Solution Architecture Cheat Sheet

Adrian Kearns

Geekle Meet-Up, Feb-2021

Quick Intro

Me:

- Played with Lego since 1979
- Developer since 2000
- Solution Architect since 2008
- Consulting Architect since 2013

Based in New Zealand
International experience in Singapore and Australia
Still codes for fun

https://www.linkedin.com/in/adriankearns
https://morphological.wordpress.com

Works at Middleware New Zealand
https://www.middleware.co.nz

middleware nz

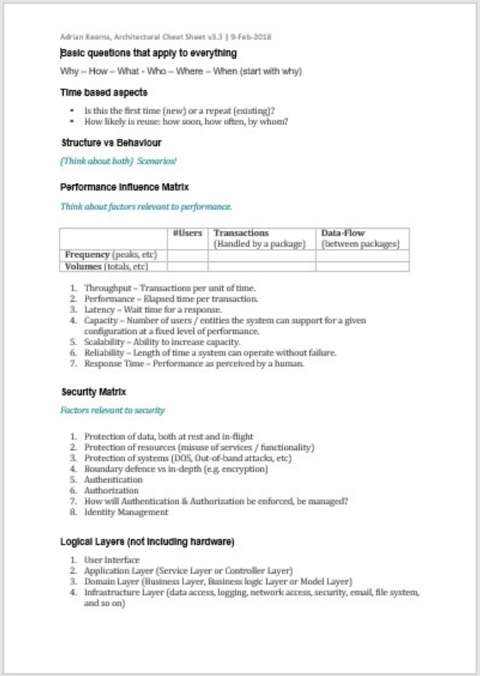
Agenda & Introduction

1. The Cheat Sheet: What is it?
2. State of Mind
3. Cheat Sheet Development Process
4. Using Your Cheat Sheet
5. Use as an Internal Training Exercise
6. Substance: Areas of knowledge

The Cheat Sheet: What is it?

A prompt / reminder
of the many concepts related to solution architecture,
both the doing of it (methods)
and the substance of it (an architecture).

Must be: easy to use, clear, fast to refer to, portable.



Adrian Kearns, Architectural Cheat Sheet v3.3 | 9-Feb-2018

Basic questions that apply to everything

Why – How – What - Who – Where – When (start with why)

Time based aspects

- Is this the first time (new) or a repeat (existing)?
- How likely is reuse: how soon, how often, by whom?

Structure vs Behaviour

(Think about both) Scenarios!

Performance Influence Matrix

Think about factors relevant to performance.

	#Users	Transactions (Handled by a package)	Data-Flow (between packages)
Frequency (peaks, etc)			
Volumes (totals, etc)			

1. Throughput – Transactions per unit of time.
2. Performance – Elapsed time per transaction.
3. Latency – Wait time for a response.
4. Capacity – Number of users / entities the system can support for a given configuration at a fixed level of performance.
5. Scalability – Ability to increase capacity.
6. Reliability – Length of time a system can operate without failure.
7. Response Time – Performance as perceived by a human.

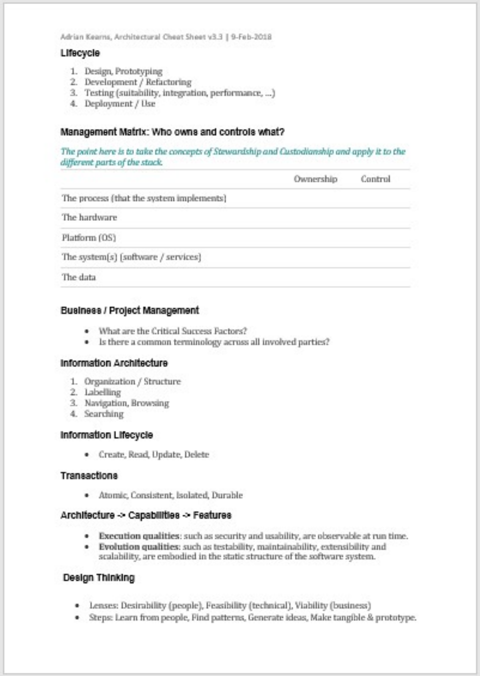
Security Matrix

Factors relevant to security

1. Protection of data, both at rest and in-flight
2. Protection of resources (misuse of services / functionality)
3. Protection of systems (DOS, Out-of-band attacks, etc)
4. Boundary defence vs in-depth (e.g. encryption)
5. Authentication
6. Authorization
7. How will Authentication & Authorization be enforced, be managed?
8. Identity Management

Logical Layers (not including hardware)

1. User Interface
2. Application Layer (Service Layer or Controller Layer)
3. Domain Layer (Business Layer, Business logic Layer or Model Layer)
4. Infrastructure Layer (data access, logging, network access, security, email, file system, and so on)



Adrian Kearns, Architectural Cheat Sheet v3.3 | 9-Feb-2018

Lifecycle

1. Design, Prototyping
2. Development / Refactoring
3. Testing (suitability, integration, performance, ...)
4. Deployment / Use

Management Matrix: Who owns and controls what?

The point here is to take the concepts of Stewardship and Custodianship and apply it to the different parts of the stack.

	Ownership	Control
The process (that the system implements)		
The hardware		
Platform (OS)		
The system(s) (software / services)		
The data		

Business / Project Management

- What are the Critical Success Factors?
- Is there a common terminology across all involved parties?

Information Architecture

1. Organization / Structure
2. Labelling
3. Navigation, Browsing
4. Searching

Information Lifecycle

- Create, Read, Update, Delete

Transactions

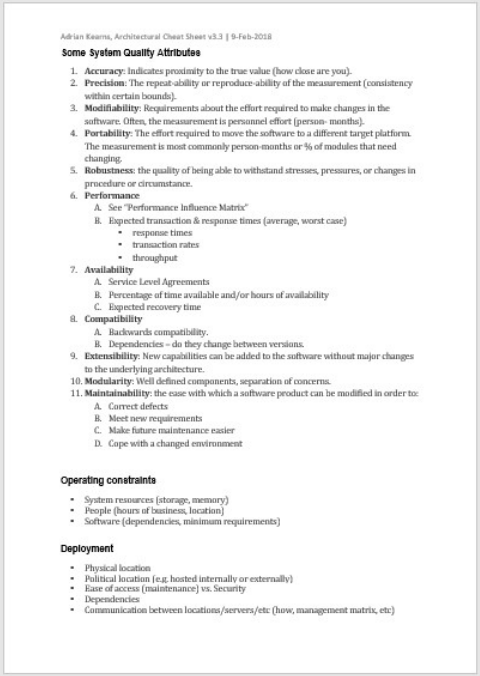
- Atomic, Consistent, Isolated, Durable

Architecture -> Capabilities -> Features

- **Execution qualities**: such as security and usability, are observable at run time.
- **Evolution qualities**: such as testability, maintainability, extensibility and scalability, are embodied in the static structure of the software system.

Design Thinking

- Lenses: Desirability (people), Feasibility (technical), Viability (business)
- Steps: Learn from people, Find patterns, Generate ideas, Make tangible & prototype.



Adrian Kearns, Architectural Cheat Sheet v3.3 | 9-Feb-2018

Some System Quality Attributes

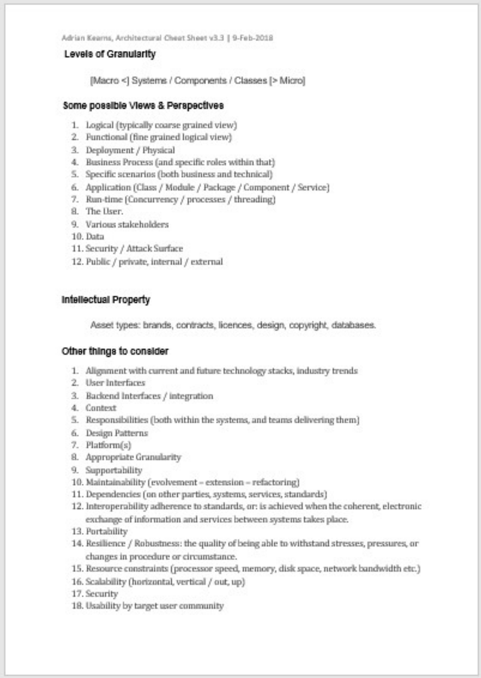
1. **Accuracy**: Indicates proximity to the true value (how close are you).
2. **Precision**: The repeat-ability or reproduce-ability of the measurement (consistency within certain bounds).
3. **Modifiability**: Requirements about the effort required to make changes in the software. Often, the measurement is personnel effort (person- months).
4. **Portability**: The effort required to move the software to a different target platform. The measurement is most commonly person-months or % of modules that need changing.
5. **Robustness**: the quality of being able to withstand stresses, pressures, or changes in procedure or circumstance.
6. **Performance**
   A. See "Performance Influence Matrix"
   B. Expected transaction & response times (average, worst case)
      - response times
      - transaction rates
      - throughput
7. **Availability**
   A. Service Level Agreements
   B. Percentage of time available and/or hours of availability
   C. Expected recovery time
8. **Compatibility**
   A. Backwards compatibility.
   B. Dependencies – do they change between versions.
9. **Extensibility**: New capabilities can be added to the software without major changes to the underlying architecture.
10. **Modularity**: Well defined components, separation of concerns.
11. **Maintainability**: the ease with which a software product can be modified in order to:
    A. Correct defects
    B. Meet new requirements
    C. Make future maintenance easier
    D. Cope with a changed environment

Operating constraints

- System resources (storage, memory)
- People (hours of business, location)
- Software (dependencies, minimum requirements)

Deployment

- Physical location
- Political location (e.g. hosted internally or externally)
- Ease of access (maintenance) vs. Security
- Dependencies
- Communication between locations/servers/etc (how, management matrix, etc)



Adrian Kearns, Architectural Cheat Sheet v3.3 | 9-Feb-2018

Levels of Granularity

[Macro <] Systems / Components / Classes [> Micro]

Some possible Views & Perspectives

1. Logical (typically coarse grained view)
2. Functional (fine grained logical view)
3. Deployment / Physical
4. Business Process (and specific roles within that)
5. Specific scenarios (both business and technical)
6. Application (Class / Module / Package / Component / Service)
7. Run-time (Concurrency / processes / threading)
8. The User.
9. Various stakeholders
10. Data
11. Security / Attack Surface
12. Public / private, internal / external

Intellectual Property

Asset types: brands, contracts, licences, design, copyright, databases.

Other things to consider

1. Alignment with current and future technology stacks, industry trends
2. User Interfaces
3. Backend Interfaces / integration
4. Context
5. Responsibilities (both within the systems, and teams delivering them)
6. Design Patterns
7. Platform(s)
8. Appropriate Granularity
9. Supportability
10. Maintainability (evolvement – extension – refactoring)
11. Dependencies (on other parties, systems, services, standards)
12. Interoperability adherence to standards, or: is achieved when the coherent, electronic exchange of information and services between systems takes place.
13. Portability
14. Resilience / Robustness: the quality of being able to withstand stresses, pressures, or changes in procedure or circumstance.
15. Resource constraints (processor speed, memory, disk space, network bandwidth etc.)
16. Scalability (horizontal, vertical / out, up)
17. Security
18. Usability by target user community

State of Mind

How effective the “cheat sheet” approach is for you may depend on how you like to learn.

I suspect I’m an autodidact & polymath. So self-guided learning works for me.

I don’t think this means you have to be an autodidact & polymath too.

State of Mind

We all have a world view, a mental model of how the world works.

As you become an architect you develop a related mental model of architecture.

For me, my mental model of architecture would have developed in parallel with the cheat sheet – each supporting the other.

Key Takeaway: the value is not just in the end product, but the journey as well.

Cheat Sheet Development Process

1. Find knowledge, information and wisdom. Use multiple sources.
2. Develop the skeleton / foundations of your mental model: breadth.
3. Critically evaluate source and content, cross reference, challenge & interpret. Discuss with others.
4. Distil things down to the core ideas; seek elegance and truth.
5. Throw it into a cheat sheet, try it out, see if it helps you.
6. Incrementally improve, but maybe don’t go overbroad.

Bottom Line: it is both a process and tool to help you,
if it’s not helping then maybe you learn in a different way.

A note on Information Sources

1. Wikipedia is not a place bad start – if nothing else it can be good for discovering the breath of a domain or who leading figures are.
   - It’s also great for “laws”, fundamentals and classics, like: Fitts’s Law, Hanlon’s razor, cognitive biases, Frederick Brooks' ‘The Mythical Man-Month’.
2. Industry bodies / standards can be great – either as sources or links to sources.
   - E.g. RFCs (e.g. HTTP), OSI, Cloud Native Computing Foundation, OpenAPI org, OWASP, CWE.
3. Existing bodies of knowledge can be great but might be an easy option. These can be great for validating what you think you know.
   - E.g. CMU SEI, IASA ITABoK, The Open Group, TOGAF, SABSA, Bredemeyer Consulting: Resources for Software and Systems Architects.
4. People / experts in the field – their blogs, conferences, presentations etc. Be sure to validate them as a source. Academic papers can also be a good source.
5. Using vendors as sources can be useful, but its recommended you use multiple vendors if possible. Be really aware of their commercial bias, product focus, etc.

Using Your Cheat Sheet

As a prompt:

1. Review it before key meetings.
2. Light-skim it immediately before key meetings.
3. During meetings.
4. In general, and whilst reviewing work.

So that you:

1. Are better prepared in advance.
2. Can quickly switch context
3. Can be reminded of things you might be forgetting, especially if the topic of conversation changes.
4. Are confident, because you know you're less likely to missing anything important.

Use as an Internal Training Exercise

I’ve put together a:
“Architecture Cheat Sheet - Internal Training Exercise Outline”.

Feel free to try it out. Use this deck to help you.

I’ll post it all on my blog.

Substance: Areas of knowledge

1. Meta / the universe (why, time, etc)
2. Design thinking
3. Logical layers, levels of granularity
   - Layers vs tiers.
4. Views & Perspectives (formal and informal)
5. Solution Lifecycle / SDLC
6. Governance
7. Business & project management concerns
8. Information architecture
   - Information lifecycle (CRUD)
9. System Quality Attributes & NFRs
10. Security
11. Performance
12. Deployment
13. Operation & support
14. Existing models/concepts, e.g.:
    - Transactions (ACID)
    - OSI Model
15. Anything else, e.g.:
    - Licensing
    - Intellectual property