using [HTTPS]
-, Views account balances, and makes payments using
-, Views account balances, and makes payments using
-, Sends e-mails to

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

Container diagram for Internet Banking System

The container diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Level 3

Component diagram
The component diagram shows the components that reside inside an individual container.
**Single-Page Application**
(Container: Javascript and Angular)
Provides all of the Internet banking functionality to customers via their web browser.

**Mobile App**
(Container: Xamarin)
Provides a limited subset of the Internet banking functionality to customers via their mobile device.

---

**Database**
(Container: Oracle Database Schema)
Stores user registration information, hashed authentication credentials, access logs, etc.

**E-mail System**
(Software System)
The internal Microsoft Exchange e-mail system.

**Mainframe Banking System**
(Software System)
Stores all of the core banking information about customers, accounts, transactions, etc.
Component diagram for Internet Banking System - API Application

The component diagram for the API Application.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Component diagram for Internet Banking System - API Application

The component diagram for the API Application.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Level 4

Code diagram
The code level diagram shows the code elements that make up a component.
Notation
The C4 model is notation independent.
Titles

Short and meaningful, include the **diagram type**, numbered if diagram order is important; for example:

**System Context diagram** for Financial Risk System

[**System Context**] Financial Risk System
Visual consistency

Try to be consistent with notation and element positioning across diagrams
Acronyms

Be wary of using acronyms, especially those related to the business/domain that you work in.
Elements

Start with simple boxes containing the element name, type, technology (if appropriate) and a description/responsibilities
**Personal Banking Customer**

A customer of the bank, with personal bank accounts.

**Internet Banking System**

[Software System]

Allows customers to view information about their bank accounts, and make payments.

**API Application**

[Container: Java and Spring MVC]

Provides Internet banking functionality via a JSON/HTTPS API.

**Mainframe Banking System Facade**

[Component: Spring Bean]

A facade onto the mainframe banking system.
Lines

Favour uni-directional lines showing the most important dependencies or data flow, with an annotation to be explicit about the purpose of the line and direction.
Summarise the intent of the relationship
Summarise, yet be specific

Single Page Application [Container] Uses API Application [Container]

Single Page Application [Container] Makes API calls using API Application [Container]
Show both directions when the intents are different

Microservice A [Container]

Requests a list of customers from [JSON/HTTPS]

Sends new customers to [Kafka topic]

Microservice B [Container]
Beware of hiding the true story
Beware of hiding the true story
Beware of hiding the true story
Beware of hiding the true story

Microservice A
[Container]

Sends customer update messages to
[via Kafka topic X]

Microservice C
[Container]

Microservice B
[Container]

Sends order creation messages to
[via Kafka topic Y]

Microservice D
[Container]
Trade Data System [Software System] sends trade data to Financial Risk System [Software System].

Add more words to make the intent explicit.
Read the relationship out loud

Web Application [Container]

Reads **from** and writes **to**

Database [Container]
Key/legend

Explain shapes, line styles, colours, borders, acronyms, etc
... even if your notation seems obvious!
Container diagram for Internet Banking System

- **Internet Banking System** (Software System)
  - Container
  - Container, Database
  - Container, Mobile App
  - Container, Web Browser

- **Person**
  - Software System, Existing System

Relationships:
- Container connects to Internet Banking System
- Container, Database
- Container, Mobile App
- Container, Web Browser
- E-mail System
- Mainframe Banking System
- ADF Application
- Web Application
- Single-Page Application
- Mobile Application
- Database

Internet banking system diagram representation created by Simon Brown

Information may not be used without permission. (Software System)
Arrowheads

Be careful, using different arrowheads is very subtle; readers may miss them.
Use shape, colour and size to complement a diagram that already makes sense.
Be careful with icons
WordPress Hosting
How to run WordPress on AWS

WordPress is one of the world’s most popular web publishing platforms, being used to publish 27% of all websites, from personal blogs to some of the biggest news sites. This reference architecture simplifies the complexity of deploying a scalable and highly available WordPress site on AWS.

1. Static and dynamic content is delivered by Amazon CloudFront.
2. An Internet gateway allows communication between instances in your VPC and the Internet.
3. NAT gateways in each public subnet enable Amazon EC2 instances in private subnets (App & Data) to access the Internet.
4. Use an Application Load Balancer to distribute web traffic across an Auto Scaling Group of Amazon EC2 instances in multiple AZs.
5. Run your WordPress site using an Auto Scaling group of Amazon EC2 instances. Install the latest versions of WordPress, Apache web server, PHP 7, and OPcache and build an Amazon Machine Image that will be used by the Auto Scaling group launch configuration to launch new instances in the Auto Scaling group.
6. If database access patterns are read-heavy, consider using a WordPress plugin that takes advantage of a caching layer like Amazon ElastiCache (Memcached) in front of the database layer to cache frequently accessed data.
7. Simplify your database administration by running your database layer in Amazon RDS using either Aurora or MySQL.
8. Amazon EC2 instances access shared WordPress data in an Amazon EFS file system using Mount Targets in each AZ in your VPC.
9. Use Amazon EFS, a simple, highly available, and scalable network file system so WordPress instances have access to your shared, unstructured WordPress data, like php files, config, themes, plugins, etc.
Container diagram for Internet Banking System

The container diagram for the Internet Banking System.

Workspace last modified: Sat Jan 11 2020 14:47:20 GMT+0000 (Greenwich Mean Time)
Use icons to supplement text, not replace it
Increase the *readability* of software architecture diagrams, so they can *stand alone*.
Any narrative should **complement** the diagram rather than explain it.
# General

<table>
<thead>
<tr>
<th>Question</th>
<th>Option 1</th>
<th>Option 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does the diagram have a title?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Do you understand what the diagram type is?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Do you understand what the diagram scope is?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Does the diagram have a key/legend?</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Abstractions first, notation second

Ensure that your team has a ubiquitous language to describe software architecture
Complete all steps of parts 3 & 4 at your online course URL

~30 minutes for part 3 | ~60 minutes for part 4
Designing software is where the complexity should be, not communicating it!
The diagrams should spark meaningful questions
No

“What does that arrow mean?”
“Why are some boxes red?”
“Is that a Java application?”
“Is that a monolithic application, or a collection of microservices?”
“How do the users get their reports?”
Yes

“What protocol are your two Java applications using to communicate with each other?”

“Why do you have two separate C# applications instead of one?”

“Why are you using MongoDB?”

“Why are you using MySQL when our standard is Oracle?”

“Should we really build new applications with .NET Framework rather than .NET Core?”
Richer diagrams lead to richer design discussions.
Richer diagrams lead to **better communication**, making it easier to scale teams.
Similar levels of abstraction provide a way to easily compare solutions.
System landscape diagrams
System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
System Landscape diagram for Big Bank plc

The system landscape diagram for Big Bank plc.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Runtime/behavioural diagrams
Static structure diagrams are very useful, but they don’t tell the whole story
Dynamic diagram for API Application

Summarises how the sign in feature works in the single-page application.
Use dynamic diagrams to describe patterns or complex interactions
Deployment diagrams
Deployment is about the mapping of containers to infrastructure.
Deployment Node

Physical infrastructure (a physical server or device), virtualised infrastructure (IaaS, PaaS, a virtual machine), containerised infrastructure (a Docker container), database server, Java EE web/application server, Microsoft IIS, etc
A deployment node can contain other deployment nodes or software system/container instances.
Deployment diagram for Internet Banking System - Development

An example development deployment scenario for the Internet Banking System.

Tuesday, 4 August 2020, 09:32 GMT+01:00
Infrastructure Node

Routers, firewalls, load balancers, DNS providers, edge caches, etc
What tooling do you recommend?
# Tooling

For design sessions, you might find a whiteboard or flip chart paper better for collaboration, and iterating quickly. For long-lived documentation, the following modelling and diagramming tools can help create software architecture diagrams based upon the C4 model.

## Modelling tools

**Structurizr**
Structurizr is a collection of tools to create software architecture diagrams and documentation based upon the C4 model. Structurizr was started in 2014 by Simon Brown (creator of the C4 model), and has grown into a community of tooling, much of which is open source.

Structurizr is unique in that it supports diagrams as code (Java, Clojure, .NET, TypeScript, PHP, Python, Go or text (DSL), or YAML) via a number of different authoring methods, with it being possible to render diagrams using a number of different tools (Structurizr, Cloud, Mermaid, WebSequenceDiagrams, etc.).

### Archi
Archi provides a way for you to create C4 model diagrams with ArchiMate. See C4 Model, Architecture Viewpoint and Archi 4.7 for more details.

### Sparx Enterprise Architect
Sparx Enterprise Architect is a comprehensive software architecture tool that allows you to create diagrams using pre-built shapes.

### MooD
MooD has support for the C4 model via a set of blueprints.

### Astah
Astah has support for the C4 model via a C4 model plugin.

## Diagramming tools

### PlantUML
There are a number of extensions for PlantUML to assist in the creation of C4 model diagrams:
- C4-PlantUML by Riccardo Nipol
- C4-PlantUML by Tawas Klenkous
- c4Builder by Victor Lupu
- plantuml4bis by Thibault Morin

You can also create C4-PlantUML diagrams using C# code via the C4Sharp library.

### diagrams.net
Diagrams.net includes support for the C4 model, and there are also a number of plugins that allow you to create diagrams using pre-built shapes:
- c4-draw.js by Chris Kamiński
- c4-draw.js by Tobias Hochtgruber
- EasyC by Marcin Stawiński

### OmniGraffle
Dennis Laumen has created a C4 model stencil for OmniGraffle, that allows you to create diagrams using pre-built shapes.

## Microsoft Visio

*Shake* has created a C4 model template for Microsoft Visio, that allows you to create diagrams using pre-built shapes.

### yEd
Forhat Karimb has created some C4 model shapes for yEd.
Documenting software architecture
Working software over comprehensive documentation

Manifesto for Agile Software Development
The code doesn’t tell the whole story
Useful information spread across hundreds of pages; rarely read or updated
Travel Guidebook

/maps, points of interest, sights, itineraries, history, culture, practical information, etc/
Software Guidebook

(maps, points of interest, sights, itineraries, history, culture, practical information, etc)
The scope is a single software system
Describe what you can’t get from the code
Documentation should be constantly evolving
Context

A system context diagram, plus some narrative text to “set the scene”
Functional Overview

An overview of the software system, perhaps including wireframes, UI mockups/screenshots, workflow diagrams, business process diagrams, etc.
Quality Attributes

A list of the quality attributes (non-functional requirements; e.g. performance, scalability, security, etc)
Constraints

A list of the environmental constraints (e.g. timescales, budget, technology, team size/skills, etc)
Principles

A list of the development and architecture principles (e.g. coding conventions, separation of concerns, patterns, etc)
Software Architecture

A description of the software architecture, including static structure (e.g. containers and components) and dynamic/runtime behaviour.
Code

A description of important or complicated component implementation details, patterns, frameworks, etc.
Data

Data models, entity relationship diagrams, security, data volumes, archiving strategies, backup strategies, etc
Infrastructure Architecture

A description of the infrastructure available to run the software system
Deployment

The mapping of software (e.g. containers) to infrastructure
Development Environment

A description of how a new developer gets started
Operation and Support

An overview of how the software system is operated, supported, monitored, etc
Decision Log

A log of the major decisions made; e.g. as free format text or a collection of “Architecture Decision Records”
Title These documents have names that are short noun phrases. For example, "ADR 1: Deployment on Ruby on Rails 3.0.10" or "ADR 9: LDAP for Multitenant Integration"

Context This section describes the forces at play, including technological, political, social, and project local. These forces are probably in tension, and should be called out as such. The language in this section is value-neutral. It is simply describing facts.

Decision This section describes our response to these forces. It is stated in full sentences, with active voice. "We will ..."

Status A decision may be "proposed" if the project stakeholders haven't agreed with it yet, or "accepted" once it is agreed. If a later ADR changes or reverses a decision, it may be marked as "deprecated" or "superseded" with a reference to its replacement.

Consequences This section describes the resulting context, after applying the decision. All consequences should be listed here, not just the "positive" ones. A particular decision may have positive, negative, and neutral consequences, but all of them affect the team and project in the future.

“Architecture Decision Record”
A short description of an architecturally significant decision

http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions (Michael Nygard)
Immutable vs mutable ADRs?
This is a starting point; add and remove sections as necessary.

**Context**
A system context diagram, plus some narrative text to “set the scene”.

**Software Architecture**
A description of the software architecture, including static structure (e.g. containers and components) and dynamic/ runtime behaviour.

**Code**
A description of important or complicated component implementation details, patterns, frameworks, etc.

**Data**
Data models, entity relationship diagrams, security, data volumes, archiving strategies, backup strategies, etc.

**Constraints**
A list of the environmental constraints (e.g. timescales, budget, technology, team size/skills, etc).

**Principles**
A list of the development and architecture principles (e.g. coding conventions, separation of concerns, patterns, etc).

**Development Environment**
A description of how a new developer gets started.

**Deployment**
The mapping of software (e.g. containers) to infrastructure.

**Operation and Support**
An overview of how the software system is operated, supported, monitored, etc.

**Decision Log**
A log of the major decisions made; e.g. as free format text or a collection of “Architecture Decision Records”.

---

**Functional Overview**
An overview of the software system; perhaps including wireframes, UI mockups, screenshots, workflow diagrams, business process diagrams, etc.

**Quality Attributes**
A list of the quality attributes (non-functional requirements; e.g. performance, scalability, security, etc).

---

**Infrastructure Architecture**
A description of the infrastructure available to run the software system.
arc42 Template Overview

arc42 is a template for architecture communication and documentation.

arc42 answers the following two questions in a pragmatic way, but can be tailored to your specific needs:

- *What* should we document/communicate about our architecture?
- *How* should we document/communicate?

1. Introduction and Goals

Short description of the requirements, driving forces, extract (or abstract) of requirements. Top three (max five) quality goals for the architecture which have highest priority for the major stakeholders. A table of important stakeholders with their expectation regarding architecture.

Read More
Documentation format?

Microsoft Word, Microsoft SharePoint, Atlassian Confluence, Markdown or Asciidoc, etc
How long?

Something I can read in 1-2 hours;
a good starting point for exploring the code
How do you keep software architecture documentation up to date?
C4 model diagrams
+ software guidebook/arc42
+ architecture decision records
Software architecture in the delivery process
Big design up front

Software Architecture Document

vs

No design up front
Big design up front is dumb.
Doing no design up front is even dumber.

Dave Thomas
Evolutionary architecture
I’m referring to **technical design** rather than product design.
How much up front design should you do?

0% - 100%
it depends
Sometimes requirements are known, and sometimes they aren’t
(enterprise software development vs product companies and startups)
just enough
Up front design is not necessarily about creating a perfect end-state or complete architecture.
Iteration (via prototyping and experimentation) is great for product design but...

you don’t just “build the car”
Evolutionary Design
Beginning With A Primitive Whole
Evolutionary Design
Beginning With A Primitive Whole
We’re not trying to make every decision
I think there is a role for a broad starting point architecture. Such things as stating early on how to layer the application, how you'll interact with the database (if you need one), what approach to use to handle the web server.

Martin Fowler
https://martinfowler.com/articles/designDead.html
A starting point adds value
If you don’t **engage** in the problem, you end up with a very simplified and superficial view of the solution.
Part of the design activity is about discovering “unknown unknowns”
The typical s-curve of learning

- Slow initial progress
- Accelerated learning
- Plateau
1. Is that what we’re going to **build**?

2. Is it going to **work**?
Diagrams are a visual checklist for design decisions
System Context diagram

What is the scope of the software system we're building?
Who is using it? What are they doing?
What system integrations does it need to support?
Financial Risk System: Context Diagram

**User**
- Business Users (Person)
  - A customer that accesses risk reports
- Business Users with Configuration Access (Person)
  - A user that can modify risk calculation parameters

**Interaction**
- Delivers link to report to
- Access reports using
- Sends report links through
- Uses data from
- Use data from
- Changes configure for

**Proposed System**
- Email (Software System)
  - Sends emails to business users

**Pre-existing System**
- Central Monitoring System (Software System)
  - Monitors health of all services

**Software System**
- Trade Data System
  - Provides data about all trades by the bank
- Reference Data System
  - Provides reference data about counterparties

**Financial Risk System (Software System)**
- Takes trade data and reference data and calculates risk; generates risk report and makes it available to business users.
Container diagram
What are the major technology building blocks?
What are their responsibilities?
How do they communicate?
Understand the structure and create a shared vision
1. Is that what we’re going to **build**?

2. Is it going to **work**?
Teams need to explicitly manage technical risk
An example timeline from “Beyond Retrospectives”
Linda Rising, GOTO Aarhus 2011

Problems with new technology
Identify and mitigate your highest priority risks
<table>
<thead>
<tr>
<th>Impact</th>
<th>Low 1</th>
<th>Medium 2</th>
<th>High 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low 1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Medium 2</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>High 3</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>
The software architecture role should own the technical risks
Architecturally significant?

Costly to change, complex or new
Like estimates, risks are subjective.
Visual and collaborative “games”
Risk-storming
A visual and collaborative technique for identifying risk
Threat modelling

(STRIDE, LINDDUN, Attack Trees, etc)
Diagram 1 of 1

Mobile App
[Container: Xamarin]
Provides a limited subset of the internet banking functionality to customers via their mobile device.

Customer's mobile device
[Deployment Node: Apple iOS or Android]

Makes API calls to
[JSON/HTTPS]

API Application
[Container: Java and Spring MVC]
Provides internet banking functionality via a JSON/HTTPS API.

Apache Tomcat
[Deployment Node: Apache Tomcat 8.x]

bigbank-api***
[Deployment Node: Ubuntu 16.04 LTS]

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Oracle - Primary
[Deployment Node: Oracle 12c]
bigbank-db01
[Deployment Node: Ubuntu 16.04 LTS]

Replicates data to

Single-Page Application
[Container: JavaScript and Angular]
Delivers to the customer's web browser.

Web Application
[Container: Java and Spring MVC]
Base your architecture on requirements, travel light and prove your architecture with concrete experiments.

Agile Architecture: Strategies for Scaling Agile Development
ScottAmbler
Concrete experiment

Proof of concept, prototype, spike, tracer, vertical slice, walking skeleton, executable reference architecture, ...
Just enough up front design to create firm and sufficient foundations
Technical priorities vs product priorities?
The product owner(s) and software architect(s) are peers

(“Architecture Owner” is another term you can use)
How much up front design should you do?
“I’m good with maybe a day for a one-year effort.”
Up front design is an iterative and incremental process; stop when:

- You understand the significant architectural drivers (requirements, quality attributes, constraints).
- You understand the context and scope of what you’re building.
- You understand the significant design decisions (i.e. technology, modularity, etc).
- You have a way to communicate your technical vision to other people.
- You are confident that your design satisfies the key architectural drivers.
- You have identified, and are comfortable with, the risks associated with building the software.

Techniques: Workshops, interviews, Event Storming, Impact Mapping, domain modelling, OOAD, CRC, DDD, architecture reviews, ATAM, architecture dry runs, Risk-storming, concrete experiments, C4 model, ADRs, etc.
How long?

Hours, days or weeks ... not months or years
Some Design Up Front
+ Evolutionary Design
Some up front design to create a starting point and direction for further evolutionary design
Estimates?

Features vs components
we used to do things like this, it worked but we stopped doing it when we became agile
Adopt an agile mindset

Choose a starting point and continuously improve to discover what works for you
Thank you!