Quick Summary

1. There are Lots of Ways of Running Software Projects

There are lots of different ways to look at a project in-flight. For example, metrics such as "number of open tickets", "story points", "code coverage" or "release cadence" give us a numerical feel for how things are going and what needs to happen next. We also judge the health of projects by the practices used on them, such as Continuous Integration, Unit Testing or Pair Programming.

Software methodologies, then, are collections of tools and practices: "Agile", "Waterfall", "Lean" or "Phased Delivery" all prescribe different approaches to running a project, and are opinionated about the way they think projects should be done and the tools that should be used.

None of these is necessarily more "right" than another- they are suitable on different projects at different times.

A key question then is: **how do we select the right tools for the job?**

2. We can Look at Projects in Terms of Risks

One way to examine the project in-flight is by looking at the risks it faces.

Commonly, tools such as RAID logs[1] and RAG status[2] reporting are used. These techniques should be familiar to project managers and developers everywhere.

However, the Risk-First view is that we can go much further: that each item of work being done on the project is to manage a particular risk. Risk isn't something that just appears in a report, it actually drives *everything we do*.

For example:

[1] `https://www.projectmanager.com/blog/raid-log-use-one`

[2] `https://pmtips.net/blog-new/what-does-rag-status-mean`

- A story about improving the user login screen can be seen as reducing *the risk of users not signing up*.
- A task about improving the health indicators could be seen as mitigating *the risk of the application failing and no-one reacting to it*.
- Even a task as basic as implementing a new function in the application is mitigating *the risk that users are dissatisfied and go elsewhere*.

One assertion of Risk-First is that **every action you take on a project is to manage a risk.**

3. We Can Break Down Risks on a Project Methodically

Although risk is usually complicated and messy, other industries have found value in breaking down the types of risks that affect them and addressing them individually.

For example:

- In manufacturing, *tolerances* allow for calculating the likelihood of defects in production.
- In finance, projects and teams are structured around monitoring risks like *credit risk*, *market risk* and *liquidity risk*.
- *Insurance* is founded on identifying particular risks and providing financial safety-nets for when they occur, such as death, injury, accident and so on.

Software risks are difficult to quantify, and mostly, the effort involved in doing so *exactly* would outweigh the benefit. Nevertheless, there is value in spending time building *classifications of risk for software*. That's what Risk-First does: it describes a set of *risk patterns* we see every day on software projects.

With this in place, we can:

- Talk about the types of risks we face on our projects, using an appropriate language.
- Anticipate Hidden Risks that we hadn't considered before.
- Weigh the risks against each other, and decide which order to tackle them.

4. We can Analyse Tools and Techniques in Terms of how they Manage Risk

If we accept the assertion above that *all* the actions we take on a project are about mitigating risks, then it stands to reason that the tools and techniques available to us on a project are there for mitigating different types of risks.

For example:

- If we do a Code Review, we are partly trying to minimise the risks of bugs slipping through into production, and also manage the Key-Man Risk of knowledge not being widely-enough shared.
- If we write Unit Tests, we're addressing the risk of bugs going to production, but we're also mitigating against the risk of *regression*, and future changes breaking our existing functionality.
- If we enter into a contract with a supplier, we are mitigating the risk of the supplier vanishing and leaving us exposed. With the contract in place, we have legal recourse against this risk.

From the above examples, it's clear that **different tools are appropriate for managing different types of risks.**

5. Different Methodologies are for Different Risk Profiles

In the same way that our tools and techniques are appropriate to dealing with different risks, the same is true of the methodologies we use on our projects. We can use a Risk-First approach to examine the different methodologies, and see which risks they address.

For example:

- **Agile** methodologies prioritise the risk that requirements capture is complicated, error-prone and that requirements change easily.
- **Waterfall** takes the view that development effort is an expensive risk, and that we should build plans up-front to avoid re-work.
- **Lean** takes the view that risk lies in incomplete work and wasted work, and aims to minimise that.

Although many developers have a methodology-of-choice, the argument here is that there are tradeoffs with all of these choices.

"Methodologies are like *bicycles*, rather than *religions*. Rather than simply *believing*, we can take them apart and see how they work. "

6. We can Drive Development With a Risk-First Perspective

We have described a model of risk within software projects, looking something like this:

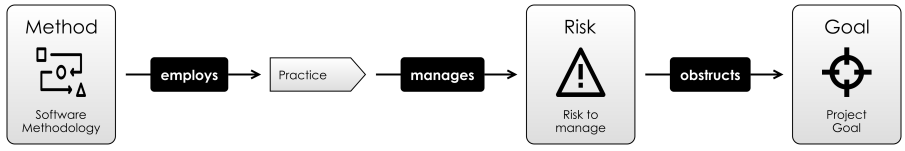

Figure 1: Methodologies, Risks, Practices

How do we take this further?

One idea explored is the *Risk Landscape*: Although the software team can't remove risk from their project, they can take actions that move them to a place in the Risk Landscape where the risks on the project are more favourable than where they started.

From there, we examine basic risk archetypes you will encounter on the software project, to build up a Taxonomy of Software Risk, and look at which specific tools you can use to mitigate each kind of risk.

Then, we look at different software practices, and how they manage various risks. Beyond this we examine the question: *how can a Risk-First approach inform the use of this practice?*

For example:

- If we are introducing a **Sign-Off** in our process, we have to balance the risks it *mitigates* (coordination of effort, quality control, information sharing) with the risks it *introduces* (delays and process bottlenecks).
- If we build in **Redundancy**, this mitigates the risk of a *single point of failure*, but introduces risks around *synchronizing data* and *communication* between the systems.

- If we introduce **Process**, this may make it easier to *coordinate as a team* and *measure performance* but may lead to bureaucracy, focusing on the wrong goals or over-rigid interfaces to those processes.

Risk-First aims to provide a framework in which we can *analyse these actions* and weigh up *accepting* versus *mitigating* risks.

Still interested? Then dive into reading the introduction.

Part I

Introduction

Part II

The Risk Landscape

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.

Agent

Agency is the capacity of an actor to act in a given environment. We use the term *agent* to refer to any process, person, system or organisation with agency.

Feedback Loop

The process of testing an Internal Model by testing it, through taking action to Meet Reality. Typically, we talk about short or long feedback loops, depending on the intervals between Meeting Reality.

Goal In Mind

A picture of the future that an individual or team carries within their Internal Model; An imagined destination on the Risk Landscape.

Internal Model

The model of reality held by an individual, team, software system or other Agent. You can regard the concept of Internal Model as being what you *know* and what you *think* about a certain situation. An internal model *represents* reality: reality is made of matter, whereas the internal model is information.

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

- 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 server, and what knowledge it has of the world.
- A codebase is a team's Internal Model written down and encoded as software.

Meet Reality

Any moment where we test an Internal Model by exposing it's predictive power against reality. Note that "Reality" might be limited in some way, for example, a trial period or test users.

Pay-Off

Pay-Off refers to the *value* of the actions we take. When we decide on a course of action, we have in mind a risk we wish to manage. If the action is likely to have a big positive effect on the risk of a project, we say it has a promising pay off, whereas if the action fails to manage the risk, then it hasn't *paid off*.

Risk

A possibility of loss or cost. Anything that *can* go wrong on a project, or is *going* wrong, but so far hasn't been quantified. We talk about risk because we wish to recognise both the range of possibilities and the range of cost.

Usually broken down into:

- **Attendant Risk**: A Risk you expect to face as the result of Taking Action.
- **Hidden Risk**: Risks you aren't aware of when you consider Taking Action. i.e. an *unknown unknown*.
- **Mitigated Risk**: Risks that, as a result of Taking Action have been minimized.
- **Upside Risk**: The possibility of things going well, and leaving us with a benefit. We may take action to maximize the likelihood and return of upside risks.

Risk Landscape

A hypothetical landscape on which risks can be placed. Taking Action means making a move on the Risk Landscape to reposition a project so that it has a different profile of Attendant Risks.

Taking Action

Refers to any activity in the project. Actions are usually in order to manage some kind of risk. At the same time, Taking Action usually means interacting with reality and updating the Internal Model.

Glossary Of Risk Types

Risk	Definition
Boundary	Risks due to the choices we make around dependencies, and the limitations they place on our ability to change.
Agency	Risks due to the fact that things you depend on have agency, and they have their own goals to pursue.
Channel	Risks due to the inadequacy of the physical channel used to communicate our messages. e.g. noise, loss, interception, corruption.
Communication	Risks due to the difficulty of communicating with other entities, be they people, software, processes etc.
Codebase	The specific risks to a project of having a large, complex codebase to manage.
Complexity	Risks caused by the weight of complexity in the systems we create, and their resistance to change and comprehension.
Conceptual-integrity	Risk that the software you provide is too complex, or doesn't match the expectations of your clients' internal models.
Coordination	Risks that a group of agents cannot work together in a mutually beneficial way, and their behaviour devolves into competition.
Dead-End	The risk that a particular approach to a change will fail. Caused by the fact that at some level, our internal models are not a complete reflection of reality.
Deadline	Where the use of a dependency has some kind of deadline, which can be missed.
Dependency	Risks faced by depending on something else. e.g. an event, process, person, piece of software or an organisation.
Feature-Access	Risks due to some clients not having access to some or all of the features in your product.
Feature-Drift	Risk that the features required by clients will change and evolve over time.
Feature	Risks you face when providing features for your clients.
Feature-Fit	Risk that the needs of the client don't coincide with services provided by the supplier.
Funding	A particular scarcity risk, due to lack of funding.

Risk	Definition
Implementation	Risk that the functionality you are providing doesn't match the features the client is expecting, due to poor or partial implementation.
Internal-Model	Risks to communication caused by the fact that the internal models of the participants are semantically incompatible in some way.
Invisibility	Risks caused by the choice of abstractions we use in communication.
Learning-Curve	Risks due to the difficulty faced in updating an internal model.
Map-And-Territory	Risks due to the differences between reality and the internal model of reality, and the assumption that they are equivalent.
Market	Risk that the value your clients place on the features you supply will change, over time.
Message	Risks caused by the difficulty of composing and interpreting messages in the communication process.
Operational	Risks of losses or reputational damage caused by failing processes or real-world events.
Opportunity	Risk that a particular set of market conditions.
Process	Risks due to the fact that when dealing with a dependency, we have to follow a particular protocol of communication, which may not work out the way we want.
Protocol	Risks due to the failure of encoding or decoding messages between two parties in communication.
Red-Queen	The general risk that the competitive environment we operate within changes over time.
Regression	Risk that the functionality you provide changes for the worse, over time.
Reliability	Risks of not getting benefit from a dependency due to it's reliability.
Scarcity	Risk of not being able to access a dependency in a timely fashion due to it's scarcity.
Schedule	The aspect of dependency risk related to time.
Security	Agency Risks due to actors from outside the system.
Software Dependency	Dependency Risk due to software dependencies.
Staff	The aspect of dependency risks related to employing people.
Trust-And-Belief	Risk that a party we are communicating with can't be trusted, as it has agency or is unreliable in some other way.