**Risk-First**

**Software Development**

*Volume 1: The Menagerie*

**

Rob Moffat

“You never change things by fighting the existing reality. To change something, build a new model that makes the existing model obsolete.”

—Buckminster Fuller

“We've survived 200,000 years as humans. Don't you think there's a reason why we survived? We are good at risk management.”

—Nassim Nicholas Taleb

**Risk-First: The Menagerie**

By Rob Moffat

Copyright ©2018 Kite9 Ltd.

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law. For permission requests, write to the publisher, addressed “Attention: Permissions Coordinator,” at the address below.

ISBN: 9781717491855

**Credits**

tbd

Cover Images: Biodiversity Heritage Library. Biologia Centrali-Americana. Insecta. Rhynchota. Hemiptera-Homoptera. Volume 1 (1881-1905)

Cover Design By P. Moffat (peter@petermoffat.com)

Thanks to:

**Books In The Series**

* **Risk-First: The Menagerie:** Book one of the Risk-First series argues the casefor viewing *all* of the activities on a software project through the lens of *managing risk*. It introduces the menagerie of different risks you’re likely toencounter on a software project, naming and classifying them so that we can try to understand them better.
* **Risk-First: Tools and Practices:** Book two of the Risk-First series explores therelationship between software project risks and the tools and practices we use to mitigate them. It is due for publication in 2020.

**Online**

Material for the books is freely available to read, drawn from risk-first.org.

**Published By**

Kite9 Ltd.

14 Manor Close

Colchester

CO6 4AR

i

**Contents**

|  |  |
| --- | --- |
| **Contents** | **ii** |
| **Preface** | **v** |
| **Quick Summary** | **xi** |

|  |  |  |
| --- | --- | --- |
| **I** | **Introduction** | **1** |
| **1** | **A Simple Scenario** | **3** |
| **2** | **Development Process** | **7** |
| **3** | **Meeting Reality** | **15** |
| **4** | **All Risk Management** | **23** |
| **5** | **Evaluating Risk** | **29** |
| **6** | **Cadence** | **43** |
| **7** | **De-Risking** | **47** |
| **8** | **A Conversation** | **53** |
| **9** | **One Size Fits No One** | **57** |

|  |  |  |
| --- | --- | --- |
| **II** | **The Risk Landscape** | **65** |
| **10** | **Risk Landscape** | **67** |
| **11** | **Feature Risk** | **73** |

ii

|  |  |  |
| --- | --- | --- |
| **12** | **Communication Risk** | **83** |
| **13** | **Complexity Risk** | **103** |
| **14** | **Dependency Risk** | **119** |
| **15** | **Scarcity Risk** | **127** |
| **16** | **Deadline Risk** | **135** |
| **17** | **Software Dependency Risk** | **139** |
| **18** | **Process Risk** | **157** |
| **19** | **Boundary Risk** | **167** |
| **20** | **Agency Risk** | **183** |
| **21** | **Coordination Risk** | **191** |
| **22** | **Map And Territory Risk** | **207** |
| **23** | **Operational Risk** | **221** |
| **24** | **Staging And Classifying** | **231** |

|  |  |  |
| --- | --- | --- |
| **III Tools & Practices** | | **239** |
| **25** | **Coming Next** | **241** |
| **26** | **Estimates** | **243** |
| **Glossary** | | **259** |
| **Index** | | **263** |

iii

**Preface**

Welcome to Risk-First!

Let’s cover some of the big questions up-front: The why, what, who, how, and where of *The Menagerie*.

**Why**

“Scrum, Waterfall, Lean, Prince2: what do they all have in common?”

I’ve started this because, on my career journey, I’ve noticed that the way I do things doesn’t seem to match up with the way the books *say* it should be done. And, I found this odd and wanted to explore it further. Hopefully, you, the reader, will find something of use in this.

I started with this observation: *Development Teams* put a lot of faith in methodology. Sometimes, this faith is often so strong that it borders on religion. (This in itself is a concern.) For some, this is Prince2. For others, it might be Lean or Agile.

*Developers* put a lot of faith in *particular tools* too. Some developers are pro-or-anti-Java; others are pro-or-anti-XML. All of them have their views coloured by their *experiences* (or lack of) with these tools. Was this because their past projects *succeeded* or *failed* because of them?

As time went by, I came to see that the choice of methodology, process, or tool was contingent on the problem being solved and the person solving the problem. We don’t face a shortage of tools in IT, or a shortage of methodologies, or a shortage of practices. Essentially, that all the tools and methodologies that the industry had supplied were there to help *minimize the risk of my project* *failing*.

This book considers that perspective: that building software is all about *managing risk*, and that these methodologies are acknowledgements of this

v

fact and they differ because they have *different ideas* about which are the most important *risks to manage*.

**What This Is**

Hopefully, after reading this, you’ll come away with:

* An appreciation of how risk underpins everything we do as developers, whether we want it to or not.
* A framework for evaluating methodologies, tools, and practices and choosing the right one for the task-at-hand.
* A recontextualization of the software process as being an exercise in mitigating different kinds of risk.
* The tools to help you decide when a methodology or tool is *letting you* *down*, and the vocabulary to argue for when it’s a good idea to deviatefrom it.

This is not intended to be a rigorous scientific work: I don’t believe it’s possible to objectively analyze a field like software development in any meaningful, statistically significant way. (For one, things just change too fast.)

“I have this Pattern”

—Attributed to Ward Cunningham, *Have This Pattern, C2 Wiki*1

Does that diminish it? If you have visited the TVTropes2 website, you’ll know that it’s a set of web-pages describing *common patterns* of narrative, production, character design etc. to do with fiction. For example:

“Sometimes, at the end of a Dream Sequence or an All Just a Dream episode, after the character in question has woken up and demonstrated any [lesson] that the dream might have been communicating, there’s some small hint that it wasn’t a dream after all, even though it quite obviously was. . . right?.”

—Or Was It a Dream?, *TVTropes*3



1. http://c2.com/ppr/wiki/WikiPagesAboutWhatArePatterns/HaveThisPattern.html
2. https://tvtropes.org
3. https://tvtropes.org/pmwiki/pmwiki.php/Main/OrWasItADream

vi

Is it scientific? No. Is it correct? Almost certainly. TVTropes is a set of *empirical patterns* for how stories on TV and other media work. It’s reallyuseful, and is a lot of fun. (Warning: it’s also incredibly addictive).

In the same way, “Design Patterns: Elements of Reusable Object-Oriented Software4”, is a book detailing patterns of *structure* within Object-Oriented programming, such as:

“[The] Adapter [pattern] allows classes with incompatible interfaces to work together by wrapping its own interface around that of an already existing class. . . ”

—Design Patterns, *Wikipedia*5

**Patterns For Practitioners**

Design Patterns aims to be a set of *useful* patterns which practitioners could use in their software to achieve certain goals. “I have this pattern” was a phrase used to describe how they had seen a certain set of constraints before, and how they had solved it in software.

This book was a set of experts handing down their battle-tested practices for other developers to use, and, whether you like patterns or not, knowing them is an important part of being a software developer, as you will see them used everywhere you go and probably use them yourself.

In the same way, Risk-First aims to be a set of *Patterns for Software Risk*. Hopefully, after reading this book, you will see where risk hides in software projects, and have a name for it when you see it.

**Towards a “Periodic Table”**

In the latter chapters of “The Menagerie”, we try to assemble these risk patterns into a cohesive whole. Projects fail because of risks, and risks arise from predictable sources.

**What This is Not**

This is not intended to be a rigorous scientific work: I don’t believe it’s possible to objectively analyze a field like software development in any meaningful, statistically significant way. (For one, things just change too fast.)



1. http://amzn.eu/d/3cOwTkH
2. https://en.wikipedia.org/wiki/Design\_Patterns

vii

Neither is this site going to be an exhaustive guide of every possible software development practice and methodology. That would just be too long and tedious.

Neither is this really a practitioner’s guide to using any particular methodology: If you’ve come here to learn the best way to do Retrospectives, then you’re in the wrong place. There are plenty of places you can find that information already. Where possible, this site will link to or reference concepts on Wikipedia or the wider internet for further reading on each subject.

**Who**

This work is intended to be read by people who work on software projects, and especially those who are involved in managing software projects.

If you work collaboratively with other people in a software process, you should find Risk-First a useful lexicon of terms to help describe the risks you face.

But here’s a warning: This is going to be a depressing book to read. It is book one of a two-book series, but in **Book One,** you only get to meet the bad guy.

While **Book Two** is all about *how to succeed*. This book is all about how projects *fail*. In it, we’re going to try and put together a framework for understandingthe risk of failure, so that we can reconstruct our understanding of our activities on a project based on avoiding it.

So, if you are interested in *avoiding your project failing*, this is probably going to be useful knowledge.

**For Developers**

Risk-First is a tool you can deploy to immediately improve your ability to plan your work.

Frequently, as developers, we find software methodologies “done to us” from above. Risk-First is a toolkit to help *take apart* methodologies like Scrum, Lean and Prince2, and understand them. Methodologies are *bicycles*, rather than *religions*. Rather than simply *believing*, we can take them apart and see how they work.

**For Project Managers and Team Leads**

All too often, Project Managers don’t have a full grasp of the technical details of their projects. And this is perfectly normal, as the specialization belongs

viii

below them. However, projects fail because risks materialize, and risks materialize because the devil is in those details.

This seems like a lost cause, but there is hope: the ways in which risks materialize on technical projects is the same every time. With Risk-First, we attempt to name each of these types of risk, which allows for a dialog with developers about which risks they face and the order in which they should be tackled.

Risk-First allows a project manager to pry open the black box of development and talk with developers about their work, and how it will affect the project. It is another tool in the (limited) arsenal of techniques a project manager can bring to bear on the task of delivering a successful project.

**How**

One of the original proponents of the Agile Manifesto, Kent Beck, begins his book Extreme Programming by stating:

“It’s all about risk”

—Kent Beck, *Extreme Programming Explained* 6

This is a promising start. From there, he introduces his methodology, Extreme Programming, and explains how you can adopt it in your team, the features to observe, and the characteristics of success and failure. However, while *Risk* has clearly driven the concept of Extreme Programming, there is no clear model of software risk underpinning the work, and the relationship between the practices he espouses and the risks he is avoiding are hidden.

In this book, we are going to introduce a model of software project risk. This means that in **Book Two** (Risk-First: Tools and Practices), we can properly analyse Extreme Programming (and Scrum, Waterfall, Lean, and all the others) and *understand* what drives them. Since they are designed to deliver successful software projects, they must be about managing risks, and we will uncover *exactly which risks* and *how they do it*.



1. http://amzn.eu/d/gUQjnbF

ix

**Where**

All of the material for this book is available Open Source on github.com 7, and at the risk-first.org8 website. Please visit, your feedback is appreciated.

There is no compulsion to buy a print or digital version of the book, but we’d really appreciate the support. So, if you’ve read this and enjoyed it, how about buying a copy for someone else to read?

**A Note on References**

Where possible, references are to the Wikipedia9 website. Wikipedia is not perfect. There is a case for linking to the original articles and papers, but using Wikipedia references are free and easy for everyone to access, and hopefully, will exist for a long time into the future.

On to The Quick Summary



1. https://github.com
2. https://risk-first.org
3. https://wikipedia.org

x

**Quick Summary**

**1. There are Lots of Ways to 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 of 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 for 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 10 and RAG status11 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

xi

* 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 address 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.

xii

1. **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.**

1. **Different Methodologies are for Different Risk Profiles**

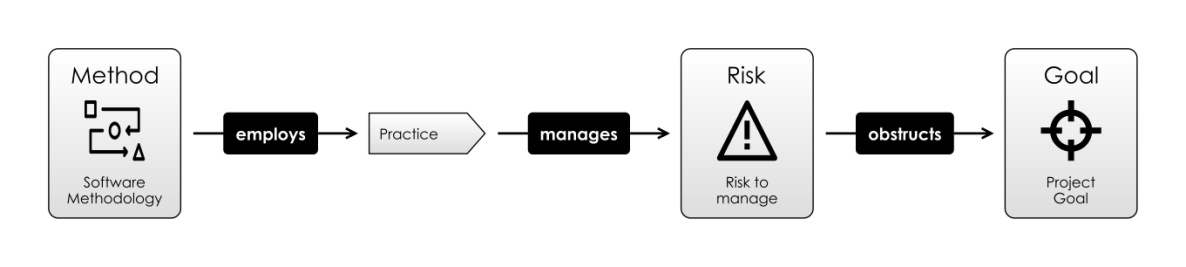
In the same way that our tools and techniques are appropriate for dealing with different risks, the same is true for the methodologies we use on our projects. We can use a Risk-First approach to examine the different methodologies and see which risks they address.

For example:

* **Agile** methodologies prioritise the risk that requirements capture iscomplicated, error-prone, and that requirements change easily.
* **Waterfall** takes the view that development effort is an expensive risk,and that we should build plans upfront 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.

xiii



***Figure 1: Methodologies, Risks, Practices***

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

1. **We can Drive Development With a Risk-First Perspective**

We have described a model of risk within software projects which looks like something similar to this:

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.

xiv

* 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.**

xv

**Part I**

**Introduction**

1

CHAPTER 1



**A Simple Scenario**

In this chapter, I’m going to introduce some terms for thinking about risk.

Let’s for a moment forget about software completely and think about *any* *endeavour at all* in life. It could be passing a test, mowing the lawn or goingon holiday. Choose something now. I’ll discuss from the point of view of “cooking a meal for some friends”, but you can play along with your own example.

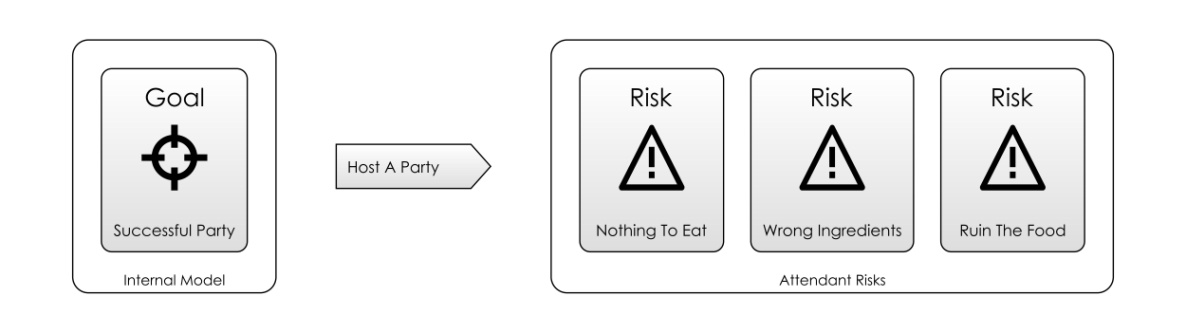
**1.1** **Goal In Mind**

Now, in this endeavour, we want to be successful. That is to say, we have a **Goal In Mind**: we want our friends to go home satisfied after a decent meal,and not to feel hungry. As a bonus, we might also want to spend time talking with them before and during the meal. So, now to achieve our Goal In Mind, we *probably* have to do some tasks.

Since our goal only exists *in our head*, we can say it is part of our **Internal** **Model** of the world. That is, the model we have of reality. This model extendsto *predicting what will happen*.

If we do nothing, our friends will turn up and maybe there’s nothing in the house for them to eat. Or maybe, the thing that you’re going to cook is going to take hours and they’ll have to sit around and wait for you to cook it and they’ll leave before it’s ready. Maybe some ingredients would not be sufficient enough, or maybe you’re not confident of the steps to prepare the meal and you’re worried about messing it all up.

3



***Figure 1.1: Goal In Mind, with the risks you know about***

**1.2** **Attendant Risk**

These *nagging doubts* that are going through your head are what I’ll call the Attendant Risks: they’re the ones that will occur to you as you start to think about what will happen.

When we go about preparing for this wonderful evening, we can choose to deal with these risks: shop for the ingredients in advance, prepare parts of the meal, and maybe practice the cooking in advance. Or, we can wing it, and sometimes we’ll get lucky.

How much effort we expend on these Attendant Risks depends on how big we think they are. For example, if you know there’s a 24-hour shop, you’ll probably not worry too much about getting the ingredients well in advance (although, the shop *could still be closed*).

**1.3** **Hidden Risks**

There are also **Hidden Risks** that you *don’t* know about: if you’re poaching eggs for dinner, perhaps you don’t know that fresh eggs poach best. The difference is, Attendant Risks are risks you are aware of, but can’t be sure of the amount they will impact you. Hidden Risks are the ones you are unaware of.

Donald Rumsfeld1 famously called these “Unknown Unknowns”.

Different people evaluate risks differently, and they’ll also *know* about different risks. What is an Attendant Risk for one person is a Hidden Risk for another.

Which risks we get to know about depends on our **knowledge** and **experience**, then. And that varies from person to person (or team to team).