Resilient Software Design

The past, the present and the future

Uwe Friedrichsen (codecentric AG) – Software Architecture Gathering – Online, 17. November 2022

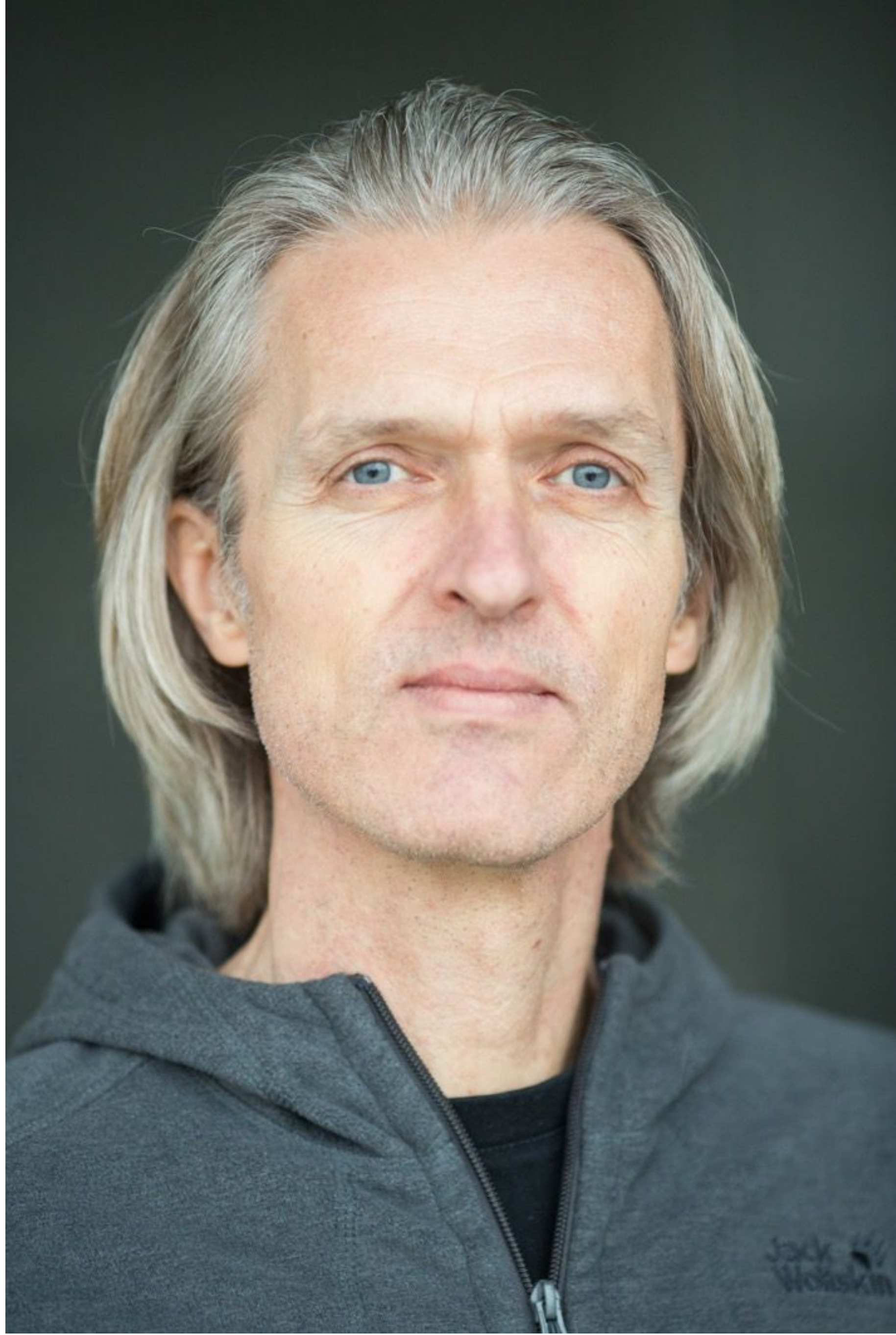
Uwe Friedrichsen

Works @ codecentric

https://twitter.com/ufried

https://www.speakerdeck.com/ufried

https://ufried.com/



The past

Perception of resilience in IT in the past

(and often still in the present)

Resilience = Fault tolerance

Perception of resilience in IT in the past

Fault tolerance (early days)

- Fault tolerance started decades ago
  - SAPO (1950s)
  - NASA LLNM computing (1960s), e.g., for Apollo and Voyager
  - F14 CADC (1970s)
  - Telecommunication switches (1970s)
  - Tandem Computers, Inc. (1970s)
- Fault tolerance typically solved at hardware and OS level
- Software development usually only affected marginally

Fault tolerance (continued)

- Boom with rise of cloud and microservices (early 201x)
  - E.g., Netflix OS (especially Hystrix)
- More software development attention
- Called “resilience”, but still focus on fault-tolerance
- Meanwhile more infrastructure-level support
  - E.g., service meshes, API gateways, cloud infrastructure
- Often neglected for the sake of cost-efficiency

Perception of resilience outside of IT in the past

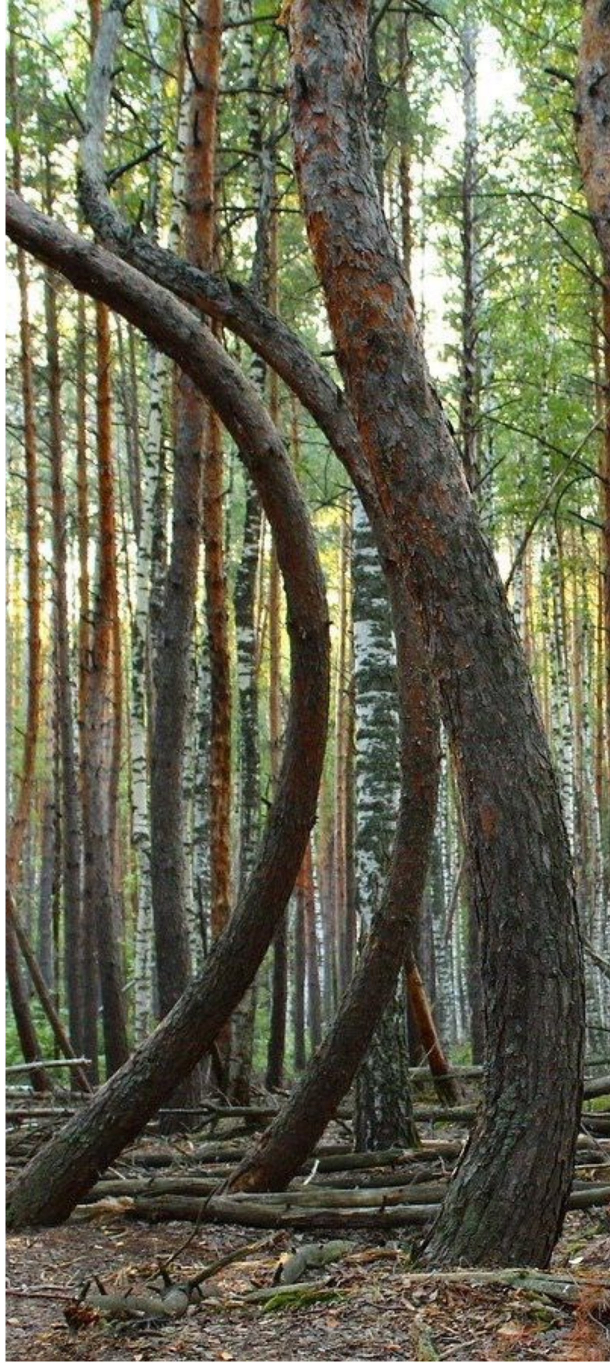
(and still in the present)

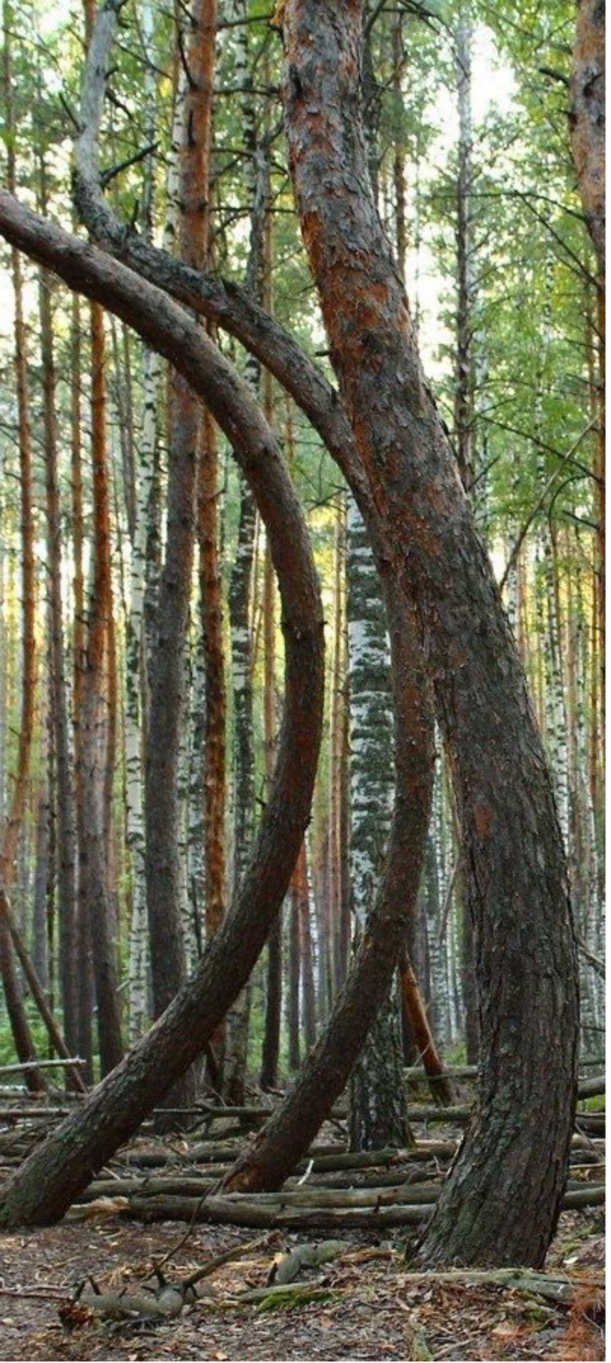
Resilience ≠ Fault tolerance

Perception of resilience outside of IT in the past

Resilience outside of IT

- Multidisciplinary field
    - Psychological resilience
    - Organizational resilience
    - Supply chain resilience
    - Ecological resilience
    - Resilience engineering (safety)
    - Materials resilience
    - Cyber resilience (security)
    - ...





Resilience outside of IT (continued)

- Often different focus
  - Robustness/fault tolerance: Handle known failure modes
  - Resilience: Adapt to unknown failure modes (“surprises”)
- Still, no generally accepted definition
  - Definition depends on the field
  - Sometimes robustness is part of it, sometimes it is not
  - Often related to safety and dependability
- Common ground: Resilience is more than just robustness

Bottom line

Bottom line (past)

- Broad multidisciplinary field
- No generally accepted definition
- Used as synonym for fault-tolerance in IT
- Usually expected to be solved at infrastructure level
- Often ignored for the sake of maximizing cost-efficiency

The present

It became a bit quiet about resilient software design

Microservices became popular, but nothing else changed much ...

Ignoring the effects of distribution

- Architects ignore effects of distribution
- Developers ignore effects of distribution
- Everyone else expects things to become faster and cheaper

- Development expects infrastructure to solve the problems
- Operations curses and is stressed out

You build it. You ignore it.

Build things as you like and neglect the consequences of your acting!

Why is this a problem?

Distributed systems in a nutshell

Everything fails, all the time.

-- Werner Vogels

Effects of distributed systems

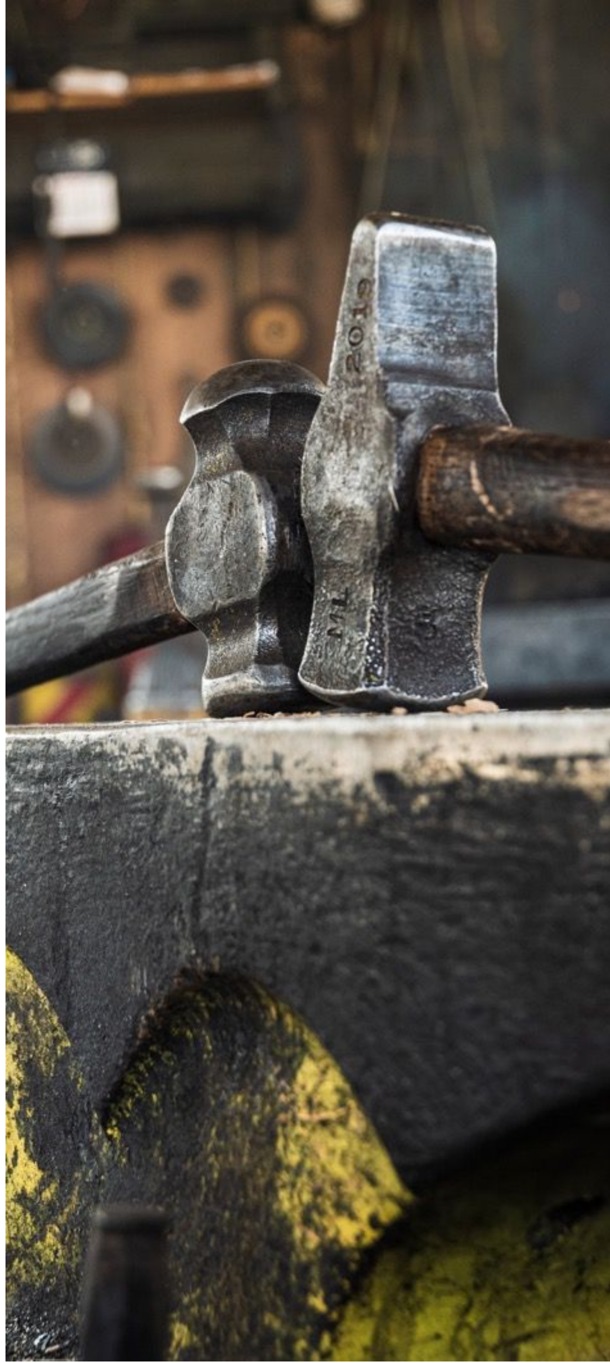
- **Distributed systems introduce non-determinism regarding**
  - **Execution completeness**
  - **Message ordering**
  - **Communication timing**

- You will be affected by this at the application level
  - Don’t expect your infrastructure to hide all effects from you
  - Better know how to detect if it hit you and how to respond

What can the infrastructure do for us in such a setting?

Infrastructure level means

- Detect if a peer does not (timely) respond
- Retry accessing the peer
- Try to access a different instance from failover group
- Try to fire up new instances
  - After instance loss is detected
  - If load exceeds a certain level (“autoscale”)
- Throttle incoming requests
- Notify administrators if additional action is required
- …



This is quite a bit, but …

Infrastructure level limitations

- Not all failure modes supported (e.g., response failures)
- Not all patterns supported (e.g., idempotency, fallback)
- Not ubiquitously available (e.g., on-premises autoscale)
- Often support from application level required (e.g., metrics)
- Only undifferentiated, coarse-grained actions possible

The effects of distribution will still hit you at the application level

Complexity of IT system landscapes grows continually ...

System landscape complexity

- New development projects only focus on local optimization
  - Ignoring impact on complexity of whole system landscape
  - Leads to disproportionate increase in complexity
- New paradigms only focus on their advantages
  - Ignoring effects on complexity of whole system landscape
  - Leads to disproportionate increase in complexity
- Only a matter of time until IT will collapse beyond repair

This requires resilience thinking beyond application robustness

But it also requires more focus on application robustness, i.e., resilient software design

Slowly, companies start to realize that there might be a problem ...

... that their future viability might depend on their resilience

Still, they usually act based on habits and old practice ...

... neglecting building resilience for short-term gains

Bottom line

Bottom line (present)

- Understanding grows that resilience is needed at all levels
- Complexity of IT landscapes has become a problem
- Still, investments are scarce
- “It’s going to be alright” mindset still prevalent

The future

Resilience will become **the** topic of the 21st century

alongside with sustainability

Not because companies want to
but because they do not have a choice

Still, several challenges need to be solved first

Homework that needs to be done

Homework that needs to be done

- Stop fighting about the "right" definition of resilience
  - In the end, all resilience proponents have the same goal
  - Debates about the "right" definition only confuse other people
  - Makes it harder to spread the ideas and their implementation

Here is my suggestion ...

resilience

The ability to successfully cope with adverse events and situations, including

1. handling expected adverse events and situations (robustness)
2. handling unexpected adverse events and situations (surprise)
3. improving due to adverse events and situations (anti-fragility)

resilient software design

Designing and building software-based systems in ways that improve their dependability and thus support resilience according to the definition above

Homework that needs to be done

- Stop fighting about the "right" definition of resilience
- Break traditional company habits
  - Maximizing efficiency cripples resilience
  - Short-term thinking compromises resilience
  - Focus on minimizing short-term development costs compromises resilient software design
  - Huge change of ingrained mindset

Homework that needs to be done

- Stop fighting about the “right” definition of resilience
- Break traditional company habits
- Understand resilience in IT
  - Resilience is a socio-technical topic
  - Cannot be solved at the technical level alone
  - Cannot be solved with tools or products
  - Technology can only support

Homework that needs to be done

- Stop fighting about the “right” definition of resilience
- Break traditional company habits
- Understand resilience in IT
- Understand resilient software design
  - Cannot be solved at the infrastructure level
  - Requires tight ops-dev feedback loops to be effective
  - Without a proper functional design, nothing else matters

Now what?

Some recommendations

- Regarding system design
    - Mind the functional design
    - Strive for functional independence of runtime units
    - Then augment with resilience patterns
    - Domain-driven design can support

Some recommendations

- Regarding system design
- Regarding software landscape grooming
    - Simplify! – Complexity is the enemy of resilience
    - Coordinate infrastructure, application and organization level measures

Some recommendations

- Regarding system design
- Regarding software landscape grooming
- Regarding IT organization and processes
  - Establish short feedback loops across the IT value chain
  - Make resilience a continuous improvement process
  - Include chaos engineering

Some recommendations

- Regarding system design
- Regarding software landscape grooming
- Regarding IT organization and processes
- Regarding product functionality
  - Simplify! –Keep the product simple
  - Regularly remove features that are rarely or not used
  - Implement business metrics

“Perfection is achieved,
not when there is nothing more to add,
but when there is nothing left to take away.”

-- Antoine de Saint-Exupery

Some recommendations

- Regarding system design
- Regarding software landscape grooming
- Regarding IT organization and processes
- Regarding product functionality
- Regarding humans
    - Provide great user experience for all types of users
    - Provide training for all parties along the IT value chain

Summing up

Summing up

- Resilience is huge multidisciplinary topic
    - Started as fault-tolerance in IT
    - Had a little hype a few years ago
- Will become essential topic of the 21st century
    - Much more than fault-tolerance or robustness alone
    - Awareness increases
    - Yet currently little investments
- Lots of homework to be done

Uwe Friedrichsen

Works @ codecentric

https://twitter.com/ufried

https://www.speakerdeck.com/ufried

https://ufried.com/