Part I

Introduction

Part II

The Risk Landscape

CHAPTER 1

Coordination Risk

As in Agency Risk, we are going to use the term *agent*, which refers to anything with agency[1] in a system to decide it's own fate. That is, an agent has an Internal Model, and can take actions based on it. Here, we leave aside Agency Risk and work on the assumption that the agents *are* working towards a common Goal, even though in reality it's not always the case, as we saw in the chapter on Agency Risk.

Coordination Risk is the risk that, a group of people or processes, maybe with a common Goal In Mind, can fail to coordinate to meet this goal and end up making things worse. Coordination Risk is embodied in the phrase "Too Many Cooks Spoil The Broth": more people, opinions or *agents* often make results worse.

In this chapter, we'll first build up a model of Coordination Risk and what exactly coordination means and why we do it. Then, we'll look at some classic problems of coordination. Then, we're going to consider agency at several different levels (because of Scale Invariance) . We'll look at:

- Team Decision Making,
- Living Organisms,
- Larger Organisations and the staff within them,
- and Software Processes.

. . . and we'll consider how Coordination Risk is a problem at each scale.

But for now, let's crack on and examine where Coordination Risk comes from.

[1] https://github.com/risk-first/website/wiki/Agency-Risk#software-processes-and-teams

1.1 A Model Of Coordination Risk

Earlier, in Dependency Risk, we looked at various resources (time, money, people, events etc) and showed how we could depend on them, taking on risk. Here, however, we're looking at the situation where there is *competition for those dependencies*, that is, Scarcity Risk: other parties want to use them in a different way.

Law Of Diminishing Returns

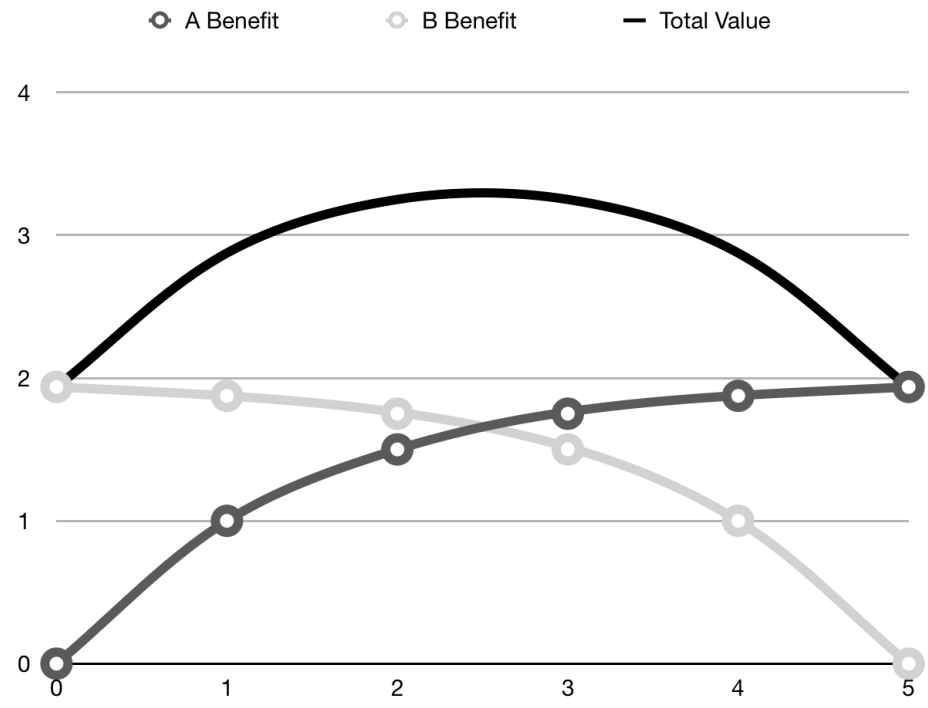


Figure 1.1: Sharing Resources. 5 units are available, and the X axis shows A's consumption of the resource. B gets whatever remains. Total benefit is maximised somewhere in the middle

One argument could come from Diminishing Returns[2], which says that the earlier units of a resource (say, chocolate bars) give you more benefit than later ones.

We can see this in Figure 1.1. Let's say A and B compete over a resource, of which there are 5 units available. For every extra A takes, B loses one. The X

[2] https://en.wikipedia.org/wiki/Diminishing_returns

axis shows A's consumption of the resource, so the biggest benefit to A is in the consumption of the first unit.

As you can see, by *sharing*, it's possible that the *total benefit* is greater than it can be for either individual. But sharing requires coordination. Further, the more competitors involved, the *worse* a winner-take-all outcome is for total benefit.

Just two things are needed for competition to occur:

- Multiple, Individual agents, trying to achieve Goals.
- Scarce Resources, which the agents want to use as Dependencies.

Coordination via Communication

The only way that the agents can move away from competition towards coordination is via Communication, and this is where their coordination problems begin.

Coordination Risk commonly occurs where people have different ideas about how to achieve a goal, and they have different ideas because they have different Internal Models. As we saw in the chapter on Communication Risk, we can only hope to synchronise Internal Models if there are high-bandwidth Channels available for communication.

You might think, therefore, that this is just another type of Communication Risk problem, and that's often a part of it, but even with synchronized Internal Models, coordination risk can occur. Imagine the example of people all trying to madly leave a burning building. They all have the same information (the building is on fire). If they coordinate, and leave in an orderly fashion, they might all get out. If they don't, and there's a scramble for the door, more people might die.

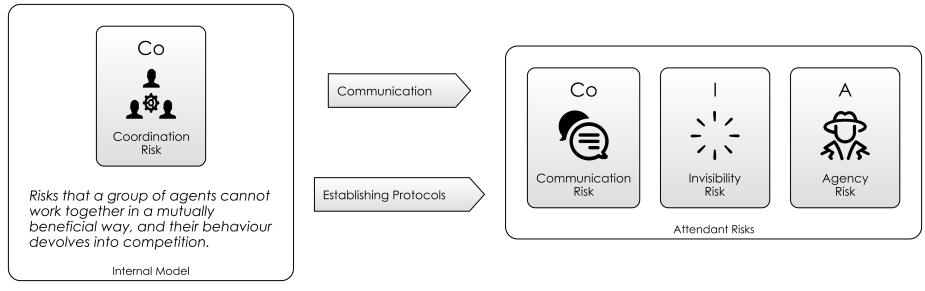


Figure 1.2: Coordination Risk - Mitigated by Communication

1.2 Problems Of Coordination

Let's unpack this idea, and review some classic problems of coordination, none of which can be addressed without good communication. Here are some examples:

1. **Merging Data:** If you are familiar with the source code control system, Git[3], you will know that this is a *distributed* version control system. That means that two or more people can propose changes to the same files without knowing about each other. This means that at some later time, Git then has to merge (or reconcile) these changes together. Git is very good at doing this automatically, but sometimes, different people can independently change the same lines of code and these will have to be merged manually. In this case, a human arbitrator "resolves" the difference, either by combining the two changes or picking a winner.

2. **Consensus:** Making group decisions (as in elections) is often decided by votes. But having a vote is a coordination issue, and requires that everyone has been told the rules:

- Where will the vote be held?
- How long do you provide for the vote?
- What do you do about absentees?
- What if people change their minds in the light of new information?
- How do you ensure everyone has enough information to make a good decision?

3. **Factions:** Sometimes, it's hard to coordinate large groups at the same time, and "factions" can occur. That the world isn't a single big country is probably partly a testament to this: countries are frequently separated by geographic features that prevent the easy flow of communication (and force). We can also see this in distributed systems, with the "split brain"[4] problem. This is where a network of processes becomes disconnected (usually due to a network failure between data centers), and you end up with two, smaller networks with different knowledge. We'll address in more depth later.

4. **Resource Allocation[5]:** Ensuring that the right people are doing the right work, or the right resources are given to the right people is a

[3] https://en.wikipedia.org/wiki/Git

[4] https://en.wikipedia.org/wiki/Split-brain_(computing)

[5] https://en.wikipedia.org/wiki/Resource_allocation

coordination issue. On a grand scale, we have Logistics[6], and Economic Systems[7]. On a small scale, the office's *room booking system* solves the coordination issue of who gets a meeting room using a first-come-first-served booking algorithm.

5. **Deadlock[8]:** Deadlock refers to a situation where, in an environment where multiple parallel processes are running, the processing stops and no-one can make progress because the resources each process needs are being reserved by another process. This is a specific issue in Resource Allocation, but it's one we're familiar with in the computer science industry. Compare with Gridlock[9], where traffic can't move because other traffic is occupying the space it wants to move to already.

6. **Race Conditions[10]:** A race condition is where we can't be sure of the result of a calculation, because it is dependent on the ordering of events within a system. For example, two separate threads writing the same memory at the same time (one ignoring and over-writing the work of the other) is a race.

7. **Contention**: Where there is Scarcity Risk for a dependency, we might want to make sure that everyone gets fair use of it, by taking turns, booking, queueing and so on. As we saw in Scarcity Risk, sometimes, this is handled for us by the Dependency itself. However if it isn't, it's the *users* of the dependency who'll need to coordinate to use the resource fairly, again, by communicating with each other.

1.3 Decision Making

Within a team, Coordination Risk is at it's core about resolving Internal Model conflicts in order that everyone can agree on a Goal In Mind and cooperate on getting it done. Therefore, Coordination Risk is worse on projects with more members, and worse in organizations with more staff.

As an individual, do you suffer from Coordination Risk at all? Maybe: sometimes, you can feel "conflicted" about the best way to solve a problem. And weirdly, usually *not thinking about it* helps. Sleeping too. (Rich Hickey calls this "Hammock Driven Development[11]"). This is probably because, unbeknownst to you, your subconscious is furiously communicating internally,

[6] https://en.wikipedia.org/wiki/Logistics

[7] https://en.wikipedia.org/wiki/Economic_system

[8] https://en.wikipedia.org/wiki/Deadlock

[9] https://en.wikipedia.org/wiki/Gridlock

[10] https://en.wikipedia.org/wiki/Race_condition

[11] https://www.youtube.com/watch?v=f84n5oFoZBc

trying to resolve these conflicts itself, and will let you know when it's come to a resolution.

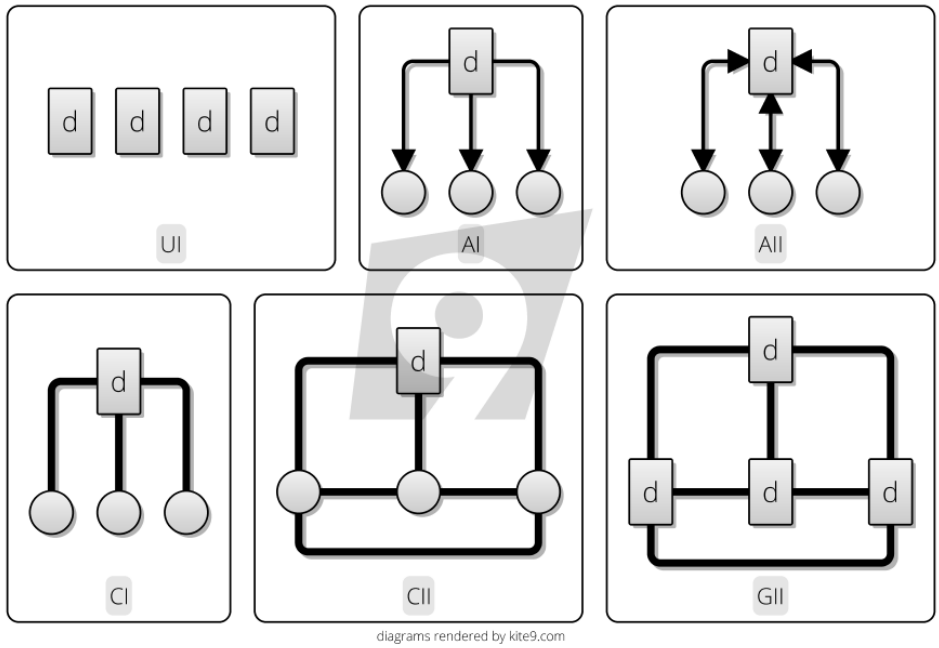


Figure 1.3: Vroom And Yetton Decision Making Styles. "d" indicates authority in making a decision, circles are subordinates. Thin lines with arrow-heads show information flow, whilst thick lines show* opinions *being passed around.

Vroom and Yetton[12] introduced a model of group decision making which delineated five different styles of decision making within a team. These are summarised in the table below (**AI, AII, CI, CII, GII**). To this, I have added a sixth (**UI**), which is the *uncoordinated* option, where everyone competes. Figure 1.3 illustrates these, with the following conventions:

- Thin lines with arrow-heads show a flow of *information*, either one-way or two-way.
- Thick lines show a flow of *opinion*.
- Boxes with corners are *decision makers*, whereas curved corners don't have a part in the decision.

At the top, you have the *least* consultative styles, and at the bottom, the *most*. At the top, decisions are made with just the leader's Internal Model

[12] `https://en.wikipedia.org/wiki/Vroom-Yetton_decision_model`

Type	Description	Decision Makers	Opinions	Channels	Coordination Risk
UI	Uncoordinated	1	1	0	Competition
AI	Autocratic	1	1	**s**	Maximum Coordination Risk
AII	Autocratic (with upward information flow)	1	1	**s**	
CI	Consultative (Individual)	1	1 + **s**	2**s**	
CII	Consultative (Group)	1	1 + **s**	$\mathbf{s}^2$	
GII	Group Consultation and Voting	1 + **s**	1 + **s**	$\mathbf{s}^2$	Maximum Communication Risk, Schedule Risk

s = subordinate

but moving down, the Internal Models of the *subordinates* are increasingly brought into play.

The decisions at the top are faster, but don't do much for mitigating **Coordination Risk**. The ones below take longer, (incurring Schedule Risk) but mitigate more **Coordination Risk**. Group decision-making inevitably involves everyone *learning*, and improving their Internal Models.

The trick is to be able to tell which approach is suitable at which time. Everyone is expected to make decisions *within their realm of expertise*: you can't have developers continually calling meetings to discuss whether they should be using an Abstract Factory[13] or a Factory Method[14], this would waste time. The critical question is therefore, "what's the biggest risk?"

- Is the Coordination Risk greater? Are we going to suffer Dead End Risk if the decision is made wrongly? What if people don't agree with it? Poor leadership has an impact on Morale too.
- Is the Schedule Risk greater? If you have a 1-hour meeting with eight people to decide a decision, that's *one man day* gone right there: group decision making is *expensive*.

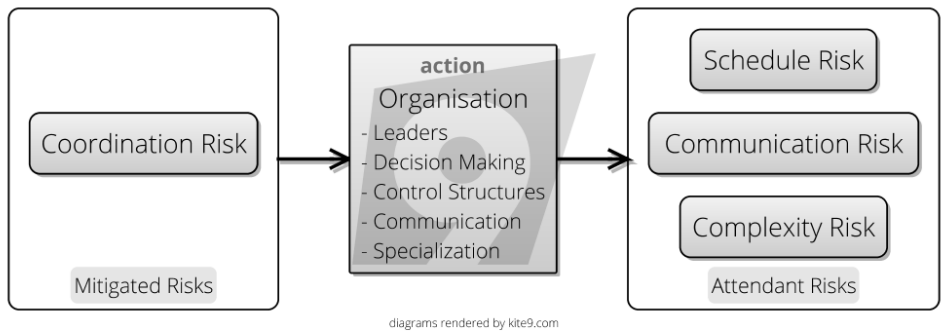


Figure 1.4: Coordination Risk traded for Complexity Risk, Schedule Risk and Communication Risk

Hopefully, this model shows how *organisation* can reduce Coordination Risk. But, to make this work, we need more *communication*, and this has attendant complexity and time costs. So, we can draw diagram above of our move on the Risk Landscape:

[13] https://en.wikipedia.org/wiki/Abstract_factory_pattern

[14] https://en.wikipedia.org/wiki/Factory_method_pattern

Staff As Agents

Staff in a team have a dual nature: they are **Agents** and **Resources** at the same time. The team depends on staff for their resource of *labour*, but they're also part of the decision making process of the team, because they have *agency* over their own actions.

Part of Coordination Risk is about trying to mitigate differences in Internal Models. So it's worth considering how varied people's models can be:

- Different skill levels
- Different experiences
- Expertise in different areas
- Preferences
- Personalities

The job of harmonising this on a project would seem to fall to the team leader, but actually people are self-organising to some extent. This process is called Team Development[15]:

> "The forming–storming–norming–performing model of group development was first proposed by Bruce Tuckman in 1965, who said that these phases are all necessary and inevitable in order for the team to grow, face up to challenges, tackle problems, find solutions, plan work, and deliver results."
>
> —Tuckman's Stages Of Group Development, *Wikipedia*[16]

Specifically, this describes a process whereby a new group will form and then be required to work together. In the process, they will have many *disputes*. Ideally, the group will resolve these disputes internally and emerge as a team, with a common Goal In Mind.

They can be encouraged with orthogonal practices such as team-building exercises[17] (generally, submitting everyone to extreme experiences in order to bond them together). With enough communication bandwidth and detente, a motivated team will self-organise code reviews, information exchange and improve their practices.

As described above, the job of Coordination is Resource Allocation, and so the skills of staff can potentially be looked at as resources to allocate. This means handling Coordination Risk issues like:

[15] https://en.wikipedia.org/wiki/Tuckmans_stages_of_group_development

[16] https://en.wikipedia.org/wiki/Tuckmans_stages_of_group_development

[17] https://en.wikipedia.org/wiki/Team_building

- People leaving, taking their Internal Models and expertise with them Key Man Risk.
- People requiring external training, to understand new tools and techniques Learning-Curve Risk.
- People being protective about their knowledge in order to protect their jobs Agency Risk.
- Where there are mixed ability levels, senior developers not helping juniors as it "slows them down".
- People not getting on and not helping each other.

> "As a rough rule, three programmers organised into a team can do only twice the work of a single programmer of the same ability - because of time spent on coordination problems."
>
> —Gerald Wienberg, *The Psychology of Computer Programming*[18]

1.4 In Living Organisms

Vroom and Yetton's organisational style isn't relevant to just teams of people. We can see it in the natural world too. Although *the majority* of cellular life on earth (by weight) is single celled organisms[19], the existence of *humans* (to pick a single example) demonstrates that sometimes it's better to try to mitigate Coordination Risk and work as a team, accepting the Complexity Risk and Communication Risk this entails. As soon as cells start working together, they either need to pass *resources* between them, or *control* and *feedback*.

For example, in the human body, we have:

- **Various systems[20]:** such as the Respiratory System[21] or the Digestive System[22]. Each of these systems contains. . .
- **Organs**, such as the heart or lungs, which contain..
- **Tissues**, which contain. . .
- **Cells** of different types. (Even cells are complex systems containing multiple different, communicating sub-systems.)

There is huge attendant Communication Risk to running the human body, but, given the success of humanity as a species, you must conclude that these

[18] https://en.wikipedia.org/wiki/Gerald_Weinberg

[19] http://www.stephenjaygould.org/library/gould_bacteria.html

[20] https://en.wikipedia.org/wiki/List_of_systems_of_the_human_body

[21] https://en.wikipedia.org/wiki/Respiratory_system

[22] https://en.wikipedia.org/wiki/Human_digestive_system

steps on the *evolutionary* Risk Landscape have benefited us in our ecological niche.

Decision Making

The key observation from looking at biology is this: most of the cells in the human body *don't get a vote*. Muscles in the motor system have an **AI** or **AII** relationship with the brain - they do what they are told, but there are often nerves to report pain back. The only place where **CII** or **GII** *could* occur is in our brains, when we try to make a decision and weigh up the pros and cons.

This means that there is a deal: *most* of the cells in our body accede control of their destiny to "the system". Living within the system of the human body is a better option than going it alone as a single-celled organism. Occasionally, due to mutation, we can end up with Cancer[23], which is where one cell genetically "forgets" its purpose in the whole system and goes back to selfish individual self-replication (**UI**). We have White Blood Cells[24] in the body to shut down this kind of behaviour and try to kill the rogue cells. In the same way, societies have police forces to stop undesirable behaviour amongst their citizens.

1.5 Large Organisations

Working in a large organisation often feels like being a cell in a larger organism. Just as cells live and die, but the organism goes on, in the same way, workers come and go from a large company but the organisation goes on. By working in an organisation, we give up self-control and competition and accept **AI** and **AII** power structures above us, but we trust that there is symbiotic value creation on both sides of the employment deal.

Less consultative decision making styles are more appropriate then when we don't have the luxury of high-bandwidth channels for discussion, or when the number of parties rises above a room-full of people. As you can see from the table above, for **CII** and **GII** decision-making styles, the amount of communication increases non-linearly with the number of participants, so we need something simpler. As we saw in the Complexity Risk chapter, hierarchies are an excellent way of economizing on number of different communication channels, and we use these frequently when there are lots of parties to coordinate.

[23] https://en.wikipedia.org/wiki/Cancer

[24] https://en.wikipedia.org/wiki/White_blood_cell

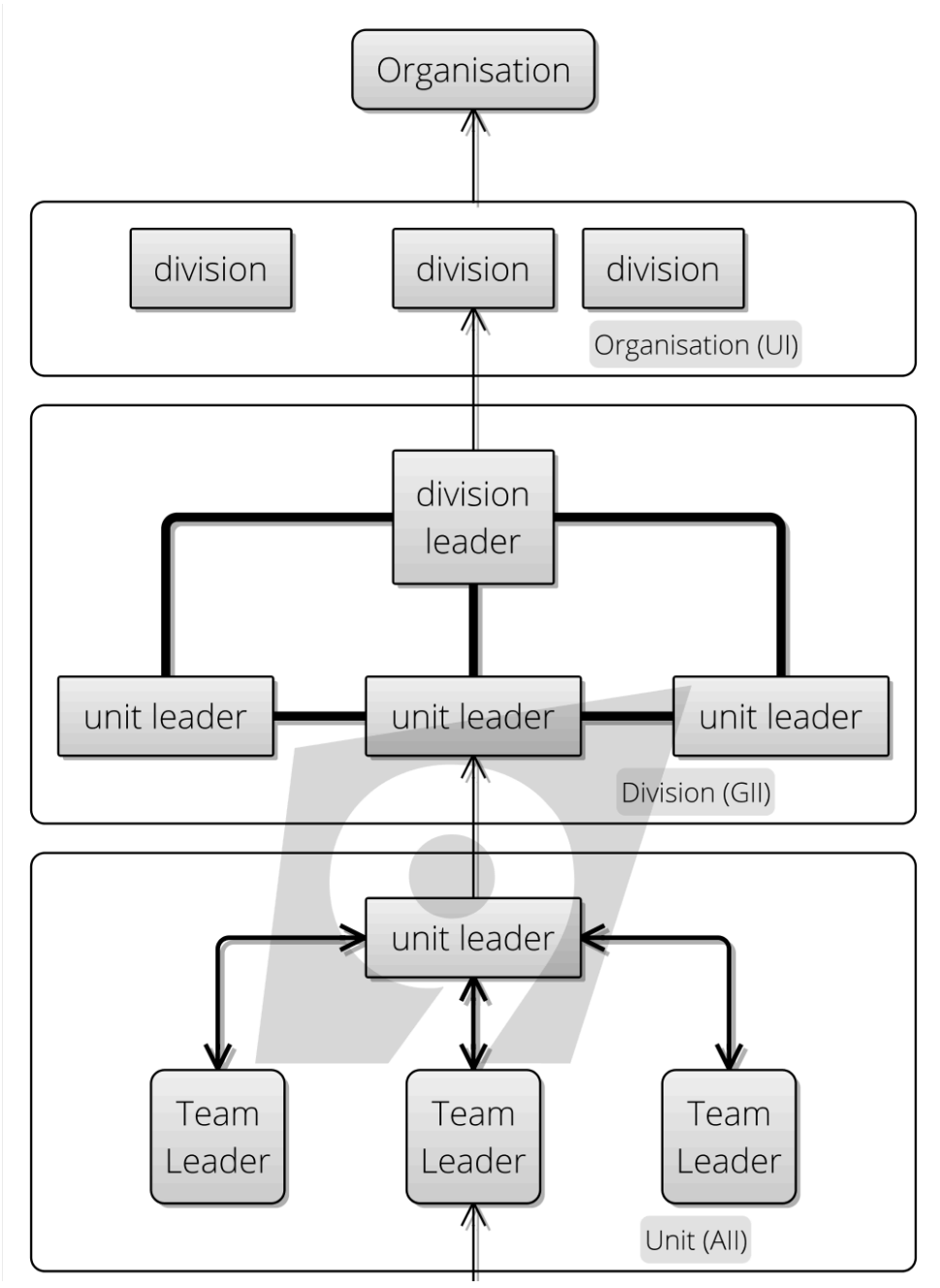


Figure 1.5: Hierarchy of Function in an Organisation

In large organisations, teams are created and leaders chosen for those teams precisely to mitigate Communication Risk. We're all familiar with this: control of the team is ceded to the leader, who takes on the role of 'handing down' direction from above, but also 'reporting up' issues that cannot be resolved within the team. In Vroom and Yetton's model, this is moving from a **GII** or **CII** to an **AI** or **AII** style of leadership.

As shown in Figure 1.5, we end up with a hierarchy of groups, each having it's own decision-making style. The team leader at the bottom level is a *decision maker* within his team, but moving up, doesn't have decision making power in the next team up.. and so on.

Sometimes, parts of an organisation are encouraged *not* to coordinate, but to compete. In Figure 1.5, we have an M-Form[25] organisation, composed of *competing divisions*.

Clearly, this is just a *model*, it's not set in stone and decision making styles usually change from day-to-day and decision to decision. The same is not true in our software - *rules are rules*.

1.6 In Software Processes

It should be pretty clear that we are applying the Scale Invariance rule to Coordination Risk: all of the problems we've described as affecting teams, also affect software, although the scale and terrain are different. Software processes have limited *agency* - in most cases they follow fixed rules set down by the programmers, rather than self-organising like people can (so far).

As before, in order to face Coordination Risk in software, we need multiple agents all working together. Coordination Risks (such as race conditions or deadlock) only really occurs where *more than one thing is happening at a time*. This means we are considering *at least* multi-threaded software and anything above that (multiple CPUs, servers, data-centres and so on).

CAP Theorem

The CAP Theorem[26] has a lot to say about Coordination Risk. Imagine talking to a distributed database, where your request (*read* or *write*) can be handled by one of many agents.

In Figure 1.6, we have just two agents `1` and `2`, in order to keep things simple. `User A` *writes something* to the database, then `User B` *reads it back* afterwards.

[25] `https://en.wikipedia.org/wiki/Multi-divisional_form`

[26] `https://en.wikipedia.org/wiki/CAP_theorem`

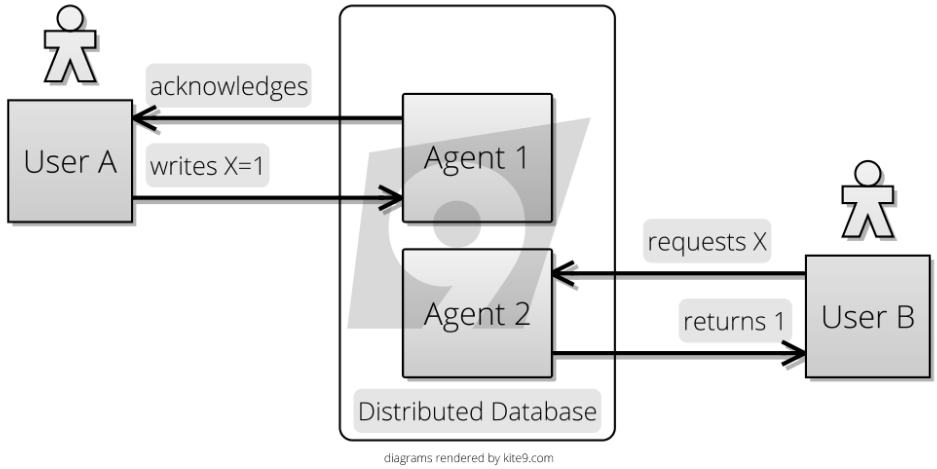


Figure 1.6: User A and User B are both using a distributed database, managed by Agents 1 and 2, whom each have their own Internal Model

According to the CAP Theorem, there are three properties we could desire in such a system:

- **Consistency**: Every read receives the most recent value from the last write.
- **Availability**: Every request receives a response.
- **Partition tolerance**: The system can operate despite the isolation (lack of communication with) some of it's agents.

The CAP Theorem states that this is a Trilemma[27]. That is, you can only have two out of the three properties.

There are plenty of resources on the internet that discuss this in depth, but let's just illustrate with some diagrams to show how this plays out. In our diagram example, we'll say that *any* agent can receive the read or write. So this might be a **GII** decision making system, because all the agents are going to need to coordinate to figure out what the right value is to return for a read, and what the last value written was. In these, the last write (setting X to 1) was sent to Agent 1 which then becomes *isolated*, and can't be communicated with, due to network failure. What will User B get back?

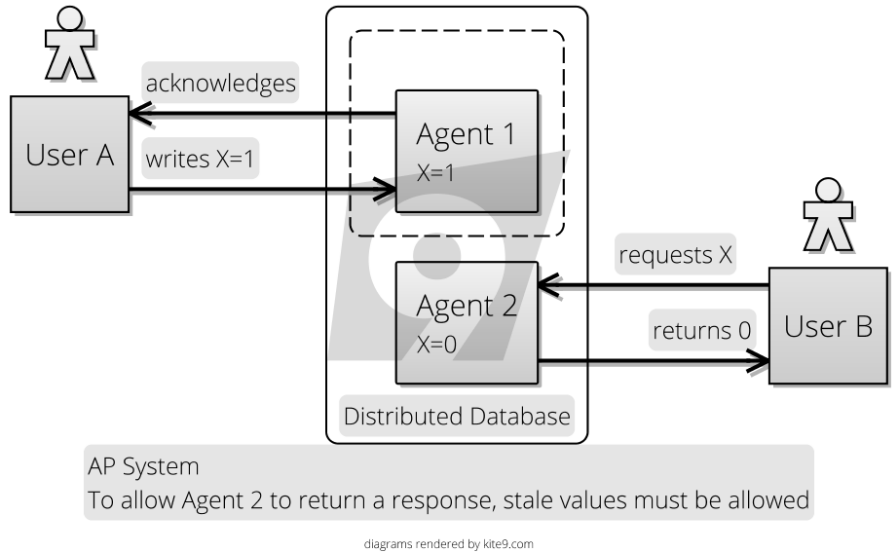


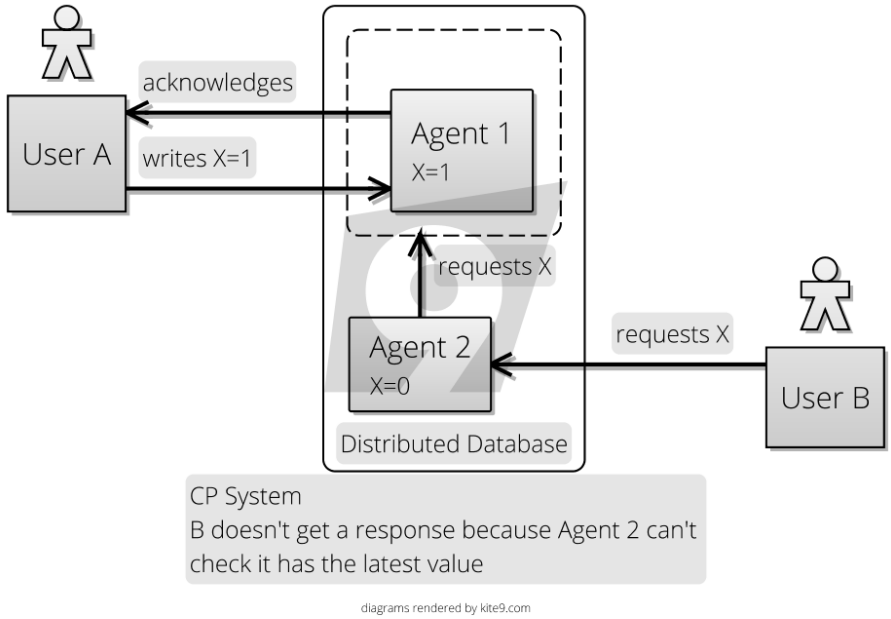
Figure 1.7: In an AP system, the User B will get back a* stale value *for X

With an AP System

With `AP`, you can see that `User B` is getting back a *stale value*. `AP` scenarios lead to Race Conditions: `Agent 1`s availability determines what value `User B` gets back.

[27] `https://en.wikipedia.org/wiki/Trilemma`

With an CP System



.

Where Agent 2 is left waiting for Agent 1 to re-appear, we are *blocked*. So CP systems lead to Deadlock scenarios.

With an CA System

Finally, if we have a CA system, we are essentially saying that *only one agent is doing the work*. (You can't partition a single agent, after all). But this leads to Resource Allocation and **Contention** around use of the scarce resource of `Agent 2`'s attention. (Both Coordination Risk issues we met earlier.)

Some Real-Life Examples

This sets an upper bound on Coordination Risk: we *can't* get rid of it completely in a software system, -or- a system on any other scale. Fundamentally, coordination problems are inescapable at some level. The best we can do is mitigate it by agreeing on protocols and doing lots of communication.

Let's look at some real-life examples of how this manifests in software.

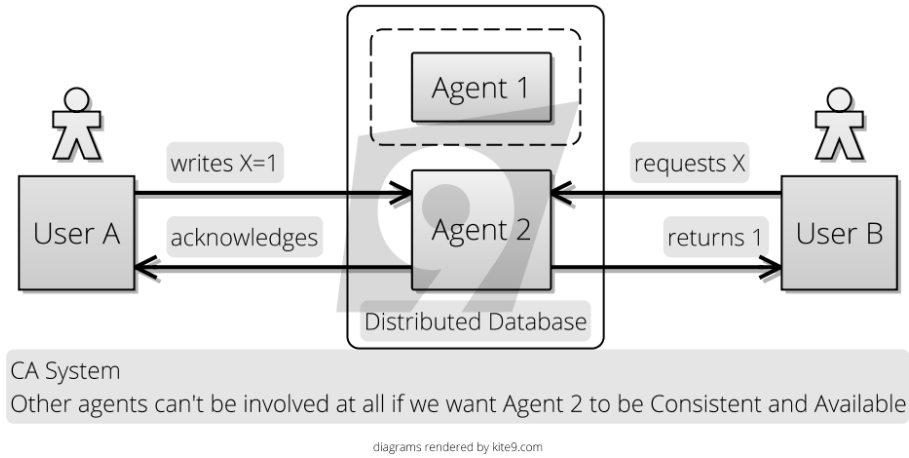


Figure 1.8: In an CA system, we can't have partition tolerance, so in order to be consistent a single Agent has to do all the work

ZooKeeper

First, ZooKeeper[28] is an Open-Source datastore, which is used a lot for coordinating a distributed systems, and storing things like configuration information across them. If the configuration of a distributed system gets changed, it's important that *all of the agents in the system know about it*, otherwise... disaster.

This *seems* trivial, but it quickly gets out-of-hand: what happens if only some of the agents receive the new information? What happens if a datacentre gets disconnected while the update is happening? There are lots of edge-cases.

ZooKeeper handles this by communicating inter-agent with it's own protocol. It elects a **master agent** (via voting), turning it into an **AI**-style team. If the master is lost for some reason, a new leader is elected. *Writes* are then coordinated via the **master agent** who makes sure that a *majority of agents* have received and stored the configuration change before telling the user that the transaction is complete. Therefore, ZooKeeper is a `CP` system.

Git

Second, git is a (mainly) write-only ledger of source changes. However, as we already discussed above, where different agents make incompatible changes, someone has to decide how to resolve the conflicts so that we have a single source of truth.

[28] `https://zookeeper.apache.org`

The Coordination Risk just *doesn't go away*.

Since multiple users can make all the changes they like locally, and merge them later, Git is an `AP` system: individual users may have *wildly* different ideas about what the source looks like until the merge is complete.

Bitcoin

Finally, Bitcoin (BTC)[29] is a write-only distributed ledger[30], where agents *compete* to mine BTC, but also at the same time record transactions on the ledger. BTC is also `AP`, in a similar way to Git. But new changes can only be appended if you have the latest version of the ledger. If you append to an out-of-date ledger, your work will be lost.

Because it's based on outright competition, if someone beats you to completing a mining task, then your work is wasted. So, there is *huge* Coordination Risk.

For this reason, BTC agents coordinate into mining consortia[31], so they can avoid working on the same tasks at the same time. But this in itself is a problem, because the whole *point* of BTC is that it's competitive, and no one entity has control. So, mining pools tend to stop growing before they reach 50% of the BTC network's processing power. Taking control would be politically disastrous[32] and confidence in the currency (such as there is) would likely be lost.

1.7 Communication Is For Coordination

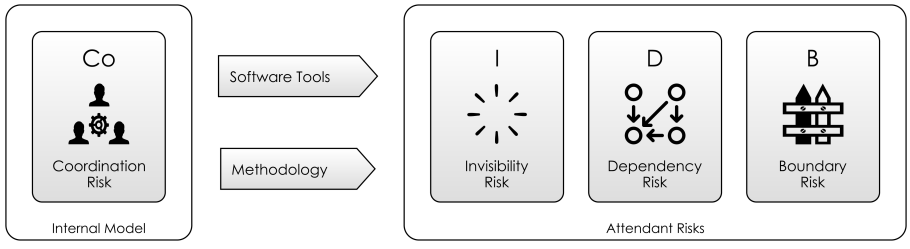


Figure 1.9: Coordination Risk - Mitigated by Software Tools

[29] `https://en.wikipedia.org/wiki/Bitcoin`

[30] `https://en.wikipedia.org/wiki/Distributed_ledger`

[31] `https://en.bitcoin.it/wiki/Comparison_of_mining_pools`

[32] `https://www.reddit.com/r/Bitcoin/comments/5fe9vz/in_the_last_24hrs_three_mining_pools_have_control/`

So, now we have a fundamental limit on how much Coordination Risk we can mitigate. And, just as there are plenty of ways to mitigate Coordination Risk within teams of people, organisations or living organisms, so it's the case in software.

Earlier in this chapter, we questioned whether Coordination Risk was just another type of Communication Risk. However, it should be clear after looking at the examples of competition, cellular life and Vroom and Yetton's Model that this is exactly *backwards*:

- Most single-celled life has no need for communication: it simply competes for the available resources. If it lacks anything it needs, it dies.
- There are *no* lines of communication on the **UI** decision-type. It's only when we want to avoid competition, by sharing resources and working towards common goals that we need to communicate.
- Therefore, the whole point of communication *is for coordination*.

In the next chapter, Map And Territory Risk, we're going to look at some new ways in which systems can fail, despite their attempts to coordinate.

Part III

Tools & Practices

Glossary

Abstraction

The process of removing physical, spatial, or temporal details or attributes in the study of objects or systems in order to more closely attend to other details of interest.

Feedback Loop

Internal Model

The most common use for Internal Model is to refer to the model of reality that you or I carry around in our heads. You can regard the concept of Internal Model as being what you *know* and what you *think* about a certain situation.

Obviously, because we've all had different experiences, and our brains are wired up differently, everyone will have a different Internal Model of reality.

Alternatively, we can use the term Internal Model to consider other viewpoints: - Within an organisation, we might consider the Internal Model of a *team of people* to be the shared knowledge, values and working practices of that team. - Within a software system, we might consider the Internal Model of a single processor, and what knowledge it has of the world. - A codebase is a team's Internal Model written down and encoded as software.

An internal model *represents* reality: reality is made of atoms, whereas the internal model is information.

Taking Action

Meet Reality

Pay-Off

Risk

Risk Landscape

Goal In Mind

Initial Risk

Attendant Risk

Hidden Risk

Mitigated Risk