Tech Lead Masterclass

From Maker to Multiplier
with Patrick Kua (@patkua)

Name:	______________________________
Date:	______________________________

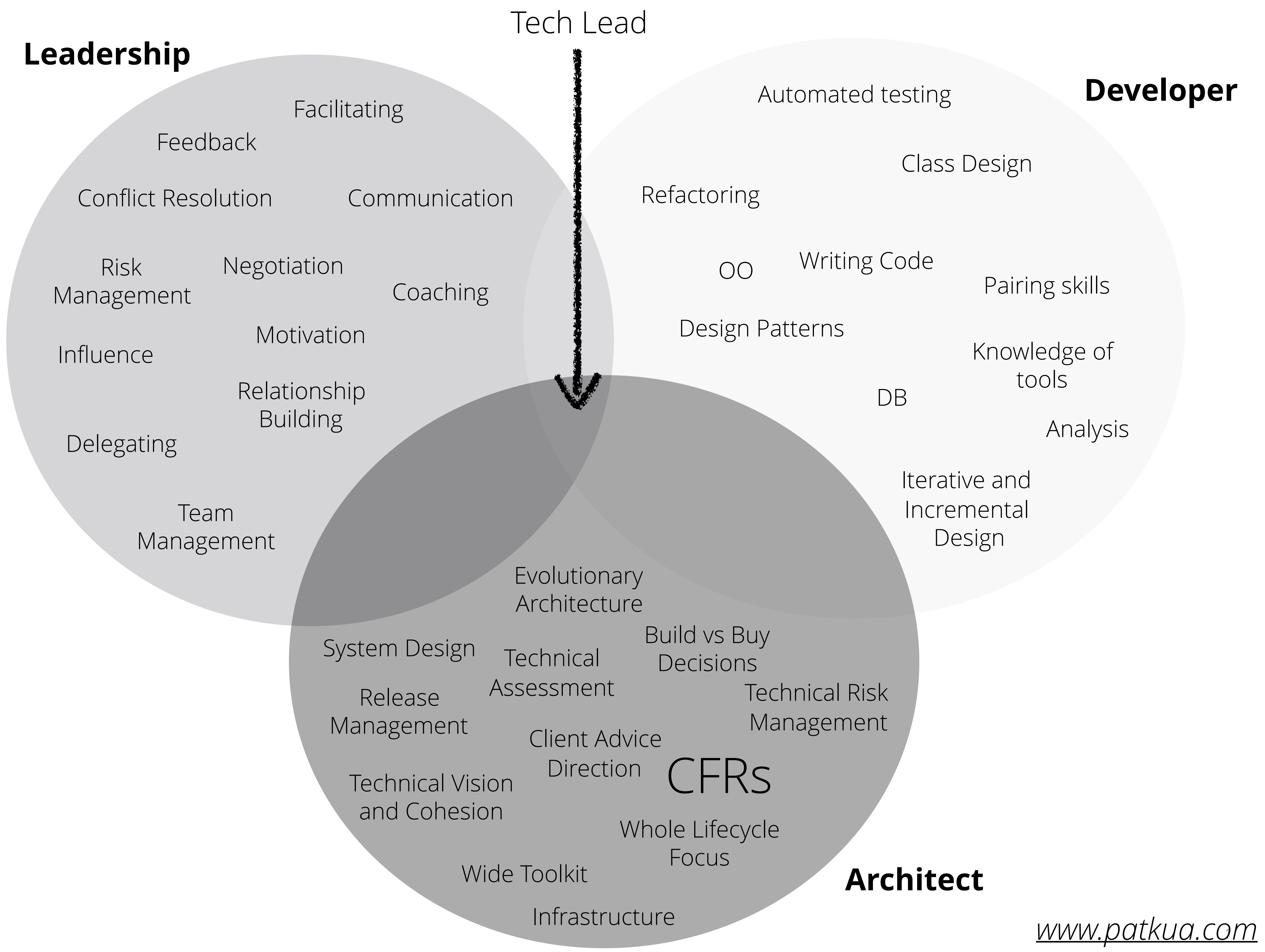
Draw face here

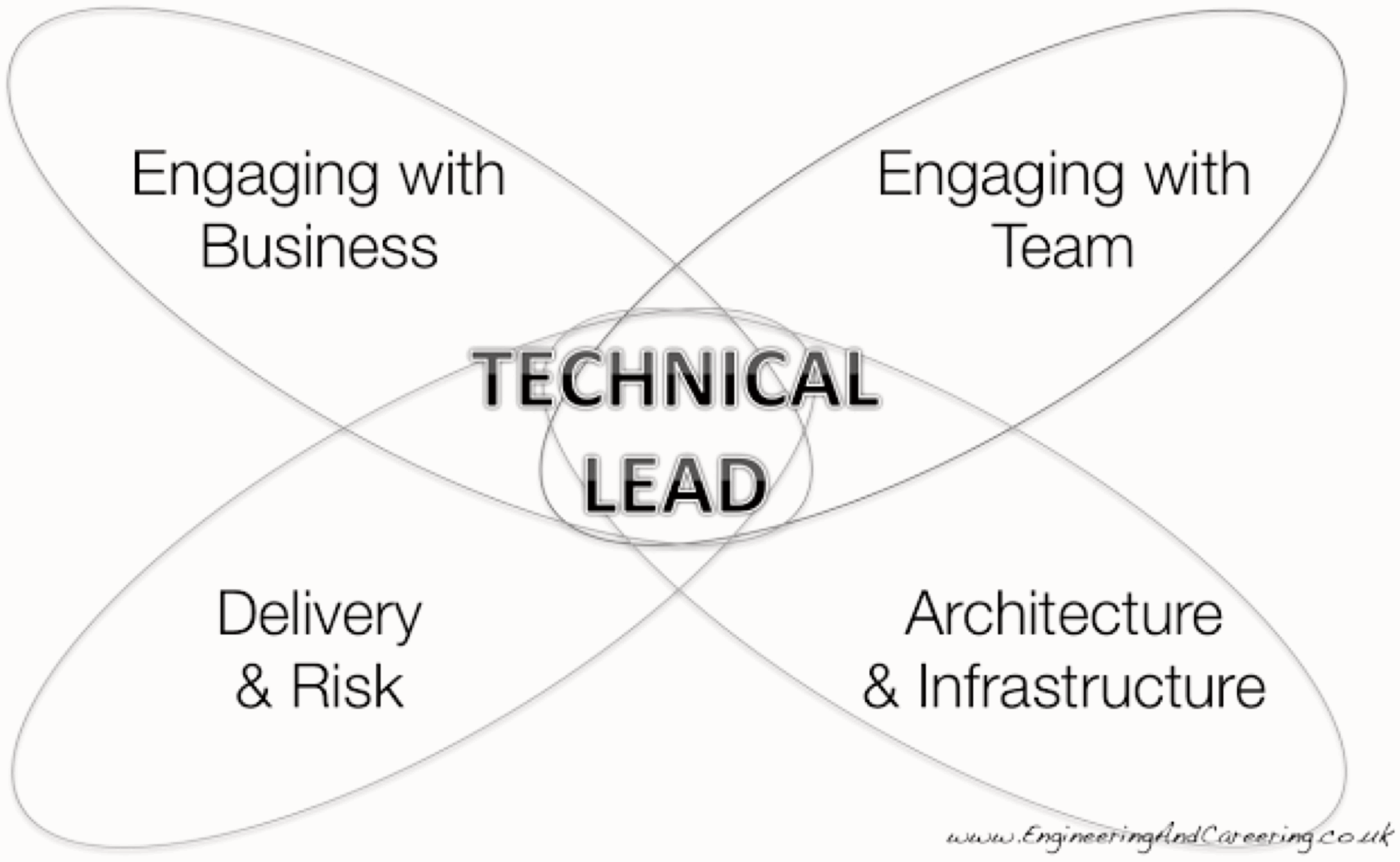
Background:

Name:

Tech Lead Superpower:

Favourite Food/Drink:


Tech Lead
Leadership
Facilitating
Feedback
Conflict Resolution
Communication
Risk Management
Negotiation
Coaching
Motivation
Influence
Relationship Building
Delegating
Team Management
Developer
Automated testing
Class Design
Refactoring
OO
Writing Code
Pairing skills
Design Patterns
Knowledge of tools
DB
Analysis
Iterative and Incremental Design
Architect
Evolutionary Architecture
System Design
Technical Assessment
Build vs Buy Decisions
Technical Risk Management
Release Management
Client Advice Direction
CFRs
Technical Vision and Cohesion
Whole Lifecycle Focus
Wide Toolkit
Infrastructure
www.patkua.com


Engaging with Business
Engaging with Team
TECHNICAL LEAD
Delivery & Risk
Architecture & Infrastructure
www.EngineeringAndCareering.co.uk

Tech Lead Self-Assessment

Use this tool to help you:

- Have a clearer understanding of where your strengths lie
- Have a better idea about where you aspire to be
- Guide your self-development process

This is NOT a test, but a tool to help you build more self-awareness.

Instructions

1. Familiarise yourself with the levels below
2. Read the description of each of the attributes
3. Use two different colours (e.g. black/red) and for each characteristic, mark:
    a. Where you believe you are now
    b. Where you would like to be
4. Connect the dots for each colour

Self-Assessment Rating

- **Level 1: Huh?** – Don't think about it, never done it or not even aware of it.
- **Level 2: Unsure** – Have done it may be once, not confident in doing this without support. This is very much a stretch activity for me.
- **Level 3: Can (just) do it** – I am comfortable doing this on my own but do not have a broad toolkit to draw from. I need to put active energy to remember to do this or think about this.
- **Level 4: Improving** – I consciously practice at getting better in this aspect or doing it more regularly so that it becomes habit but it's not habit just yet.
- **Level 5: Old Hand** – I have a broad toolkit that I draw upon, can teach others and use this regularly. People come to me for me expertise and I am trusted source for this.

Characteristic Definitions

Leadership

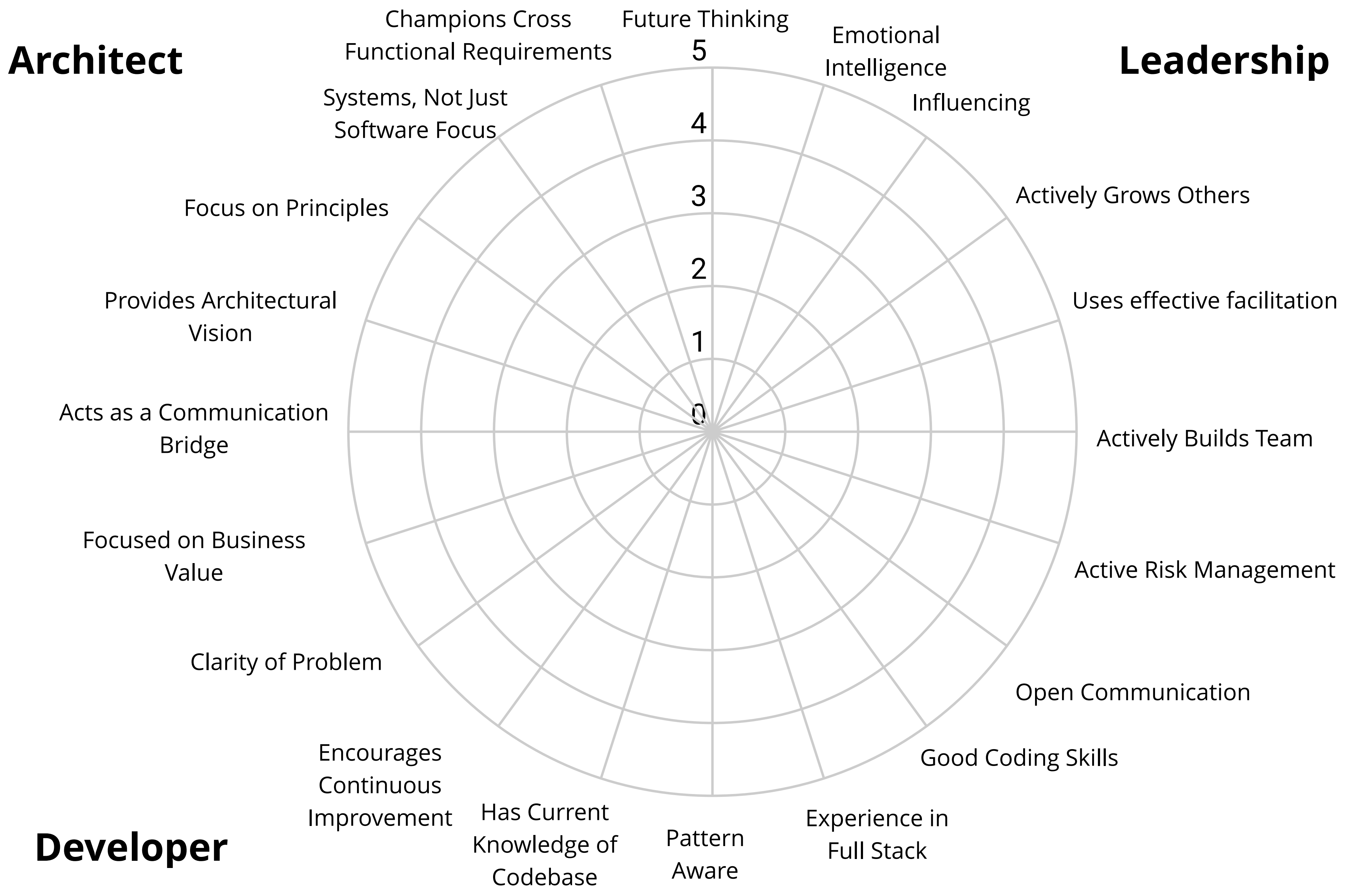
- **Emotional Intelligence** – Uses self- and situational-awareness to lead. Aware of the emotional state of other people in conversation and ensuring that their needs are met as well as my own. Aware of the state of the team/environment and can act accordingly.
- **Influencing** – Listens carefully, aware of different influencing style and understands how to convince others and successfully influences decisions at all levels (inter team and outside of team)
- **Actively grows others** – Understand the strengths/gaps of team members and works actively to build other people
- **Uses effective facilitation** – Ability to use facilitation skills to draw out other opinions, overcome conflict and to build consensus in a variety of situations (1:1, team meetings, general meetings).
- **Actively builds team** – Finds ways to turn a group of people into a team working towards the same goals. Focuses on identifying and resolving underlying team conflicts
- **Active risk management** – Consciously has a risk management strategy and involves everyone in identifying, mitigating risk both from a business and especially from a technical perspective.
- **Open communication** – Ensures there are regular, open communication channels both within and outside the team.

Developer

- **Good coding skills** – Is able to understand how to refactor, detect code smells and knows how to write modular, testable code. Understands test/infrastructure automation. The developers need to have respect that the Tech Lead is a competent developer.
- **Experience in full stack** – Understands the tools/patterns and comfortable using/modifying most of the tools across the entire system (build, development, testing, database, back-end/front-end, etc). Does not necessarily need to be the expert in all.
- **Pattern aware** – Understands software patterns and knows when/ when not to use them and help developers to simplify the codebase.
- Has **current knowledge of codebase** – Understands the current state of the codebase, knows where the "hotspots" or technical constraints of the codebase are and understand how the current system affects potentially new decision. Is comfortable editing/ adding to the existing codebase.
- **Encourages Continuous improvement** – Looks at managing technical debt and opportunities to use the latest tools/technology to improve the agility of the team.
- **Clarity of problem** – Draws upon analysis skills to ensure that the right problem is being solved, and that the most appropriate (may not even be technical) is found to solve them.

Architect

- **Focused on Business Value** – Understands and communicates regularly how the system fulfils business needs. Focuses the team on delivering the system in the most effective way. Actively looks at how technology can aid and shape business needs/opportunities.
- **Acts as a Communication bridge** – Communicates technical ideas/ constraints in ways that other non-technical people understand them to keep them informed/help them make better decisions.
- **Provides Architectural Vision** – Able to think beyond the current need to forecast how this fits into the bigger picture. Actively keeps the architectural vision alive and relevant within the development team. Updates the architectural vision to reflect implementation constraints and adapts to business needs.
- **Focus on principles** – Communicates decision-making in terms of principles, not hard-decisions and follows up to ensure principles are followed.
- **Systems, not software focus** – Understands how software is shipped into production, can articulate the differences between development and the production environment and is aware and champions the operational functional requirements as much as the functional requirements.
- **Champions Cross-Functional Requirements (CFRs)** – Identifies and monitors applicability of system quality attributes, ensuring they are met or adjusted as a system evolves or the environments changes.
- **Future thinking** – Keeps abreast of technology changes to think about what may be relevant technology or opportunities.


Architect
Leadership
Developer
Future Thinking
Emotional Intelligence
Influencing
Actively Grows Others
Uses effective facilitation
Actively Builds Team
Active Risk Management
Open Communication
Good Coding Skills
Experience in Full Stack
Pattern Aware
Has Current Knowledge of Codebase
Encourages Continuous Improvement
Clarity of Problem
Focused on Business Value
Acts as a Communication Bridge
Provides Architectural Vision
Focus on Principles
Systems, Not Just Software Focus
Champions Cross Functional Requirements
5
4
3
2
1
0

Cross Functional Requirements (CFRs)

This list is sometimes called System Quality Attributes/Traits or classically Non-Functional Requirements (NFRs). NFRs give non-technical people the impression there is no additional work to do. CFRs cut across a system, and need to be considered with every User Story/ Feature. I favour the term **Cross-Functional Requirements** to encourage everyone to consider these during development and planning.

CFR	**Meaning**
Accessibility	Support users with special needs (e.g. screen reading)
Archiving	What, how, when is information archived and who/how will it be accessed?
Auditability	What operations should be tracked? What legal or regulatory requirements must be met?
Authentication	How are users identified? What standards should be followed or what existing authentication systems should be used?
Authorisation	What roles and permissions are necessary and what access do they need? Who maintains this and at what levels will they be applied?
Availability	Is there a target availability required? What architecture do you need to satisfy these requirements? Are there specific days (e.g Black Fridays) where these are required? CAP (Consistency, Availability, Partition Tolerance)
Compatibility	What other systems do you need to integrate with and what other industry standards do you need to follow? Is there existing data formats you need to consider?
Configurability	Can users or administrators configure feature behaviours? How will this be managed (e.g. file, UI, API, etc)
Continuity	Disaster recovery plans?
Data Integrity and Consistency	Do you need to implement checksum, journaling or provide data recovery mechanisms?
Data Privacy	What data should be encrypted or visible/hidden from end users and operators? Can dev/testing have production data, or how do you mask/ remove relevant information?
Distributability	Will people be able to use the system in a specific area without location or distance being a hindrance? Can it operate with internet access? How do you sync information?
Extensibility	Do you need to provide a plugin capability and who for? What support do you need to provide and restrictions/security you need to consider?

Help & Support	What types and level of documentation do you need? A tutorial, video or provide a help and support team? Do you need to plan for training?
Installability & Deployment	What platforms will you support? Installation process? Will you support Continuous Delivery? How do you rollback/upgrade?
Integratability & Interoperability	Do you need to provide an API or library for other systems to use, and what level of versioning/upgrade do you need to support?
Legal Compliance	Are there legislative constraints on the data/system or software process?
Leveragability and reuse	Are there existing mandates to reuse enterprise components/libraries or will this be reused?
Localisation & Internationalisation	Is there multi-language support expected for static/operational data? Date/time/currency? Conversions and translations? Do you need external translators?
Monitoring	Are there existing tools and infrastructure in place? What business/technical metrics should be measured? How/what will monitoring be performed? What types of alerts are necessary?
Multiple Env Support	How many/what environments are ideal and how do you provision and manage these environments?
Performance	What throughput/response time requirements are important? Load before system fails? Do you need to plan for performance testing?
Portability	How important is it to move to another DB, operating system?
Resilience & Fault Tolerance	In what ways can the system downgrade functionalities if something (e.g. external dependency) starts failing/unavailable?
Reliability	What is the business cost of inaccuracy and how much do you need to guarantee or safeguard this (e.g. space shuttle versus mobile game)
Scalability	How do you increase throughout/response based on changing volumes of users? How will you test for this?
Security	What processes need to be put in place for security audits or penetration testing? What are the enterprise guidelines? SSL or VPN requirements?
Supportability	Define levels of user/operations support and what support you provide?
Usability and user experience	How important is UX for this system and how will that work in the process? Are there company UX guidelines to follow? Testing?
Versioning and upgrades	What will be your versioning strategy? How do you track internal/external versions? Any backward compatibility constraints?

For each of these, consider WHAT is you approach (e.g. feature, process or automated fitness function) and HOW you plan to fulfil and validate these? Be sure to quantify these.

Exercise: Going... Going... Gone!

Background

Congratulations! Your company is in the final round of a vendor selection process from an auction company, "**Bid4It.com**". In this final round, you have been asked to describe how you will build a quality, robust solution for their needs.

Bid4It started as a traditional auction house in the country capital. They now want to take their auctions online to a nation-wide scale. Customers can find auctions they would like to take part in, they can register interest in the auction, wait until the auction begins and bid as if they were there in the room, with the auctioneer.

Users:

Scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible.

Requirements:

- Auctions must be categorised and 'discoverable'.
- Auctions must be as real-time as possible.
- Auctions must be mixed live and online (i.e participants can either be in the auction house bidding, or place their bid online)
- Video stream of the action after the fact should be made available.
- Handle money exchange in various currencies.
- Participants must be tracked via a reputation index. A reputation index shows how many auctions a participants has taken part in.

Additional Context:

- The auction company is expanding aggressively by merging with smaller competitors.
- Budget is not constrained — this is a strategic direction
- Company just exited a lawsuit where they settled a suit alleging fraud

Your Task:

You will have 5-10 minutes to convince **Bid4It** that you have the best approach to building a quality solution. Ensure your group:

1. Identifies which Cross Functional Requirements are most important
2. Decides how you will address each Cross Functional Requirement - consider if you can automate these
3. Articulate any assumptions and target numbers/goals you have made

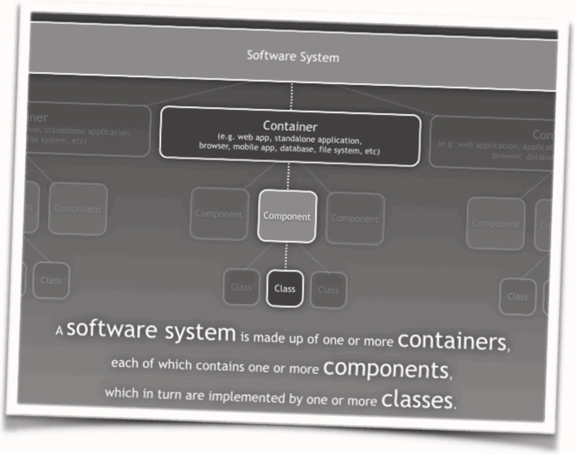
Source: Inspired by the Going...Going...Gone! Architecture Kata from (see http://nealford.com/katas/list.html)

Software architecture and the C4 model

coding {the} architecture

1 Agree on a simple set of abstractions that the whole team can use to communicate.

A common set of abstractions is more important than a common notation, but do ensure that your notation (shapes, colours, line styles, acronyms, etc) is understandable. Add a key/legend if in doubt.

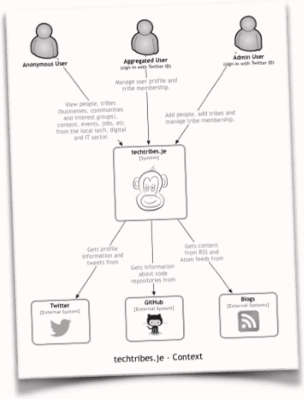

Software System
Container
(e.g. web app, standalone application, browser, mobile app, database, file system, etc)
Component
Class

A software system is made up of one or more containers, each of which contains one or more components, which in turn are implemented by one or more classes.

2 Draw a number of simple diagrams at different levels of abstraction.
C4: context, containers, components, classes

Naming
If naming is one of the hardest things in software development, resist the temptation to have a diagram full of boxes that only contain names!

Adding a brief statement of responsibilities to containers and components is a simple way to remove ambiguity. It provides a nice "at a glance" view too.



1. System context diagram

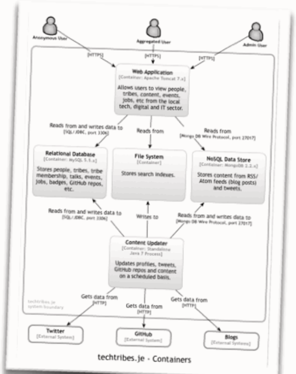
A context diagram can be a useful starting point for diagramming and documenting a software system, allowing you to step back and look at the big picture. Draw a simple block diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interfaces with.

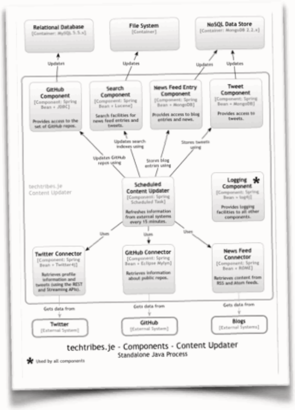
Detail isn't important here as this is your zoomed out view showing a big picture of the system landscape. The focus should be on people (actors, roles, personas, etc) and software systems rather than technologies, protocols and other low-level details. It's the sort of diagram that you could show to non-technical people.

2. Container diagram

Once you understand how your system fits in to the overall IT environment with a context diagram, a really useful next step can be to illustrate the high-level technology choices with a containers diagram. By "container" I mean something like a web application, desktop application, mobile app, database, file system, etc. Essentially, what I call a container is anything that can host code or data.

A container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another. It's a simple, high-level technology focussed diagram that is useful for software developers and support/operations staff alike.





3. Component diagram(s)

Following on from a container diagram showing the high-level technology decisions, I'll then start to zoom in and decompose each container further. However you decompose your system is up to you, but I tend to identify the major logical components and their interactions. This is about partitioning the functionality implemented by a software system into a number of distinct components, services, subsystems, layers, workflows, etc.

The components diagram shows how a container is divided into components, what each of those components are, their responsibilities and the technology/implementation details.

How would you code it?
Ask yourself this question if you can't figure out how to draw something.

4. Class diagram(s)

This is optional, but I sometimes draw one or more UML class diagrams if I want to illustrate specific component implementation details.

More information about software architecture and the C4 model can be found in "Software Architecture for Developers", available as an ebook from leanpub.com

@simonbrown | www.codingthearchitecture.com

System Name:

Why does this system exist? (Short description)

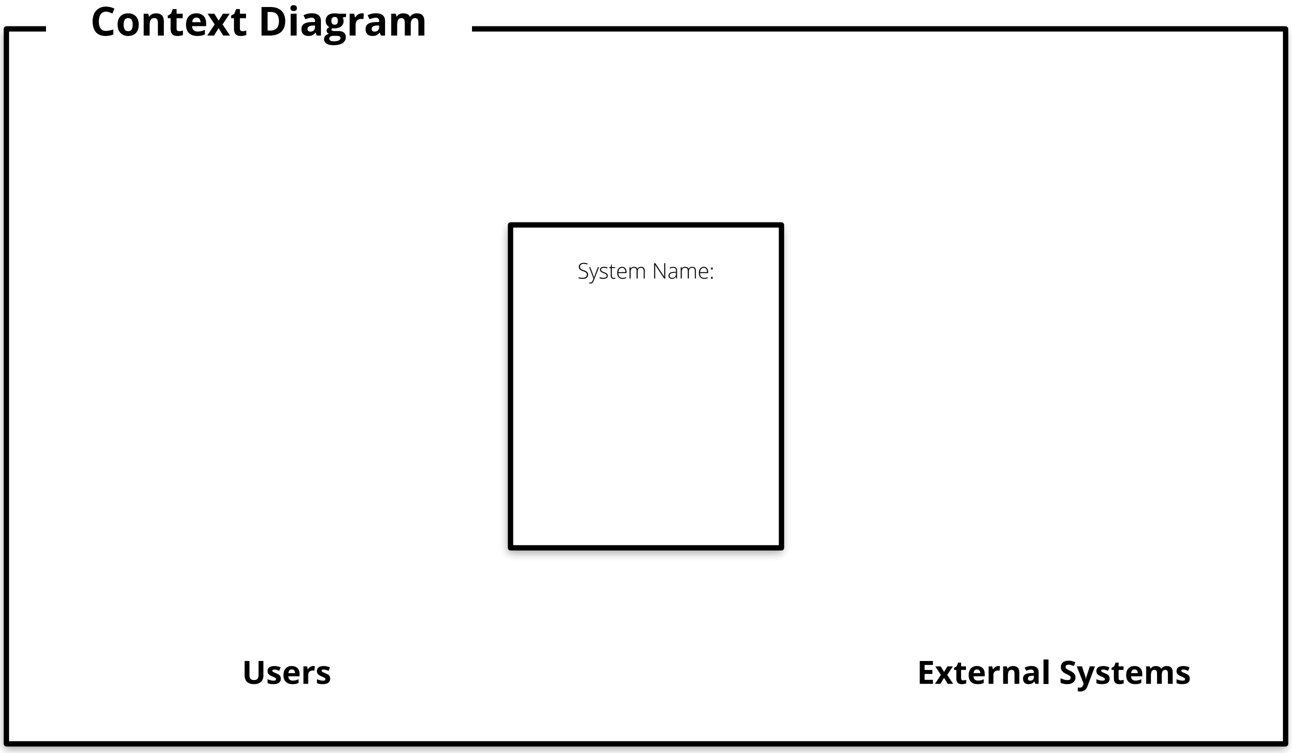
Key Responsibilities:

•
•
•
•
•
•

Key CFRs

•
•
•
•
•

Context Diagram



Container Diagram

Improve the Path to Production - Brief

You have joined a team building an online retail platform selling widgets in high demand. The business brought you on because they have many issues with outages, stability, and are worried about the team. **You are to present a concrete, realistic plan to improve outages, stability and prevent more downtime**.

The online retail platform is built using PHP, deployed as a single monolith onto the Google Cloud Platform (GCP). Production runs an instance on 5 servers in a load-balanced active-active configuration.

Your observations include:

- Developers click through the application on their machines before releasing changes
- A developer creates a zip manually on their machine and uploads it to a remote server from their machine
- There is only the developers' machines and the production environment
- Customer complaints are the first indicator the system does not work
- When the site goes down, the team has no idea what is causing it so they simply restart each service, which has usually worked
- There are 5 servers in production, each running a copy of the software. You feel like some instances are out of date but you cannot prove this
- A separate team manages their cloud infrastructure. Your team has access to each server (via SSH) and can also optionally change server configuration via a web dashboard.

You have talked to people, gathered data, observed how the team works and you must now **present a plan to the business** that includes:

- Prioritised work to be done
- Estimated time and effort
- Estimates of **significant costs** (either upfront or long term)

Document any assumptions you make in your plan. You will be presenting this to a panel of three people:

- Head of Product
- CEO
- Head of Software Engineering (who doesn't have much of a programming background)

Conflict Self Assessment

Next to each of the following 16 items, indicate how often you do the following when you differ with someone:

- **Usually:** give yourself a 5
- **Sometimes:** give yourself a 3
- **Seldom:** give yourself a 1

When I differ from someone:

1. I explore our differences, not backing down, but not imposing my view either
2. I disagree openly, then invite more discussion about our differences
3. I look for a mutually satisfactory solution
4. Rather than let the other person make a decision without my input, I make sure I am heard and also that I hear the other out
5. I agree to a middle ground rather than look for a completely satisfying solution
6. I admit I am half wrong rather than explore our differences
7. I have a reputation for meeting a person halfway
8. I expect to get out about half of what I really want to say
9. I give in totally rather than try to change another's opinion
10. I put aside any controversial aspects of an issue
11. I agree early on, rather than argue about a point
12. I give in as soon as the other party gets emotional about an issue
13. I try to win the other person over
14. I work to come out victorious, no matter what
15. I never back away from a good argument
16. I would rather win than end up compromising

Calculate your scores:

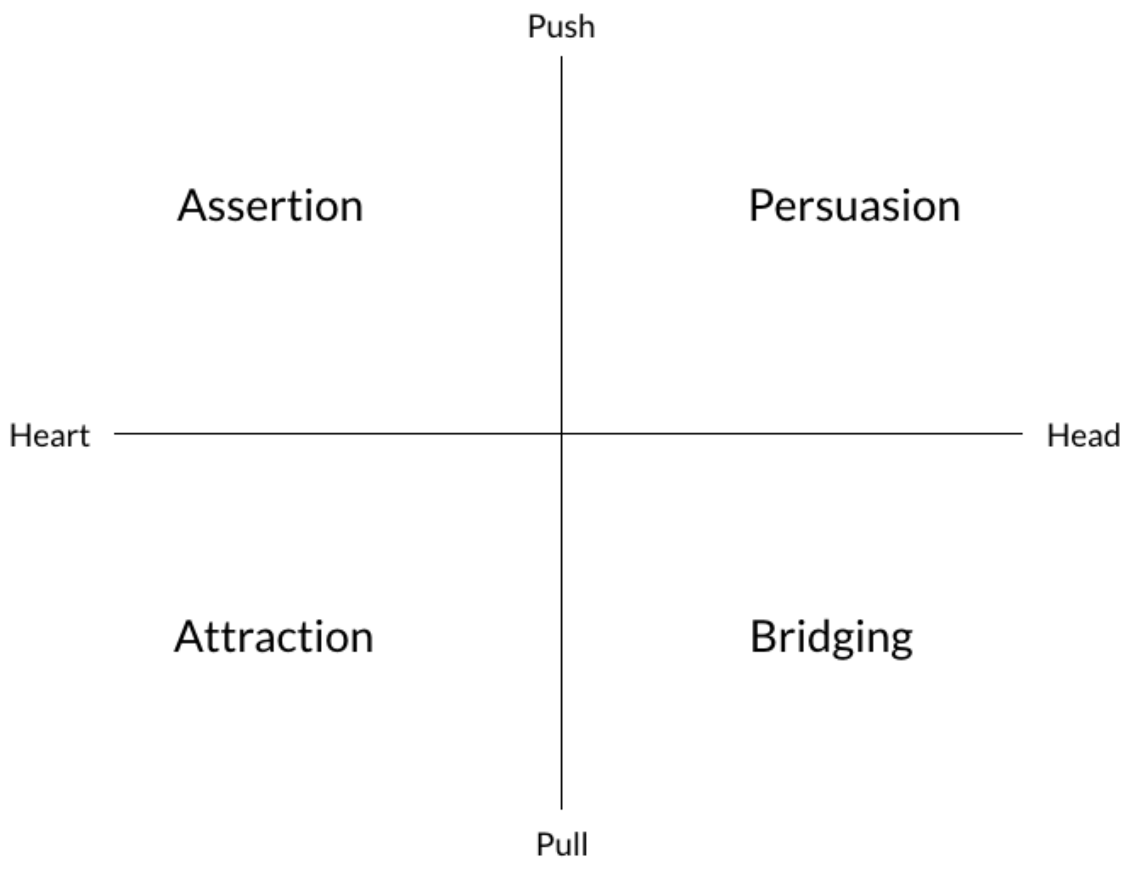
Add the totals of Questions 1-4 – A ________________________________

Add the totals of Questions 5-8 - B ________________________________

Add the totals of Questions 9-12 – C ________________________________

Add the totals of Questions 13-16 – D ________________________________

Influencing Styles



Video Worksheet – Influencing

For each of the videos, identify which influencing style you believe is being used the most and write down why.

1. Dr Strangelove (https://www.youtube.com/watch?v=VEB-OoUrNuk)

2. Princess Bride (https://www.youtube.com/watch?v=lC6dgtBU6Gs)

3. V for Vendetta (https://www.youtube.com/watch?v=dKnjxT5HRJQ)

4. Office Space (https://www.youtube.com/watch?v=YYFO5qPpM_I)

5. Inside Out (https://www.youtube.com/watch?v=QT6FdhKriB8)

Action Plan Exercise

Name:

Pair (Name & Contact Info):

3 day

Action Item

3 week

Action Item

3 month

Action Item

Further Reading Resources

Books, Papers and Online Resources

The following list is a recommendation of books grouped in a number of areas that will help you on your journey to being a more effective technical leader:

Architecture

- **The Software Architect Elevator: Redefining the Architect's Role in the Digital Enterprise by Gregor Hohpe** (https://geni.us/m47Esd9) - This book shares the vast experience of an architect who has seen this role evolve in many organisations over the years.
- **Building Evolutionary Architectures by Neal Ford et al** (https://geni.us/YKEi8) - An overview of approaches to designing systems for constant change.
- **Building Microservices: Designing Fine-Grained Systems by Sam Newman** (https://geni.us/AGOAL) - A great overview of the general approach to designing *effective* microservices and understanding their trade-offs
- **Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation by David Farley and Jez Humble** (https://geni.us/qmBjx) - Understand the principles behind building software systems that can be repeatedly and reliably released.
- **Design Patterns by Erich Gamma et al** (https://geni.us/C0qiqtf) - The famous “Gang of Four” book introducing names of patterns to common programming approaches.
- **Design It! by Michael Keeling** (https://geni.us/FeJIX) - Full of practical ideas with workshops and tools to encourage collaborative design and architecture.
- **Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems by Martin Kleppmann** (https://geni.us/txkLCI) - Essential reading for any team building systems handling large sets of data.
- **Domain-driven Design: Tackling Complexity in the Heart of Software by Eric Evans** (https://geni.us/JjEtc5R) - Contains a list of principles and practices for bringing software developers closer to the business.
- **Enterprise Architecture As Strategy** by Jeanne W. Ross et al (https://geni.us/StoOx) - Considers where architecture in a large business reflects business strategy and the interplay between the two.
- **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions by Gregor Hohpe and Bobby Wolfe** (https://geni.us/jTrYm) - A catalogue of patterns describing different integration mechanisms and when to use them.

- **Fundamentals of Software Architecture: An Engineering Approach by Mark Richards & Neal Ford** (https://geni.us/wF3HH) - A great introduction to the core concepts of software architecture in the modern era.
- **Just Enough Software Architecture by George Fairbanks** (https://geni.us/AlaL) - A look at using architecture to manage risks. Practical and aimed at developers.
- **Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith by Sam Newman** (https://geni.us/d01Y0) - A follow-up book for Building Microservices showing approaches when migrating away from a monolith to microservices.
- **Patterns of Enterprise Application Architecture by Martin Fowler** (https://geni.us/mQJG) - A catalogue of common patterns book an application architect may use in different contexts.
- **The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win by Gene Kim et al** (https://geni.us/UnBiZJA) - A business novel describing the hurdles of shipping software and how it can be better managed.
- **Release It!: Design and Deploy Production-Ready Software by Michael T. Nygard** (https://geni.us/msPN) - Tells a number of war stories and provides useful advice designing software for production systems (and dealing with failure!).
- **Site Reliability Engineering by Betsy Beyer et al** (https://geni.us/X0Daq4) - Behind the scenes of how Google approaches operating software at scale.
- **Software Architecture in Practice by Len Bass et all** (https://geni.us/n46o) - Classic literature on architecture, though with a slightly academic and theoretical foundation. Still highly recommended as foundational reading.
- **Software Architecture for Developers by Simon Brown** (http://www.codingthearchitecture.com) - Describes architecture in an agile and pragmatic approach. Lots of great advice about drawing architecture diagrams and what other considerations you want from architecture.
- **Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives by Nick Rozanski et al** (https://geni.us/6vZ5tau) A good overview of different perspectives that need to both be considered in software architecture, and likely documented.
- **Waltzing with Bears: Managing Risk on Software Projects by Tom DeMarco** (https://geni.us/zDZAP) - Introduces risk management in the context of software development. Although old, it is still highly relevant to today’s industry.
- **The Unicorn Project by Gene Kim** (https://geni.us/qkklDJn) - A sequel to The Phoenix Project, focused on changing software architecture that enables agility.

People

- **Bad is Stronger than Good by Roy F. Baumeister et al (2001)** – A paper that describes our negativity bias. This paper has good implications about dealing with Bad Apples in teams.
- **The Difference by Scott E. Page** (https://geni.us/FQJ5) - Describes the power of diversity and the necessary conditions about when it really helps.
- **Drive: The Surprising Truth About What Motivates Us by Daniel Pink** (https://geni.us/hWzLQ) – All leaders should read this book to understand the environment to create to engage people: Autonomy, Mastery and Purpose
- **Emotional Intelligence 2.0 by Travis Bradberry and Jean Graves** (https://geni.us/QBkeXe) - Describes strategies of how to identify, understand and manage emotions in yourself and others.
- **Flow: The Psychology of Happiness by Mihaly Csikszentmihalyi** (https://geni.us/8uSHaG) – Describes how people achieve "flow" and the necessary conditions. Lots of great examples in different industries.
- **Games People Play: The Psychology of Human Relationships by Eric Berne** (https://geni.us/3HnGV) - What are functional and dysfunctional communication patterns that describe common games people play when having conversations.
- **How to Win Friends and Influence People by Dale Carnegie** (https://geni.us/HAdblr) - An oldie but a good book that talks about some principles of influencing.
- **Hofstede Cultural Dimensions** (http://geert-hofstede.com/) - A website that introduces the idea of cultural dimensions across different countries.
- **Mindset: Changing The Way You think To Fulfil Your Potential by Dr Carol S. Dweck** (https://geni.us/VgnVK) - The book that talks about the Growth versus Fixed mindset and how each mindset helps/hinders performance.
- **Non-Violent Communication by Marshall Rosenberg** (https://geni.us/SvWw) - Tools and frameworks to communicate your needs without escalating or inflaming a situation.
- **Opinion: The unspoken truth about managing geeks by J. Ello** (http://thekua.io/unspoken-truth-geeks) – Explains how developers approach trust and respect.
- **Radical Candor by Kim Scott** (https://geni.us/426BW) - Communication tips to focus on being direct and having empathy. Helpful in being clear, giving feedback and still maintaining good relationships with people.
- **Thanks for the Feedback: The Science and Art of Receiving Feedback Well by Douglas Stone et al** (https://geni.us/BpZO) - A guide to understanding how to extract useful feedback out of conversations with people.

Teams and Organisations

- **Accelerate: The Science of Lean Software and Devops: Building and Scaling High Performing Technology Organizations by Nicole Forsgren and Jez Humble** (https://geni.us/spMICc8) - A book backed by research from the yearly State of DevOps Report that demonstrates organisational and technical factors that lead to high-performance technology firms.
- **A Leaner Start: Reducing Team Setup Times** (https://www.infoq.com/articles/pat-kua-onboarding-new) - An article with specific ideas to accelerate the onboarding of new team members.
- **Creativity Inc. by Ed Catmull** (https://geni.us/PcLi) - Behind the scenes stories of building that culture that led to the award-winning Pixar Studios.
- **The Devops Handbook by Gene Kim et al** (https://geni.us/0p32) - Concrete advice and practices to build a DevOps culture.
- **The Five Dysfunctions of a Team by Patrick M. Lencioni** (https://geni.us/a1Vju) - A great business novel highlighting the factors that can break a team and what you can do to improve it.
- **The Mythical Man-Month by Fred Brooks** (https://geni.us/YeNnI3z) - Classic literature that is a bit dated, but still related to poor estimating and planning processes and what to do to avoid them.
- **Leaders Eat Last by Simon Sinek** (https://geni.us/XUxBd) - An insight into how humans react and why that's important to building high performing teams.
- **Lift Off: Launching Agile Projects & Teams by Diana Larsen et al** (https://geni.us/zYtzsHl) – Offers concrete techniques to build team charters and to accelerate the norming phase of a team.
- **Project Aristotle** (https://rework.withgoogle.com/print/guides/5721312655835136/) - Google's research into what makes a high performing team.
- **Reinventing Organizations by Nelson Parker** (https://geni.us/cAB59Y) - Explores the ideas of "teal" organisations focused on building a unique environment to support motivating and empowering people and teams.
- **Running effective meetings (**http://thekua.io/effective-meetings) - Outlines some basic approaches to running more effective meetings.
- **Team Topologies: Organizing Business and Technology Teams for Fast Flow by Matthew Skelton & Manuel Pais** (https://geni.us/BwnnlEB) - Underscores the trade-offs, benefits and patterns of organisational structures when drawing the boundaries of teams.

Leadership

- **Adaptive Leadership by Jim Highsmith** (https://geni.us/Rk7Z6) – Details how a modern-day leader should work in an agile environment.
- **The Coaching Habit by Bungay Stanier Michael** (https://geni.us/GsDpBL) - Introduces the role and practices of an effective coach for building coaching skills.
- **Difficult Conversations: How to Discuss What Matters Most by Roger Fisher et al** (https://geni.us/PlSIA6) - Takes a deeper view as an individual into how to navigate conflict by understanding the multi-layered conversations - “What Happened”, Feelings and Identities.
- **Crucial Conversations Tools for Talking When Stakes Are High by Kerry Patterson et al** (https://geni.us/7WJd4H) – A step-by-step manual on when and how to approach difficult conversations with people.
- **Facilitator's Guide to Participatory Decision-Making by Sam Kaner** (https://geni.us/RebrrYy) – Details facilitation approaches that can be used in a collaborative fashion – involving other people and leads to greater commitment.
- **First, Break All The Rules: What the World's Greatest Managers Do Differently by Jim Harter et all** (https://geni.us/U5X3CWN) - Research results from Gallup on what makes managers effective.
- **Fearless Change by Linda Rising and Mary Lynn Manns** (https://geni.us/bfZz) – A patterns book with ideas that you can use to introduce change, and to have more influence.
- **Getting Things Done by David Allen** (https://geni.us/uxjaeJ) – A methodology and book that describes a time-management approach to getting more things done.
- **Getting to Yes by Roger Fisher et al** (https://geni.us/mUPMS3) – Introduces the differences between Position-based versus Interest-based negotiation. A short simple book that is very easy to read and apply.
- **The Hard Things About Hard Things by Ben Horowitz** (https://geni.us/cEyaB) - Leadership and management lessons from a well known Silicon Valley executive working at rapidly growing businesses.
- **Influence: The Psychology of Persuasion by Robert Cialdini** (https://geni.us/JJii) - Identifies and examples six key principles of successfully influencing others.
- **Made to Stick: Why some ideas take hold and others come unstuck by Chip Heath and Dan Heath** (https://geni.us/z9s70j) - The power of storytelling and how using stories can attract people to an idea.

- **Making of a Manager by: What to Do When Everyone Looks to You by Julie Zhuo** (https://geni.us/NWTBlx) - A good introduction to when you first step into a management role and how that changes from a VP of Product Design from Facebook
- **Management 3.0 by Jurgen Appello** (https://geni.us/04LBk) - A set of interactive exercises and tools for managing with a modern empowering mindset.
- **Managing Oneself by Peter Drucker** (https://geni.us/K6zG) - A self-help piece that is essential for all leaders
- **More Fearless Change by Linda Rising and Mary Lynn Manns** (https://geni.us/E1YU) - Even more patterns of introducing change from the authors of Fearless Change
- **Multipliers: How the Best Leaders Make Everyone Smart by Liz Wiseman** (https://geni.us/CVSWc) - Describes how the actions of leaders can either drain or amplify the ability of the people they lead.
- **The Power of Vulnerability by Brené Brown** (https://www.ted.com/talks/brene_brown_on_vulnerability) – A fantastic TED talk that illustrates how leaders can use vulnerability to have more influence and impact.
- **Resilient Management by Lara Hogan** (https://resilient-management.com/) - A short, clear, simple handbook with great actionable tools for people managers.
- **Start With Why: How Great Leaders Inspire Everyone To Take Action by Simon Sinek** (https://geni.us/GbmnyLo) - Discusses how to create a vision by creating compelling reasons that people can contribute to and have fulfilling work.
- **Thinking in Systems by Donella Meadows** (https://geni.us/TmZki) - The most accessible and easy digest book on Systems Thinking
- **Turn the Ship Around! By L. David Marquet** (https://geni.us/8GZF) - A great book that talks about the leadership style that changed one of the worst-performing ships in the US navy to one of the best.
- **Strengthsfinder 2.0 by Tom Rath** (https://geni.us/I2qTz) - Helps you build a better understanding of yourself and others and to understand why taking a strengths-based approach is useful.
- **What Got You Here Won't Get You There: How successful people become even more successful by Marshall Goldsmith** (https://geni.us/kPlQo8U) - A great book focused on essential leadership behaviours (and those that hold leaders back).

Engineering Management

- **An Elegant Puzzle: Systems of Engineering Management by Will Larson** (https://geni.us/ORwTm) - A book of opinionated advice from a person who's played everything from Engineering Manager to Director across many modern tech firms.
- **Become an Effective Software Engineering Manager: How to Be the Leader Your Development Team Needs by James Stainer** (https://geni.us/7avN8w) - A practical book outlining the common activities and tools for engineering managers.
- **Behind Closed Doors: Secrets of Great Management by Johanna Rothman and Esther Derby** (https://geni.us/pSpg) - A practical book with lots of stories with ideas on delegating, goal setting, doing 1-1s and more.
- **Butterfly Model of Technical Leadership by Dan Abe**l (http://thekua.io/butterfly-model) – A different model that describes the Technical Lead role.
- **Leading Snowflakes by Oren Ellenbogen** (https://leadingsnowflakes.com/) - A short ebook with a good overview of common areas an Engineering Manager will find themselves dealing with.
- **High Output Management by Andy Grove** (https://geni.us/mDqi) - A personal management guide from Intel's famous CEO. Often regarded as a highly-praised guide to modern engineering management.
- **The Manager's Path by Camille Fournier** (https://geni.us/YnCBCvK) - A concise guide to looking at different leadership and management roles in a modern technology company.
- **Managing Humans by Michael Lopp** (https://geni.us/rUSuyqJ) - A collection of entertaining tales dealing with people situations within technology companies written by the author of Rands in Repose.
- **Managing the Unmanageable by Mickey Mantle et al** (https://geni.us/iPs6LZJ) - A very specific book on engineering management that really considers how managing developers changes the way you manage.
- **Peopleware by Tom DeMarco** (https://geni.us/FDNs) - A timeless book about IT management.
- **Project Aristotle** (https://rework.withgoogle.com/blog/the-evolution-of-project-oxygen/) - Google's research into what makes an effective manager.
- **Scaling Teams by Alexander Grosse and David Loftesness** (https://geni.us/Iq1RyZR) - A concise book looking at everything from hiring, people management, organisation, culture and communication strategies for the modern manager.
- **Talking with Tech Leads by Patrick Kua** (http://thekua.io/twtl) – A collection of interviews with Tech Leads from various backgrounds and industries sharing their stories and approaches to being a Tech Lead.

Models

Any of the models below are easily found on the Internet.

- **BICEPS** (https://www.palomamedina.com/biceps) - Six core needs that are most important for humans at work.
- **GROW** – A commonly used tool for coaching contexts. Goals, Reality, Options and Way Forward
- **Situational Leadership Model** – A useful framework to consider what leadership style you should use based on the task at hand and the person doing it.
- **Tuckman's Model** – A description of phases a team experiences during its lifecycle.
- **Eisenhower Quadrant** – A tool to help you identify where you add value and what you should be working on
- **Belbin Team (Team Role Inventories)** – A team analysis tool that helps you understand individual preferences and can help a team better understand each other to work better.
- **Vroom-Yetton-Jago Decision Model** – A framework to help you decide what decision-making method you should use based on the importance of the decision and how much commitment you want from team members.