Martin Lippert

Agile Software Development and Software Architectures

it-agile

martin.lippert@it-agile.de
Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.
What does it mean?
for our daily work
Focus on Business Value

Incremental Development

Inspect & Adapt

Simple Solutions

Changing Requirements

Small Steps

Inspect & Adapt

Shipping

Short Release Cycles

Changing Code all the Time

No Big Upfront Design
We are agile because we don’t care about architecture – it will emerge magically
But you are probably wrong...

FAIL

failblog.com
Instead you live in great danger
Start simple and evolve

the long version

**Gall’s Law:** “A complex system that works is invariably found to have evolved from a simple system that worked. The inverse proposition also appears to be true: A complex system designed from scratch never works and cannot be made to work. You have to start over, beginning with a working simple system.”

– John Gall
How do systems look like in our daily work?
Looks familiar?
Wake up!
We need to change our direction...
Let’s talk about Architecture
Past...
Present...?
Future... ?!!?
But what instead?
Flexibility & Modularity
We need flexibility

changing requirements
learning process
incremental development
But wait!

We already have all this...
We have:

Object-Oriented Patterns
Information Hiding
Encapsulation
Layers

...
We think our systems look like this...
But reality can be hard...
We need a real module system
I. Dependencies

Module A → Module B
II. Visibilities

API Module A

Private Implementation Module A
III. Dynamics
Where do we go?
Loose Coupling & High Cohesion

Think about your dependencies every single day
Sounds good...

But how to realize?
Good old design principles

DIP  SOC  LSP  ADP  TDA  DRY  AIP

ISP  SCP  OCP  IHP  SRP  SDP
new design principles

- Use services
- Use extensions

- Separate between interface and implementation
- working but extensible components
What do we learn?
Guide 1:
Many small modules
instead of few big ones
Guideline 2: Fewer connections between modules instead of everything is wired to everything
Guideline 3:
Less visibilities
instead of making everything public
Guideline 4: Many small frameworks instead of few big ones
Guideline 5: Think about extensibility instead of knowing everything
Guideline 6: Design your architecture every day instead of ignoring what you have learned.
Thank you for your attention

Martin Lippert
martin.lippert@it-agile.de