Reasoning While We Still Can

Mathias Verraes
Strategy & Design Consultant
Author of “Design & Reality”
verraes.net/axoniq



Thoughtworks’
Future of Software Engineering

Zürich, June 2026

Spectrum of engineering

Vibecoding
AI assisted traditional engineering
Spec-Driven Development
Dark Factory

Spectrum of reading code

Everything
Some
Nothing

I spent the Summer experimenting on these spectrums

The quality of vibe-coding grows faster than the quality of Spec-Driven Dev

* based on my own experiments

Some predict we will no longer need to do any cognitive labour

Until we get there,
we need to manage risks

Until we get there,
we need to manage risks

Harness engineering
Testing
Accountability
Process
...

What’s missing?

Own the Rate of Change

Reasoning While We Still Have To Own the Rate of Change

Mathias Verraes

Strategy & Design Consultant

Author of “Design & Reality”

verraes.net/axoniq

Make an Asymmetric Bet

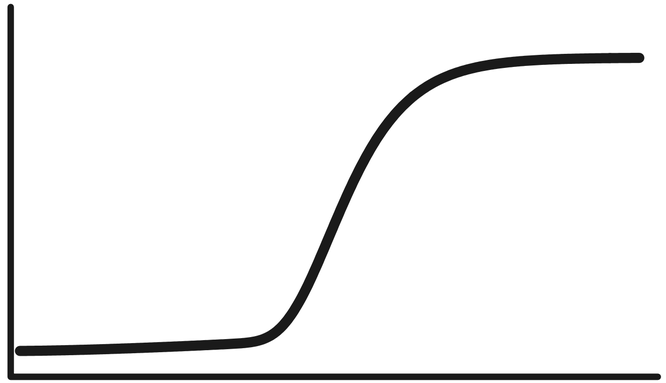
Four scenarios

1. AI destroys us all

2. The post-scarcity society

2. The post-scarcity society

Who’s making that happen?



3. Gradual growth up to a plateau

4. AI as we know it disappears

Externalities (climate, noise pollution)
Data center backlash
Government regulation
Hardware costs
Financial bubble
IP theft

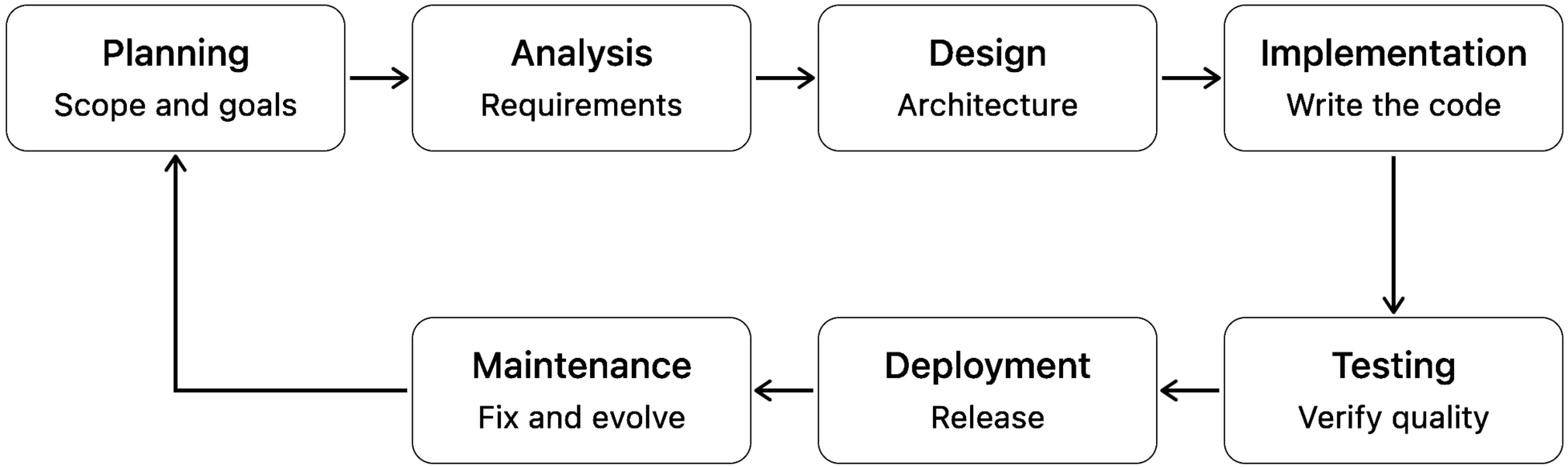
20% 1. AI destroys us all
10% 2. The post-scarcity society
65% 3. Gradual growth up to a plateau
5% 4. AI disappears

* Pick your own numbers

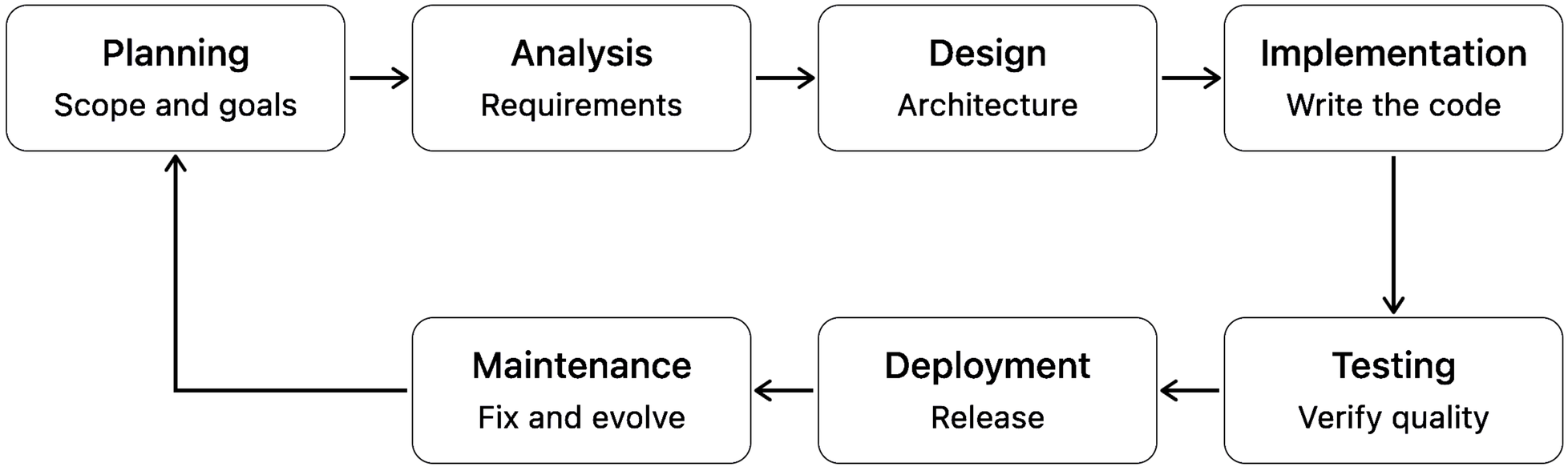
The no-regrets bet

0% 1. AI destroys us all
0% 2. The post-scarcity society
93% 3. Gradual growth up to a plateau
7% 4. AI disappears

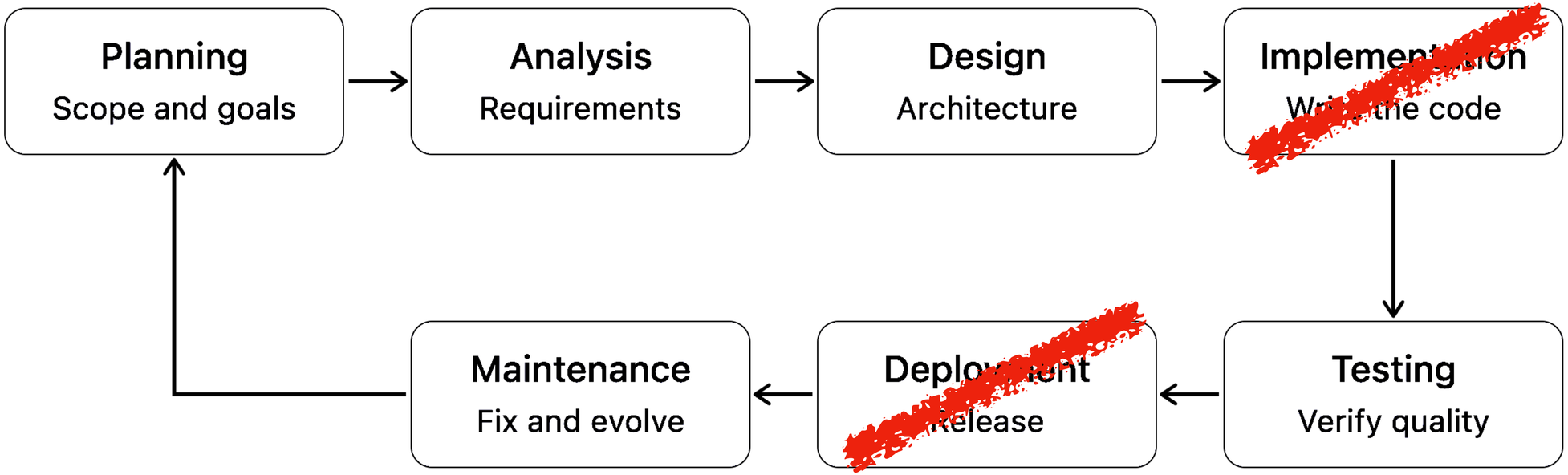
The Software Development Lifecycle



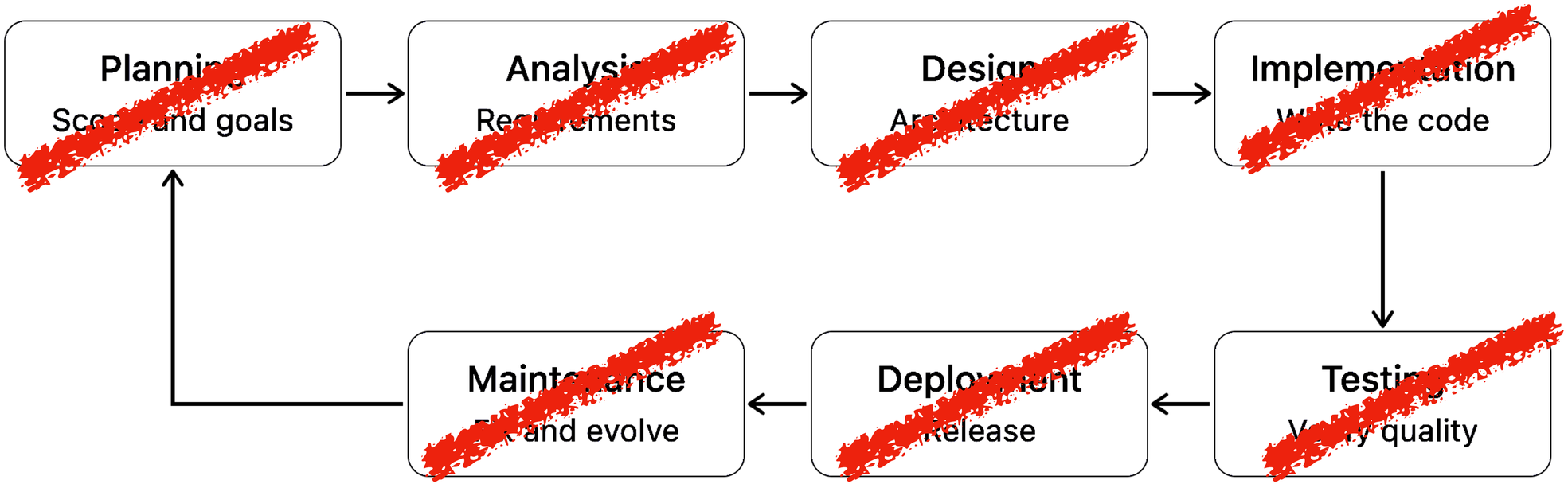
Each phase manages risk



The SDLC collapses…



…entirely


Planning
Scope and goals
Analysis
Requirements
Design
Architecture
Implementation
Write the code
Testing
Verify quality
Deployment
Release
Maintenance
Fix and evolve

The new SDLC?


Specification
Define the intent
Implementation
Write the code
Verification
Check against spec

Risk doesn’t disappear

How do we limit the risk of agents changing artefacts we no longer read?

Own the rate of change

Own the rate of change

Control
which artefacts agents can change,
and when

1. Specs as Single Source of Truth

1. Specs as Single Source of Truth

Requirement specification
Domain model
Ubiquitous Language glossary
Bounded Contexts
Contracts
Worked examples

1. Specs as Single Source of Truth

Test by Regeneration

1. Specs as Single Source of Truth

Test by Regeneration

If I can not regenerate
the same behaviour from the spec,
the truth is still in the code

2. Semi-Deterministic Regeneration

2. Semi-Deterministic Regeneration

Create intermediate artefacts with human-verifiable diffs

2. Semi-Deterministic Regeneration

Create intermediate artefacts with human-verifiable diffs

eg LaTeX/Typst to PDF

3. Change bounded in space

3. Change bounded in space

Bounded Contexts
are natural
Agent Change Boundaries

4. Change bounded in time

4. Change bounded in time

Declare artefacts as stable

4. Change bounded in time

Declare artefacts as stable
aka
Forbid changes

Candidates for stability:

Contracts between bounded contexts
Published API & event schemas
Glossary terms
Hard-won bug fixes
Regulatory logic
…

5. Deterministic tooling to prevent change

5. Deterministic tooling to prevent change

Harness, pre-commit hooks, CI/CD…

6. Non-deterministic tooling to prevent change

6. Non-deterministic tooling to prevent change

Verify the language, boundaries, models

1. Specs as Single Source of Truth
2. Semi-Deterministic Regeneration
3. Change bounded in space
4. Change bounded in time
5. Deterministic change prevention
6. Non-deterministic change prevention

Examples

Plugin architecture

Parsers using formal grammar

Remember this?

7% 4. AI disappears

7. Spend 7% of effort on code quality

7. Spend 7% of effort on code quality

If AI disappears,
you may need to go back
to human engineering again

7. Spend 7% of effort on code quality

The Ironies of Automation (Bainbridge)

7. Spend 7% of effort on code quality

Only the hardest problems will remain

7. Spend 7% of effort on code quality

Only the hardest problems will remain

You'll want code you can understand

Do you want to solve the hardest problems?

Do you want to
solve the hardest problems?

Engineering is human

Reasoning ~~While We~~ Still Can

because it's human

Mathias Verraes

verraes.net/axoniq