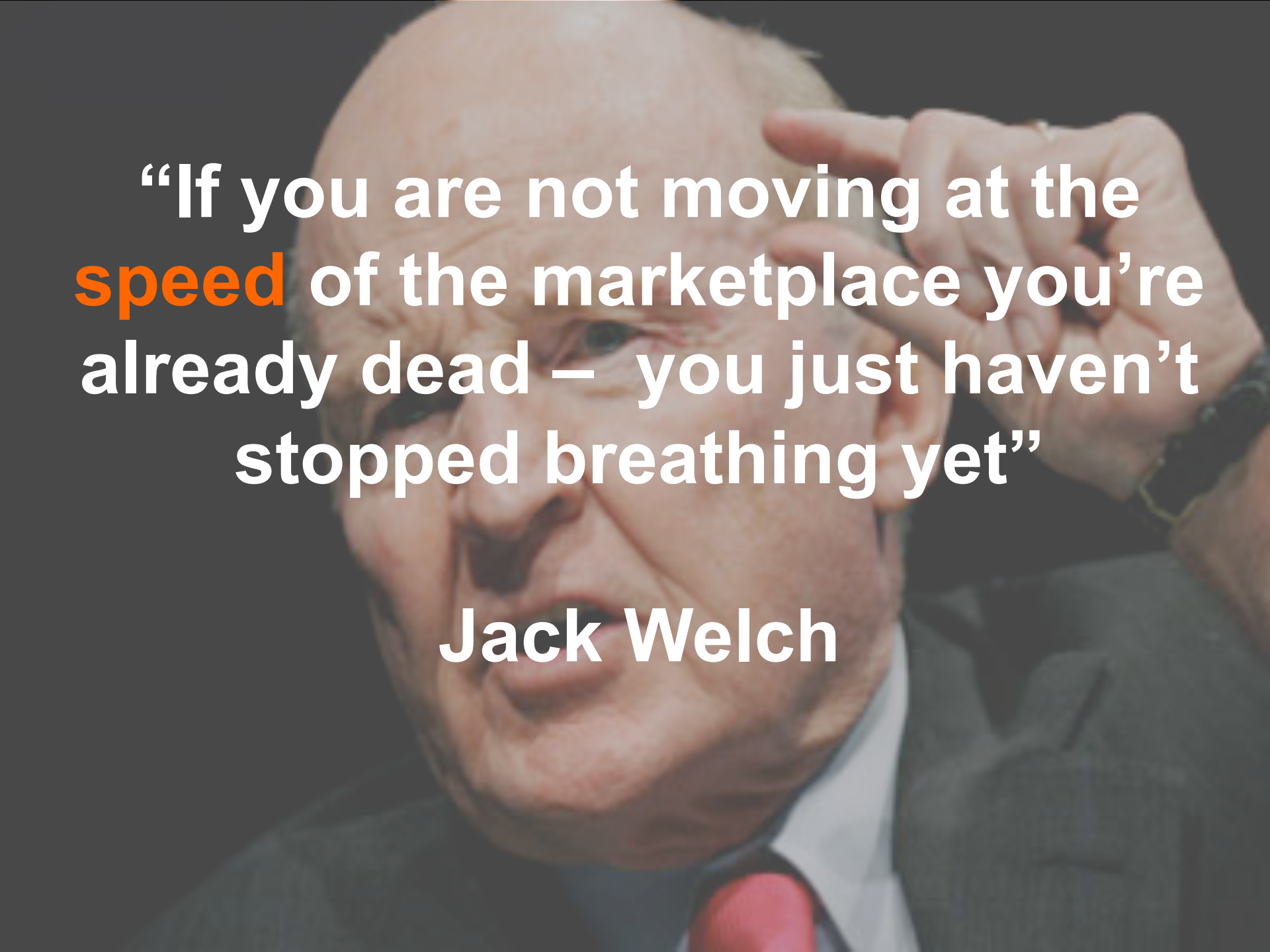
Speed, Innovation and Simplicity through Software Architecture

Jan Bosch
Professor of Software Engineering
Chalmers University of Technology
Gothenburg, Sweden.
www.janbosch.com

February 2012

Industrial Engineering and Management

Guest lecture


“If you are not moving at the speed of the marketplace you’re already dead – you just haven’t stopped breathing yet”
Jack Welch

On Education …

If you think education is expensive,
try ignorance

Derek Bok
(former president of Harvard University)

When a subject becomes totally obsolete
we make it a required course

Peter F. Drucker
(the man who invented management)

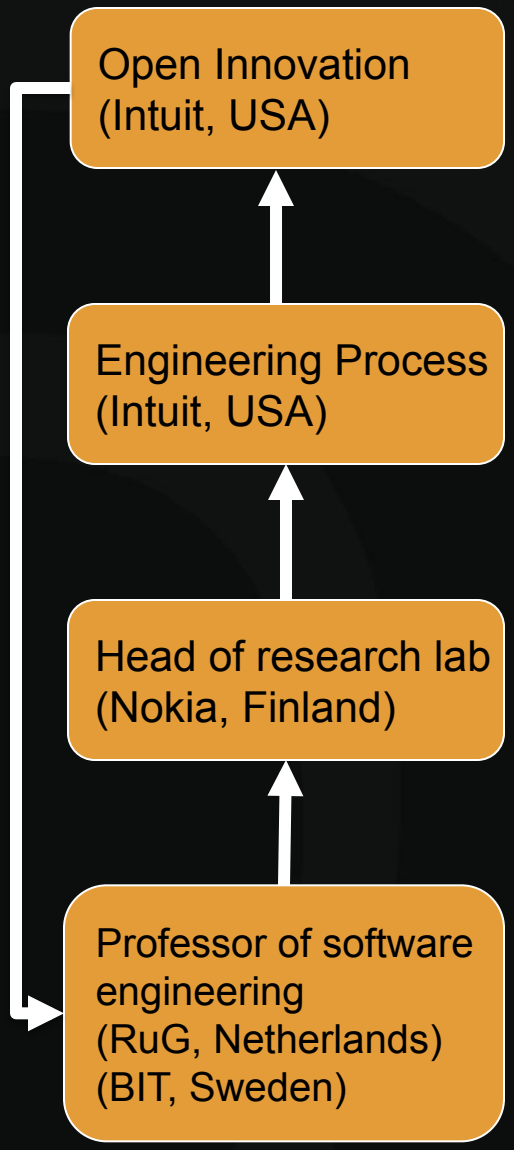
Three Key Take-Aways

- Increasing **SPEED** trumps ANY other improvement R&D can provide to the company – the goal is **continuous deployment** of new functionality
- Software engineering is at an inflection point – from “integration-oriented” to “composition-oriented” software engineering
- Software architecture is key to build **delightful** products in the context of **software ecosystems**

Overview

- Vem är jag? Wie ben ik? Who am I?
- Trends in Software: Need for Speed
- Innovation Experiment Systems
- Software Ecosystems
- Architecture & Scale
- Implications for ICT Professionals
- Conclusion

From Research to Industry



Innovation

Industrial development

Industrial research

Academia (+ consulting)

Software Center @ Chalmers

- Mission: Improve the software engineering capability of the Nordic Software-Intensive Industry with an order of magnitude
- Theme: Fast, continuous deployment of customer value
- Founding members

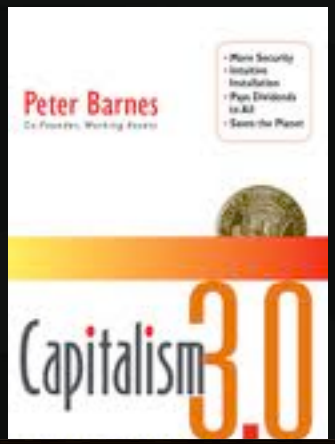
- Dual success metrics
  - Academic excellence
  - Tangible industrial impact

Overview

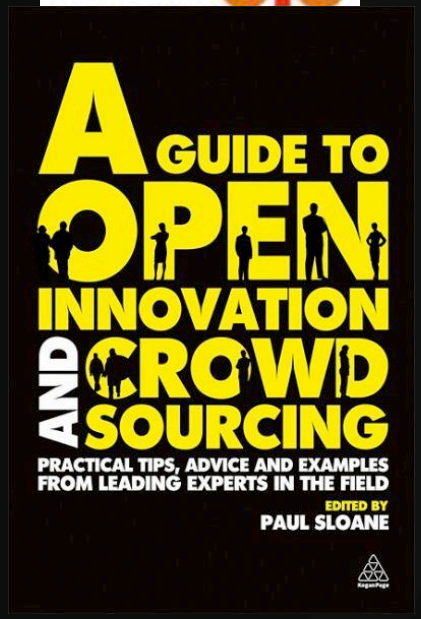
- Vem är jag? Wie ben ik? Who am I?
- Trends in Software: Need for Speed
- Innovation Experiment Systems
- Software Ecosystems
- Architecture & Scale
- Implications for ICT Professionals
- Conclusion

Trend: Products to Services

Trend: Capitalism 3.0



COINs
collaborative
innovation networks



Trend: Need for Speed

Value Creation Shifts

Emerging companies highlight importance
of user contribution and social connectedness

Founded	1984	1995	2004
1M users	~6 years	30 months	10 months
50M users	N/A	~80 months	~44 months

Need for Speed in R&D – An Example

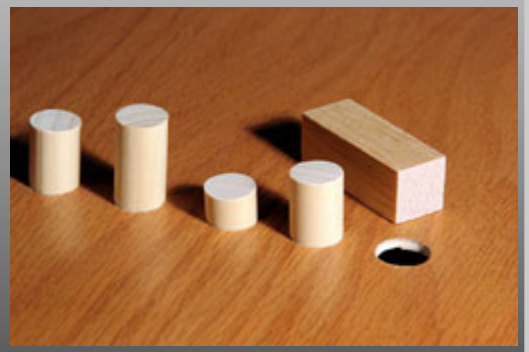
- Company X: R&D is **10%** of revenue, e.g. 100M$ for a 1B$ product
- New product development cycle: **12 months**

- Alternative 1: improve efficiency of development with 10%
  - **10 M$** reduction in development cost
- Alternative 2: reduce development cycle with 10%
  - **100M$** add to top line revenue (product starts to sell 1.2 months earlier)

No efficiency improvement will outperform cycle time reduction

Traditional Software Engineering

software product lines
global software development
software ecosystems



causing

unacceptable complexity and coordination cost

Web 2.0 Rules to SW Development (1/2)

Team size

- 3x3 = 3 persons x 3 months (Google)
- 2 pizza rule (Amazon)
- Principle: What is required is a team, where the roles are defined and each member has the right skill for that role, and following a lean, agile, method — all focused on the customer.

Release cycle

- Weeks, not months
- Continuous deployment
- Principle: short cycles are key for agility, speed and decoupling

Architecture

- 3 API rule
- Mash-ups and web services
- Principle: architecture provides simplicity, compositionality and is designed in parallel with software development

Focus on one thing: Minimize Dependencies

Web 2.0 Rules to SW Development (2/2)

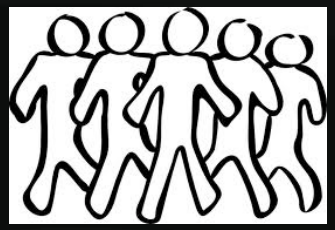
Requirements and Roadmapping

- Each team (3 persons) announces what they intend to release
- Some (QA) requirements are shared across the board, e.g. performance, latency, etc.
- Principle: the cost of overlapping teams is much lower than the cost of synchronized, planned roadmaps and plans

Process

- CMMi and other process maturity approaches address the symptoms, not the root cause
- Control is a very expensive illusion causing LOTS of inefficiency in the system
- Principle: Architecture, not process, should manage coordination and alignment

Need for Speed - Principles



Team
- 2 pizza's
- self-selected, directed and managed
- quantitative output metrics

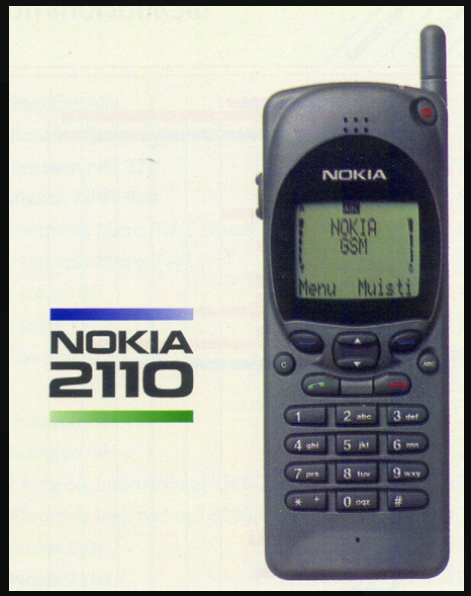
Architecture
- simplicity – 3 API rule
- backward compatibility – no versions!
- focus on compositionality

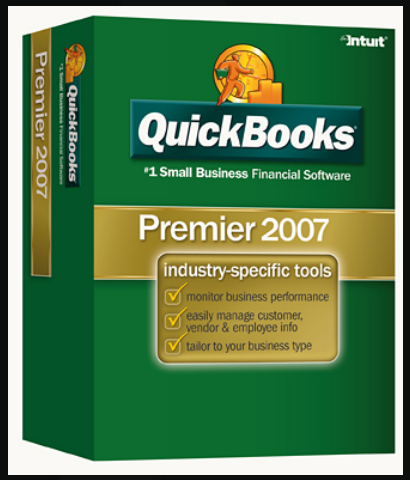
Release process
- continuous, independent deployment
- all the way to customers – installed base
- measure usage to feed back into development

Overview

- Vem är jag? Wie ben ik? Who am I?
- Trends in Software: Need for Speed
- Innovation Experiment Systems
- Software Ecosystems
- Architecture & Scale
- Implications for ICT Professionals
- Conclusion

What Do These Product Have in Common?





Example: Apple

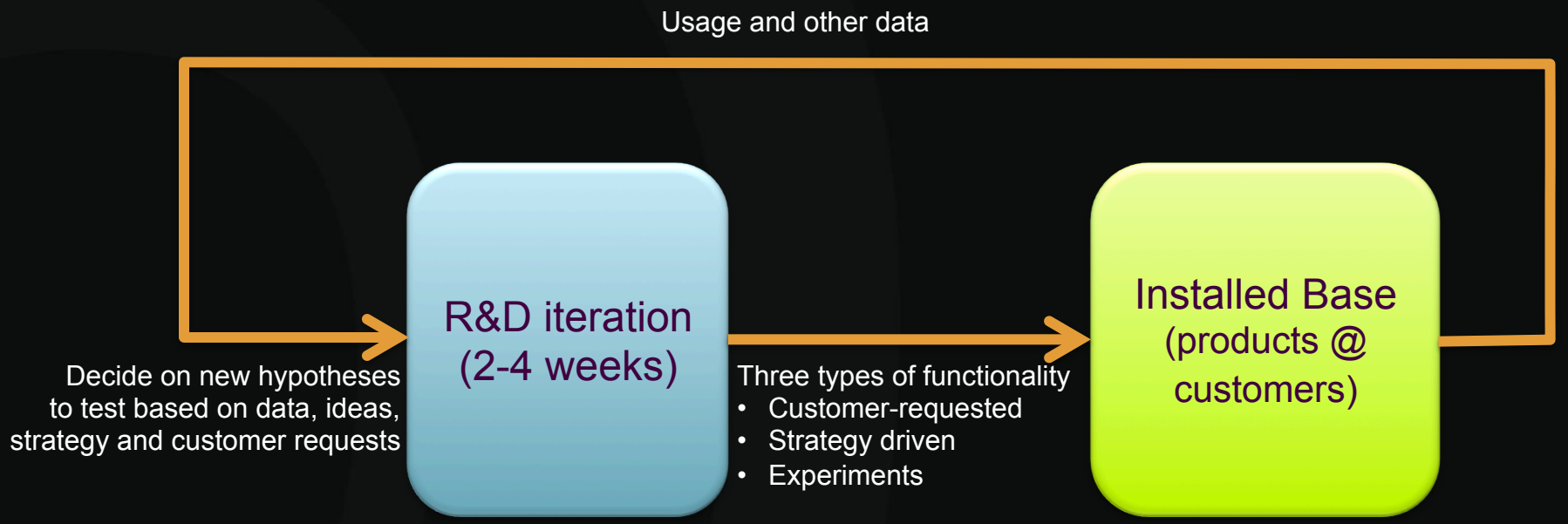
The Myth	The Reality
Inspired innovation	Create and winnow 10 pixel-perfect prototypes
Inspired design	Build a better backstory (intricate layers of business design behind the products
Brilliantly inspired marketing	Engineer the perfect customer experience to create customer experience and buzz

Reference: http://blogs.hbr.org/cs/2011/08/steve_jobs_and_the_myth_of_eur.html

R&D as an Experiment System

Learning: the company running the most experiments against the lowest cost per experiment wins

Goal: increase the number of experiments (with customers) with an order of magnitude to ultimately accelerate organic growth



Decisions should be based on DATA, not opinions

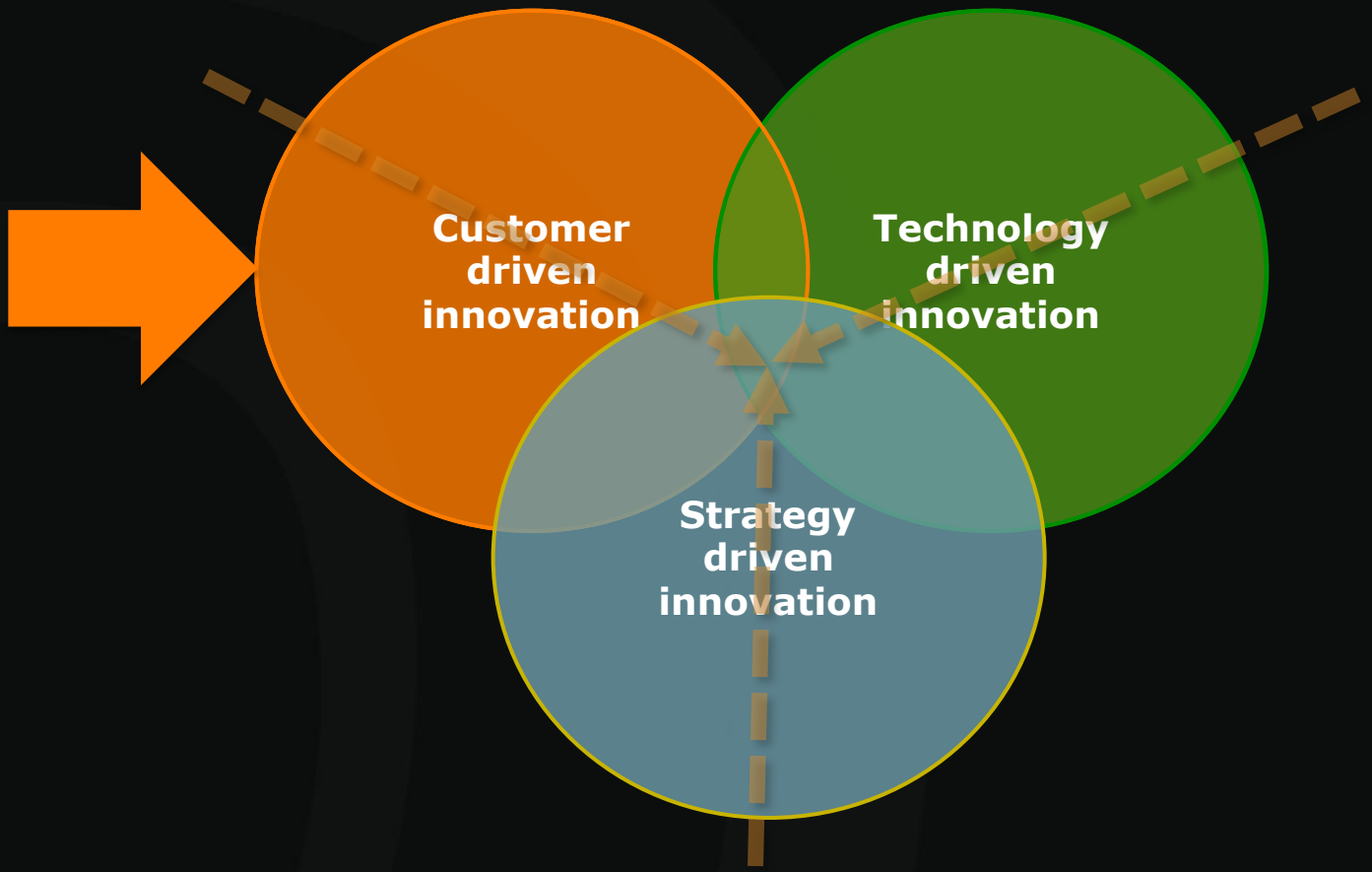
Stairway to Heaven

Scope of Experimentation

Stages and Techniques

Pre-Development	Development	Evolution
BASES testing	Independently deployed extensions	Random selection of versions
Advertising	Feature alpha	Instrumentation of usage metrics
Solution jams	Product alpha	Surveys
Mock-ups	Labs website	Ethnographic studies
	Product beta	

Innovation Approaches

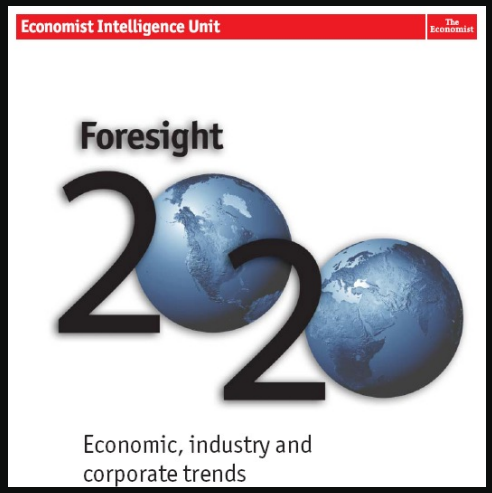


Overview

- Vem är jag? Wie ben ik? Who am I?
- Trends in Software: Need for Speed
- Innovation Experiment Systems
- Software Ecosystems
- Architecture & Scale
- Implications for ICT Professionals
- Conclusion

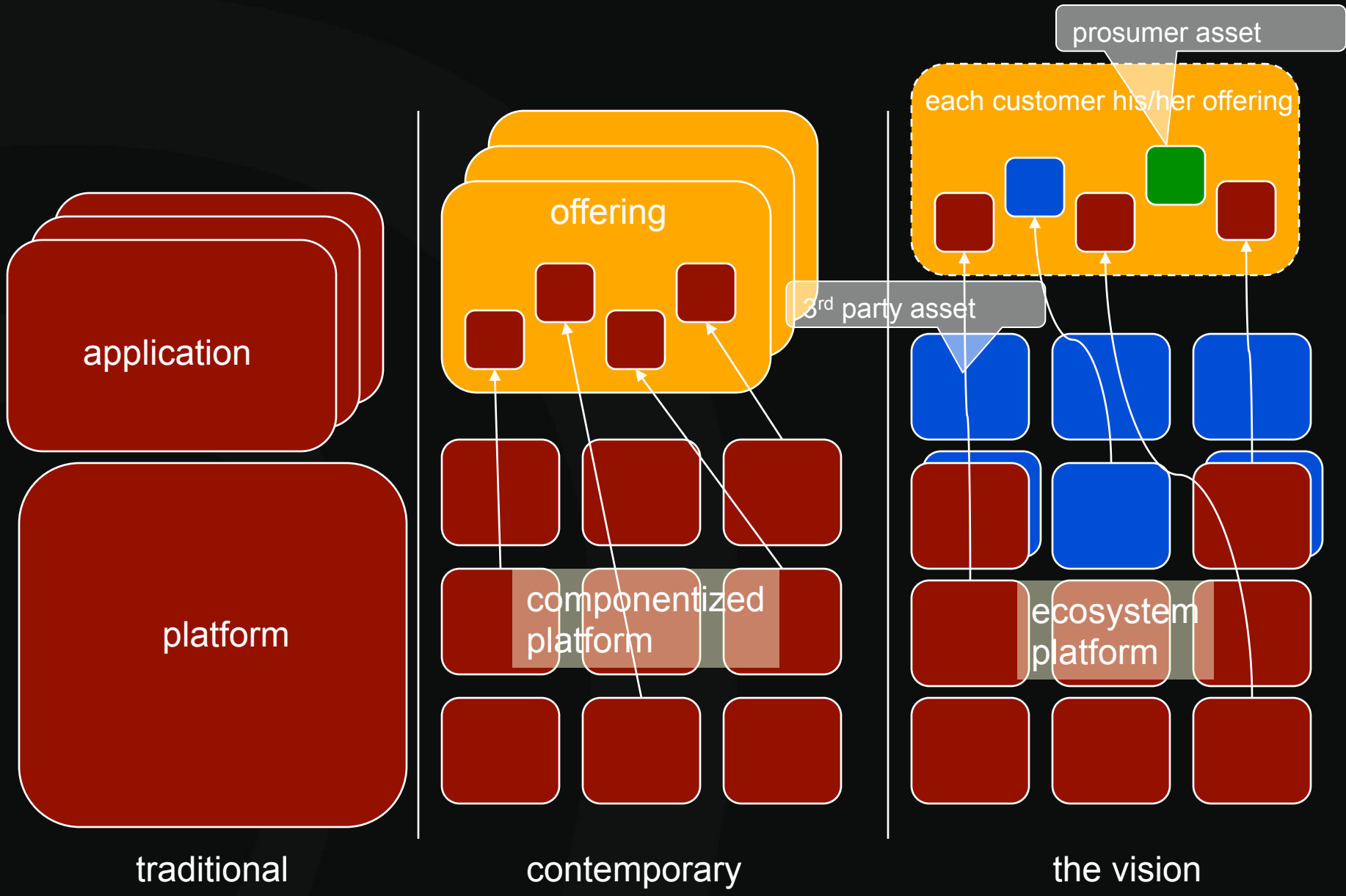
Towards Web 3.0

3 Atomisation. Globalisation and networking technologies will enable firms to use the world as their supply base for talent and materials. Processes, firms, customers and supply chains will fragment as companies expand overseas, as work flows to where it is best done and as information digitises. As a result, effective collaboration will become more important. The boundaries between different functions, organisations and even industries will blur. Data formats and technologies will standardise.



"My prediction would be that Web 3.0 will ultimately been seen as applications which are pieced together. There are a number of characteristics: the applications are relatively small, the data is in the cloud, the applications can run on any device, PC or mobile phone, the applications are very fast and they're very customizable. Furthermore, the applications are distributed virally: literally by social networks, by email. You won't go to the store and purchase them... That's a very different application model than we've ever seen in computing." —Eric Schmidt

Evolution of Development Approaches



Software Ecosystem?

- Here's a try: A software ecosystem consists of a software platform, a set of internal and external developers and a community of domain experts in service to a community of users that compose relevant solution elements to satisfy their needs.
- Some more detail:
  - **Software platform**: A hierarchical set of shared software components providing functionality that is required and common for the developers constructing solutions on top of the platform.
  - **Evolution**: Over time, the functionality in the ecosystem commoditizes and flows from unique solutions to the platform.
  - **Developers**: Although internal and external developers use the platform differently, the platform often allows developers to build on top of each other's results.
  - **Composition**: Users are able to compose their own solutions by selecting various elements into a configuration that suits their needs optimally.

Why Software Ecosystems?

- Increase value of the core offering to existing users
- Increase attractiveness for new users
- Increase “stickiness” of the application platform, i.e. it is harder to change the application platform
- Accelerate innovation through open innovation in the ecosystem
- Collaborate with partners in the ecosystems to share cost of innovation
- Platformize functionality developed by partners in the ecosystem (once success has been proven)
- Decrease TCO for commoditizing functionality by sharing the maintenance with ecosystem partners

Taxonomy of Software Ecosystems

category / platform	desktop	web	mobile
end-user programming	MS Excel, Mathematica, VHDL	Yahoo! Pipes, Microsoft PopFly, Google's mashup editor	*none so far*
application	MS Office	SalesForce, eBay, Amazon, Ning	*none so far*
operating system	MS Windows, Linux, Apple OS X	Google AppEngine, Yahoo developer, Coghead, Bungee Labs	Nokia S60, Palm, Android, iPhone

Overview

- Vem är jag? Wie ben ik? Who am I?
- Trends in Software: Need for Speed
- Innovation Experiment Systems
- Software Ecosystems
- Architecture & Scale
- Implications for ICT Professionals
- Conclusion

Role of Software Architecture

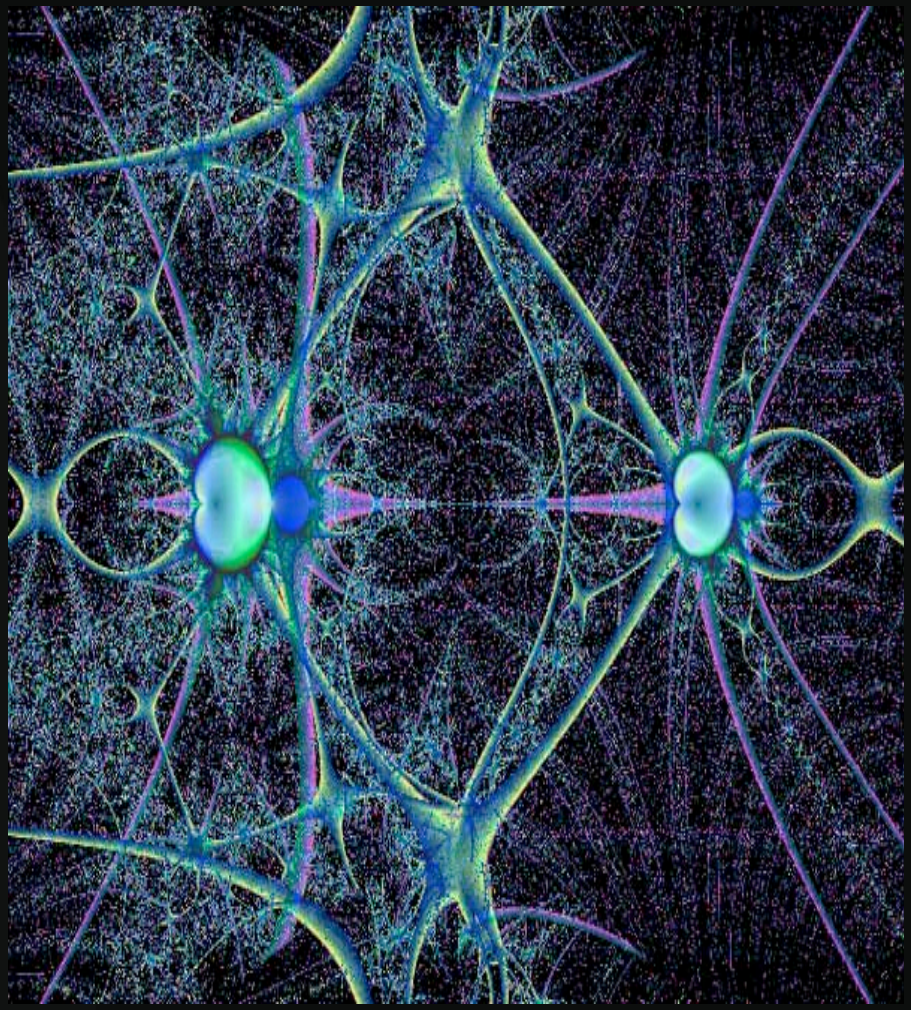
- Simplify, Simplify, Simplify
- Decoupling
  - Components
  - Teams
  - Organizations
- Lean and agile at scale
- End to end quality requirements
- Fight design erosion

Simplify, Simplify, Simplify

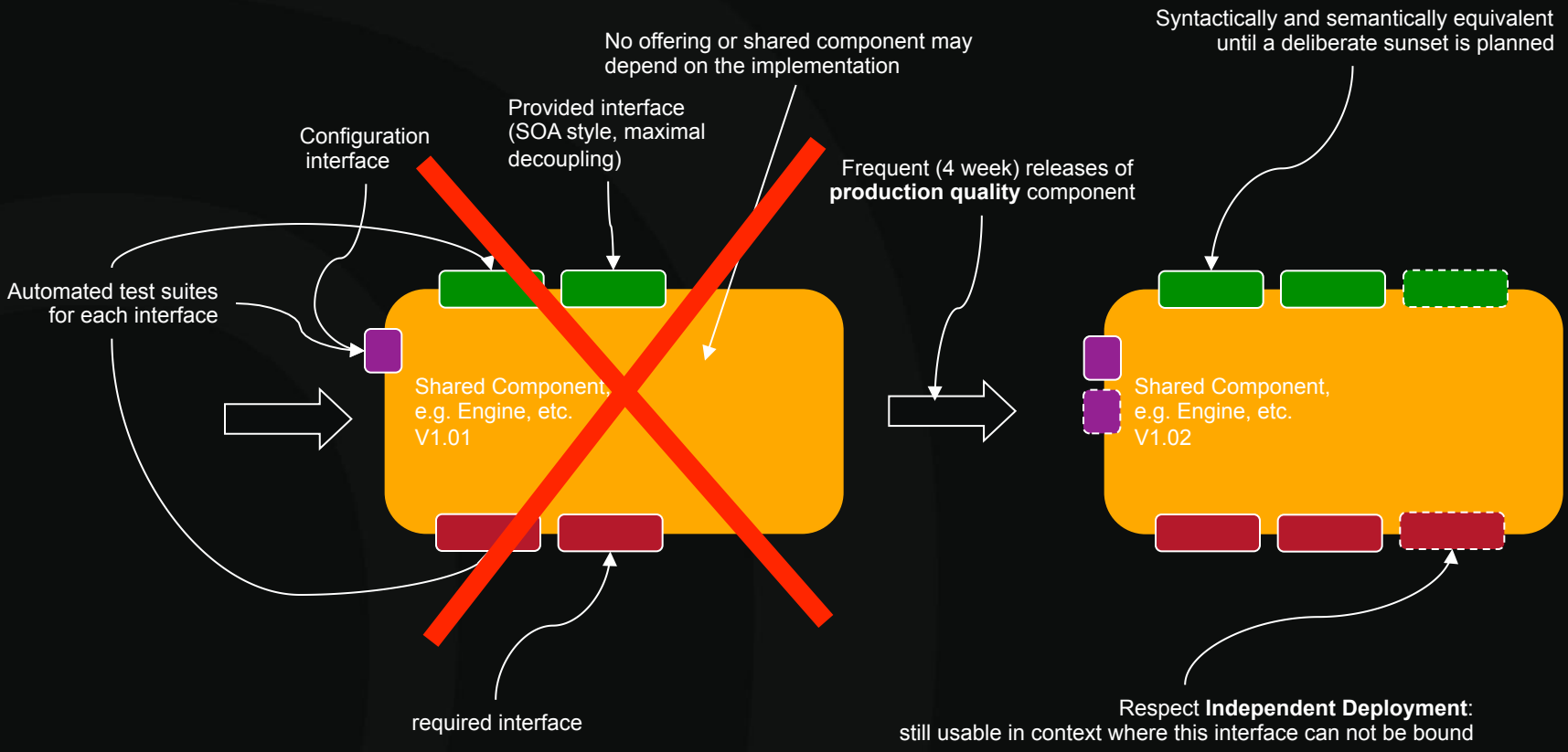
- Each architectural design decision adds design rules and constraints that cause complexity
- Insist on simplicity (3 APIs rule)
- How
  - Push down in the stack
  - Hide
  - Automate
  - Redesign

Decouple Teams and Organizations

- Interconnected teams and organizations asymptotically reduce productivity to ***zero***
- Decouple teams and make sure no continuous interaction is needed
- How
  - Continuous deployment
  - No versions
  - No concurrent development

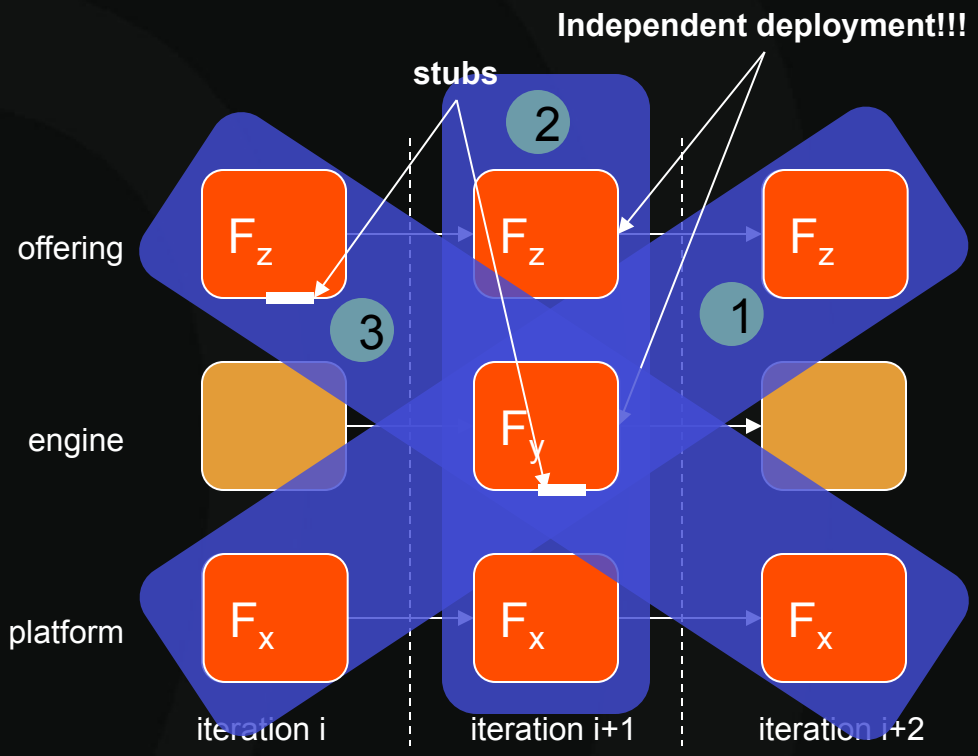


Decoupling: No Versions!



Decouple Components and Teams

1. Sequential feature development (90%)
2. Concurrent development, independent deployment enforced (8%)
3. Exploratory development (2%)



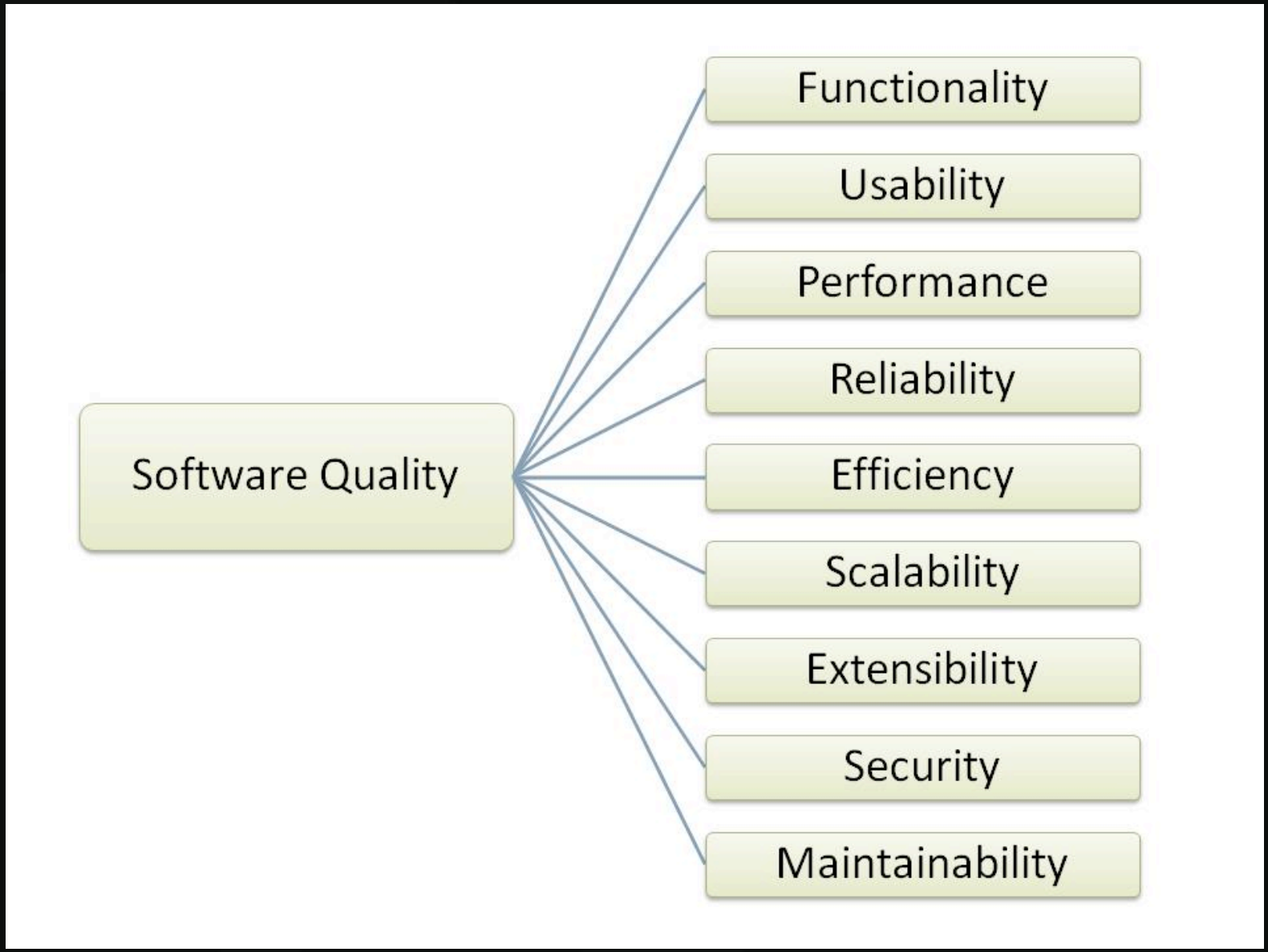
Strive For Continuous Deployment

- Software engineer checks in code => system compiles, links, tests and deploys the new code
- The automated QA infrastructure, NOT the engineer, is responsible for making sure the system does not go down
- If that's too much, aim for Independent Deployment
- If that's too much, aim for Release Trains

Lean and Agile at Scale

- Achieving lean & agile in large, legacy systems with large R&D organization considered an oxymoron
- Google, Amazon and Intuit are examples that it can be done
- How
  - Small teams
  - Short cycles
  - Direct customer connection
  - Clear success metrics

End to End Quality Requirements

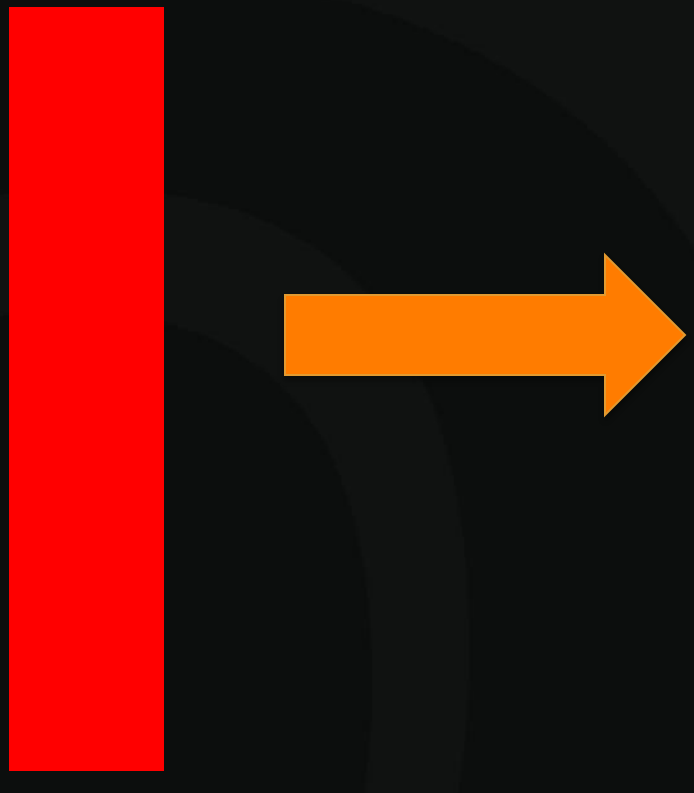


Evolve Architecture; Fight Erosion

Overview

- Vem är jag? Wie ben ik? Who am I?
- Trends in Software: Need for Speed
- Innovation Experiment Systems
- Software Ecosystems
- Architecture & Scale
- Implications for ICT Professionals
- Conclusion

Implications for ICT Engineers



Multi-disciplinary
Learn continuously
Self-starting
Love customers
Understand business
Drive to metrics
Build networks
Move fast
Entrepreneurial

Shadow Beliefs

- Humans are better than machines in identifying known and new reliability issues – we are building business critical systems, after all!

My experience: data always trumps opinion; test and validation systems pre-deployment and extensive data-collection post-deployment inform decision making

- Software-intensive systems (large, complex, tough requirements) are different and approaches from other domains do not apply

My experience: system failure is devastating in several industries and avoided in Internet systems while adopting agile and continuous deployment

- We should avoid or delay adoption of new, more efficient engineering approaches

My experience: getting first to market with new functionality that closely aligns to customer needs is a significant competitive advantage that drives growth and results in market leadership

Guidelines

1. Modularize the system in critical and (less or) not critical parts
2. Adopt agile and continuous deployment approaches for the not (so) critical part first
3. Deeply engage with customers to develop optimal solutions to their real pain points
4. Invest in testing infrastructure that continuously and thoroughly tests systems with no human involvement
5. Instrument systems for pre- and post-deployment data collection, concerning at least reliability and usage metrics
6. Architect your systems for maximum decoupling and modularization between different components to allow for independent deployment
7. Replace commoditizing functionality with Open Source or COTS components; focus R&D on truly differentiating parts

Overview

- Vem är jag? Wie ben ik? Who am I?
- Trends in Software: Need for Speed
- Innovation Experiment Systems
- Software Ecosystems
- Architecture & Scale
- Implications for ICT Professionals
- Conclusion

Speed

Increasing **SPEED** trumps ANY other improvement R&D can provide to the company – the goal is **continuous deployment** of new functionality

- If you're not a front-line engineer, there is only ONE measure that justifies your existence: how have you helped teams move faster?
- Don't optimize efficiency, optimize speed

Inflection Point

- Software engineering is at an **inflection point** – from “integration-oriented” to “composition-oriented” software engineering

- Design for automated compositionality, not manual integration
- Minimize dependencies
- Focus on small teams of engineers, give them direction and get out of their way

Software Architecture 2.0

- **Software architecture** is is central in allowing for independent, continuous deployment to customers

- Architecture happens (in parallel)
- A la Thoreau: Simplify, Simplify, Simplify
- Decouple components, decouple teams and decouple organizations
- Lean and agile at scale

Not My Job?!

Strong LEADERSHIP needed from YOU

Thank you!

Q&A