IN5320 - Development in Platform Ecosystems

Lecture 13: *Summary*

19th of November 2018

Department of Informatics, University of Oslo
Magnus Li - magl@ifi.uio.no
Olav Poppe - olavpo@ifi.uio.no

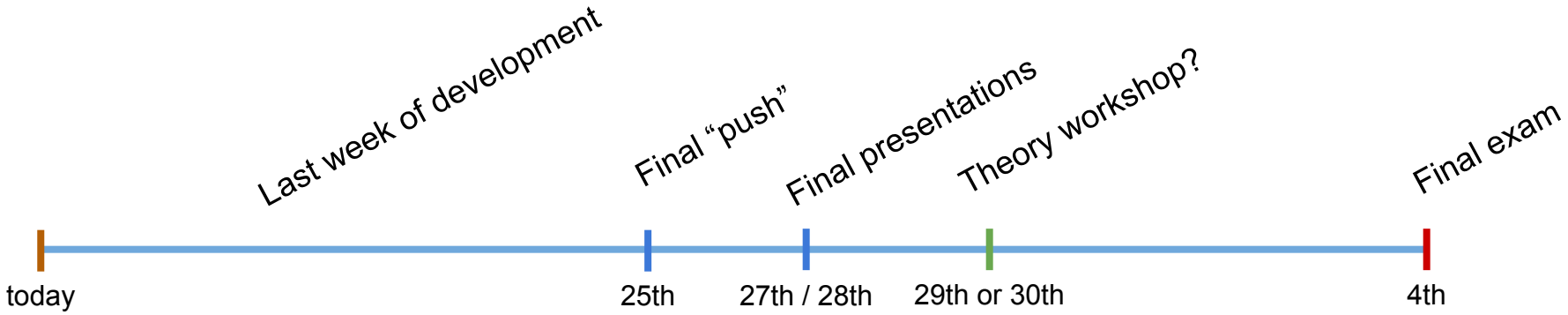
Course evaluation

Remember to fill out the course evaluation!

- The course is “new”
- What was good?
- What can be improved, and how?

https://goo.gl/vNTisZ

Following weeks



This week

Last week of development!

Important to prioritize fundamental functionality.

If you have time: focus on additionals.

Remember to send Andrei an email with the URL to your git + give access to the group teachers (info here)

Deployment to DHIS2 is mandatory (info here)

Final “push”

- Final “push” on the 25th of November.
- Give access to the group teachers (see post on piazza)

- Include a readme-file in the repository where you describe:
  - Your app’s functionality
  - How this is implemented (just a brief explanation)
  - Any missing functionality/implementations, and things that do not work optimally.

Final group presentation

- Presentation on 27th or 28th.
- All group members have to attend the presentation, but everyone do not have to speak.
- 10 minutes to present (make sure you do not use more time than this)
- 5 minutes for questions and discussion

Main objective:

- Demonstrate your final product (the app)
- Reflect on decisions regarding design, functionality, implementation, and the overall process.
- See the evaluation criteria here.

Final group presentation

The presentation should include:

- Brief intro with your requirements for the selected case
- Demo of the app, covering all functionality
- Reflections on your implementation
    - (Structure of code, use of APIs, using functionalities and data model of dhis2 versus hard-coding in app)
- Some words about the development process. (e.g., how you have coordinated the implementation work, negotiated and decided on different aspects, etc.)
- What you could have done differently / things that do not work optimally or that you would have done if you had more time.

Theory workshop?

Theory workshop after project presentations?

Working with and discussing the theoretical assignments.

Thursday 29th or Friday 30th

https://goo.gl/GGGNie

Topic summary

Topic summary

- Information systems, complexity and architectures
- Platform ecosystems fundamental concepts
- Design & innovation
- Programming / implementation / licensing

About the theoretical exam

Learning outcome

After you have completed this course you:

- will have an understanding of the socio-technical complexity related to large-scale information systems that span several use-cases and contexts.
- can reflect on how platform architectures and ecosystems might enable development of applications that are sensitive to local requirements within a larger system.
- will have insights on web development, software platforms, RESTful web services and APIs.
- can prototype and develop apps for a software platform.
- can develop web-based software with JavaScript and HTML, using modern, open-source frameworks.
- will have experience with developing software in a team.

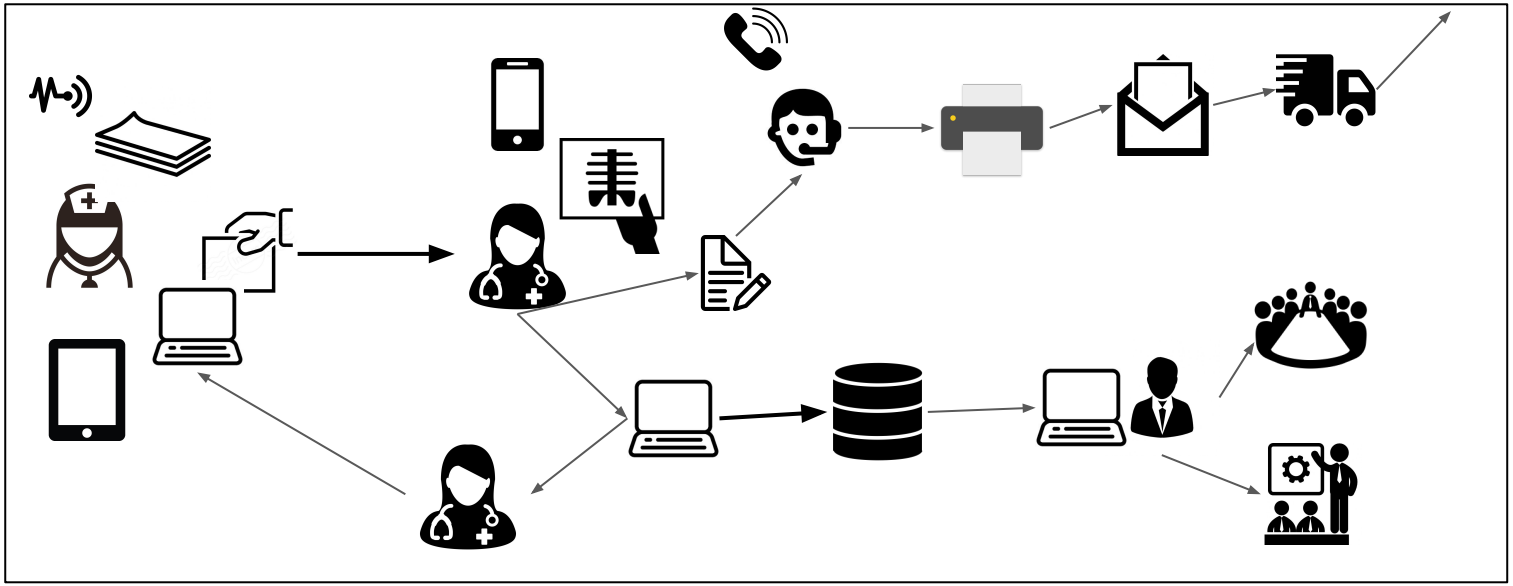
Information Systems

An information system is not the information technology alone, but the system that emerges from the ***mutually transformational interactions*** *between the* ***information technology*** *and the* ***organization***.

(Allen S. Lee, 2004)

Information Systems

Information system = technology <--> organization(s)



Complexity

Complicated systems

Linear behavior

Total is equal to the sum of its parts

Complex systems

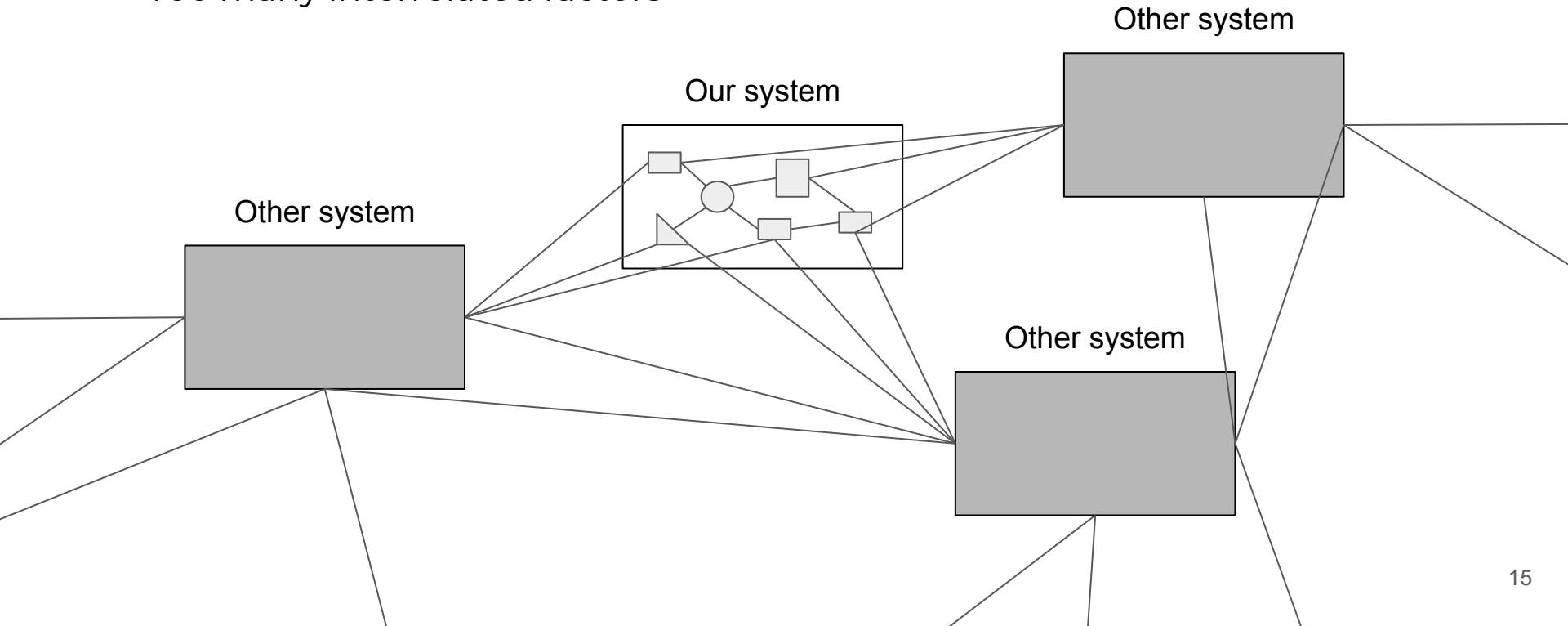
Non-linear behavior (change in input is not proportional to new output)

System can not be fully understood by investigating its parts.

"Complexity stems from the number and type of relationships ***between the systems' components*** *and* ***between the system and its environment***" (Hanseth & Lyytinen, 2010)

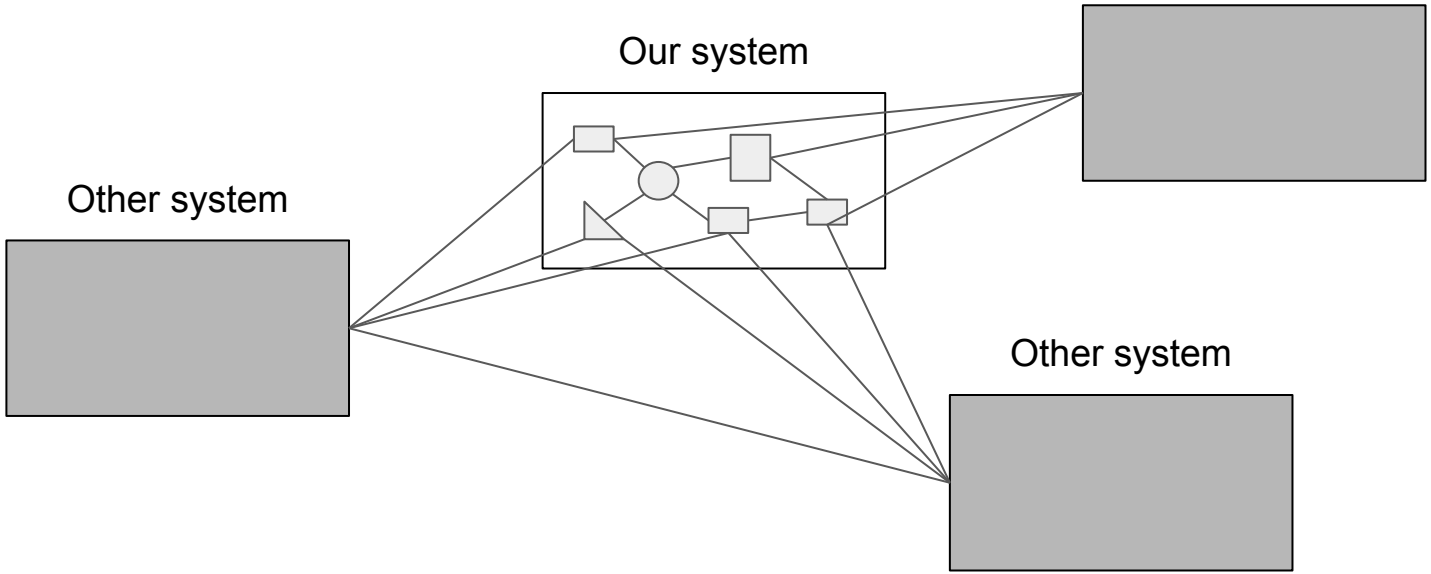
Why is it complex?

- Too many unknowns
- Too many interrelated factors



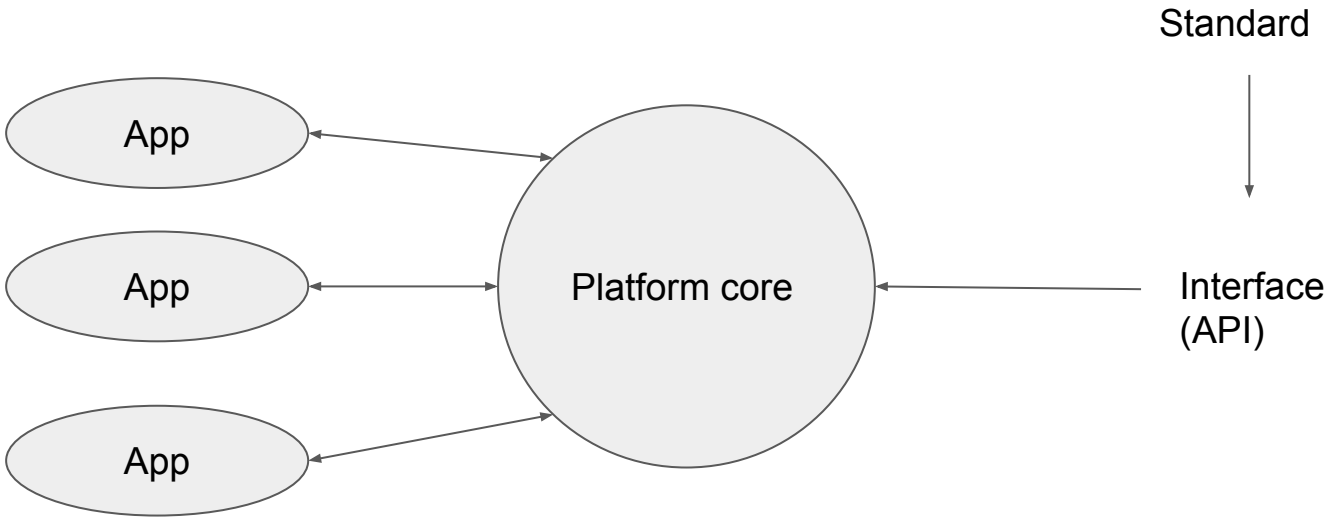
Standards

- Standards are fundamental in Information Systems



Standards

- Standards are fundamental in Information Systems



Standards

- Standardization is not a purely technical process.

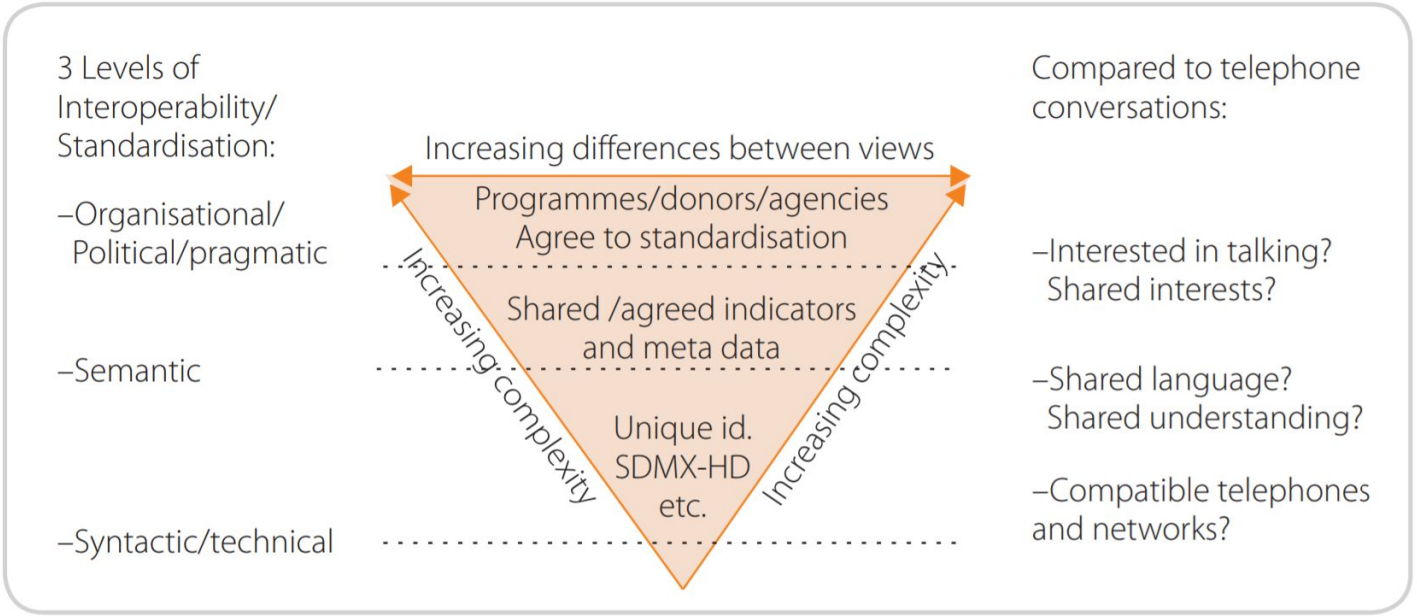


Figure 3.4 Three levels of standardisation of the increasing differences and complexities

Braa & Sahay 2012

Software architectures

- Blueprints of previous, current, or future arrangement of components and their relations in an information system (purely technical, or socio-technical)
- Architectures can act as constraining or enabling of desired aspects such as
  - Innovation
  - Design
  - Implementation
  - Maintenance
  - Scalability
  - Customization
  - Reusability
- A key aim is to reduce complexity

Software architectures

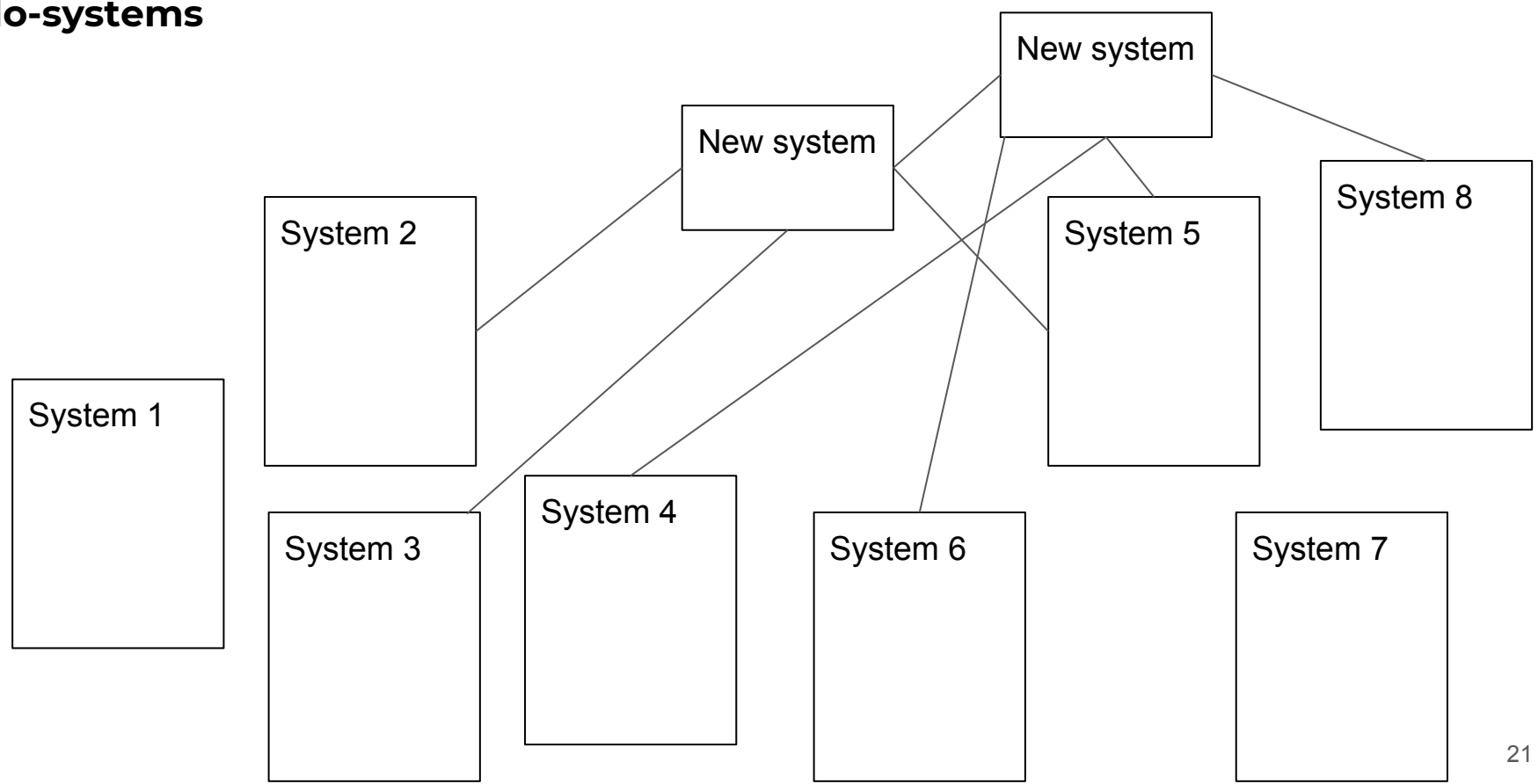
- System architecture tend to mirror the organization.
- What could be a challenge with this?

Conveys law: *"organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."*

Mealyís Law: *«The eventual structure of the system reflects the structure of the organization that builds the system»*

Software architectures

Silo-systems



Software architectures

A good architecture must exhibit four simple properties that it shares with the architecture of modern cities: simplicity, resilience, maintainability, and evolvability.

Tiwana 2012 p77

Software architectures

- **Platform Architectures** has become a common way of structuring systems to promote certain desired aspects.

- A response to contextual factors (“Drivers” - Tiwana 2013)
- Also of conscious thought and design?

Drivers towards platform ecosystems (Tiwana 2013)

1. **Deepening specialization**

   Technologies get increasingly complex and specialized → Require deeper expertise
2. **Packetization**

   Process of digitizing new phenomenons such as activities or processes.
3. **Software embedding**

   Business activities are put into software.
4. **Internet of things**

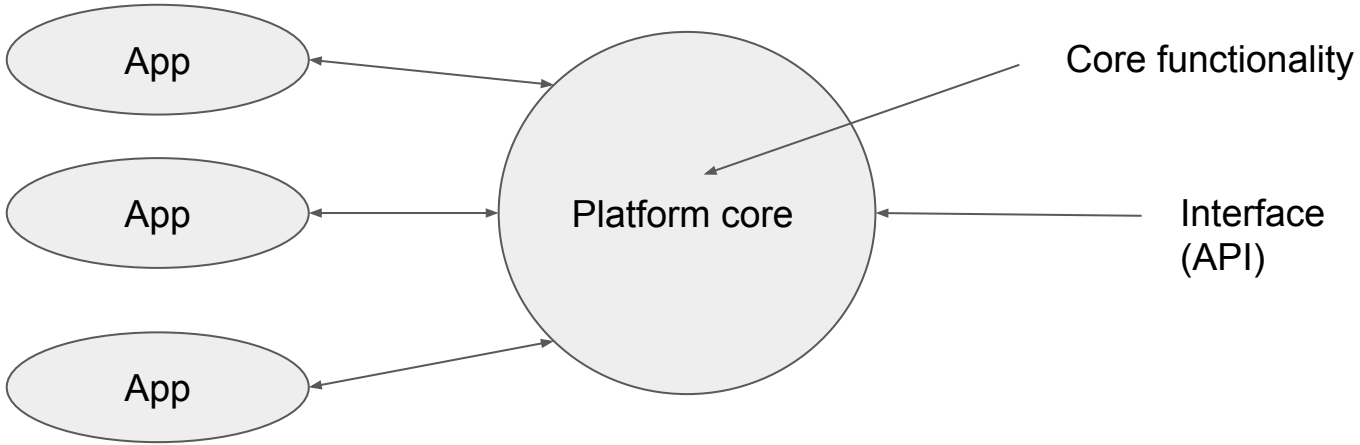
   It is increasingly easy to connect everyday objects to the web.
5. **Ubiquity**

   Fast and cheap networks are available “everywhere”

What is a platform?

"A software platform is a software-based product or service that serves as a foundation on which outside parties can build complementary products or services" - Tiwana 2013 p5

- Provides core functionality which is extendable
- Entails interfaces that allows third parties to develop *apps* that extend the functionality of the platform



Core characteristics and concepts (Tiwana 2013)

- Multisided
- Network effects (same-sided, cross-sided)
- Multihoming
- Tipping point
- Lock-in
- Competitive durability
- Envelopment

Boundary resources

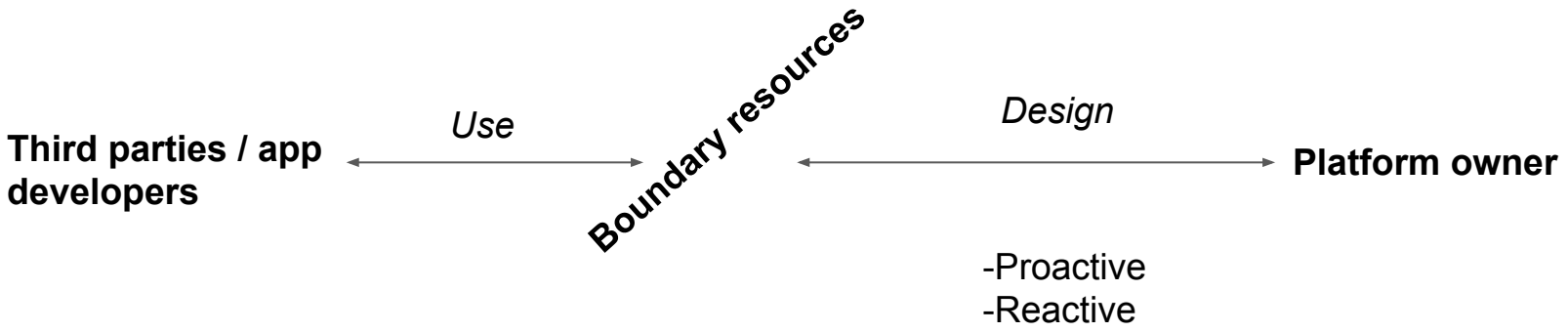
“To successfully build platform ecosystems, the focus of the platform owner must shift from developing applications to providing resources that support third-party developers in their development work” - Ghazawneh & Henfridsson 2013 p 174

→ Boundary resources: resources enabling third party development through tools and regulations

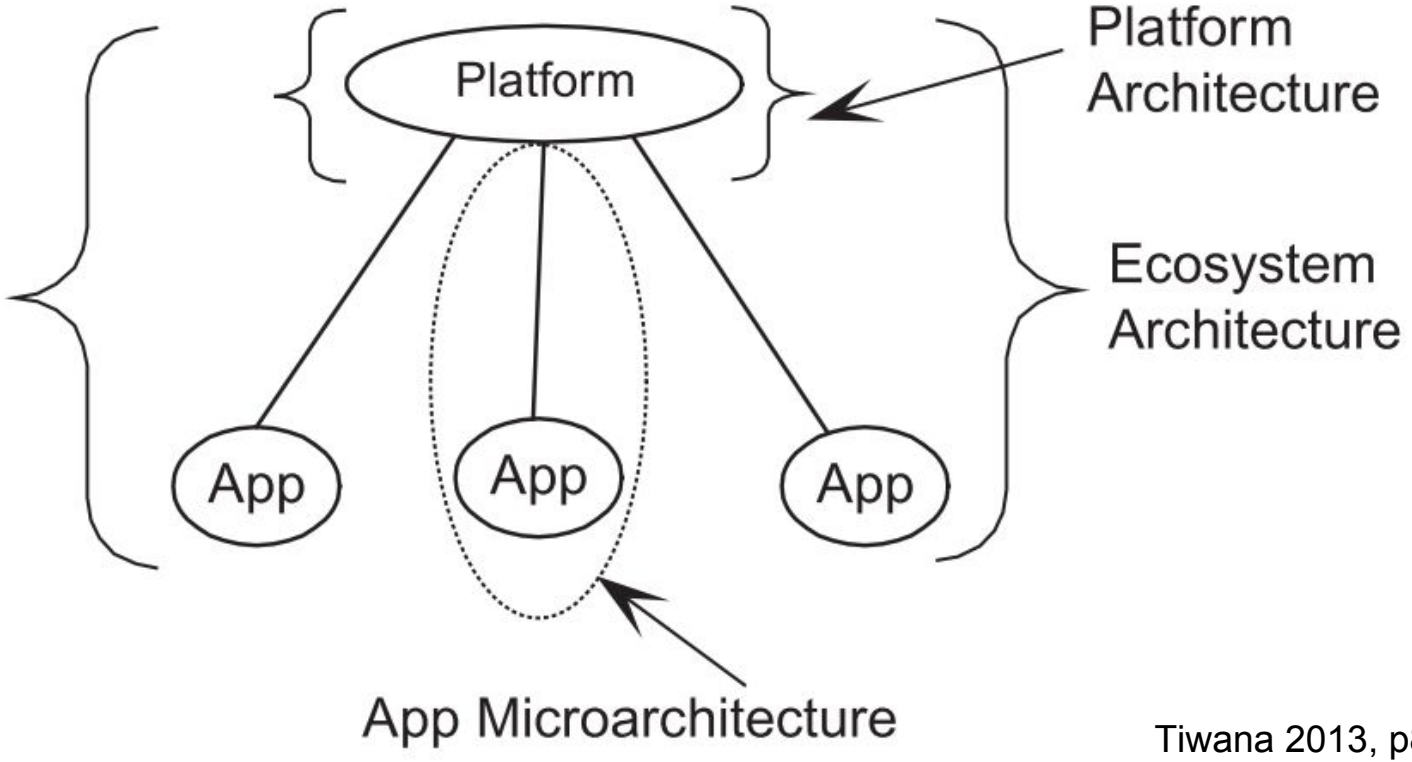
Boundary resources

- To enable innovation, design and development of new functionality to the platform.
- To control the platform and its evolution in some desired direction.

Therefore: boundary resources has to be designed with the balance between these two in mind.



Three levels of architecture



Tiwana 2013, p85

Design and innovation

Challenges in large-scale systems development

Two problems in large-scale / generic software development:

1. “Generic” usability
2. Working with local users in development

\+ Local relevance, utility and innovation!

Challenges in large-scale systems development

Variety

Alt 1: Everyone uses their own systems

Alt 2: Everyone use the same system

Alt 3: Combination

Challenges in large-scale systems development

Variety

Alt 1: Everyone uses their own systems

- Complexity
- Problem of integration
- Inefficient?
- Halts innovation

Challenges in large-scale systems development

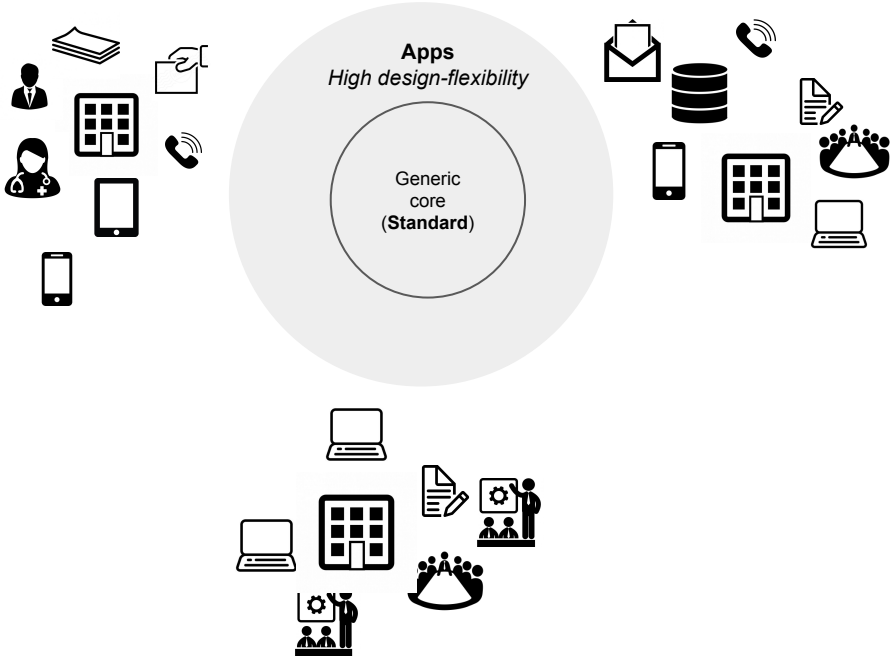
Variety

Alt 2: Everyone use the same system

- Usability
- Local relevance and utility
- Innovation

Challenges in large-scale systems development

Variety



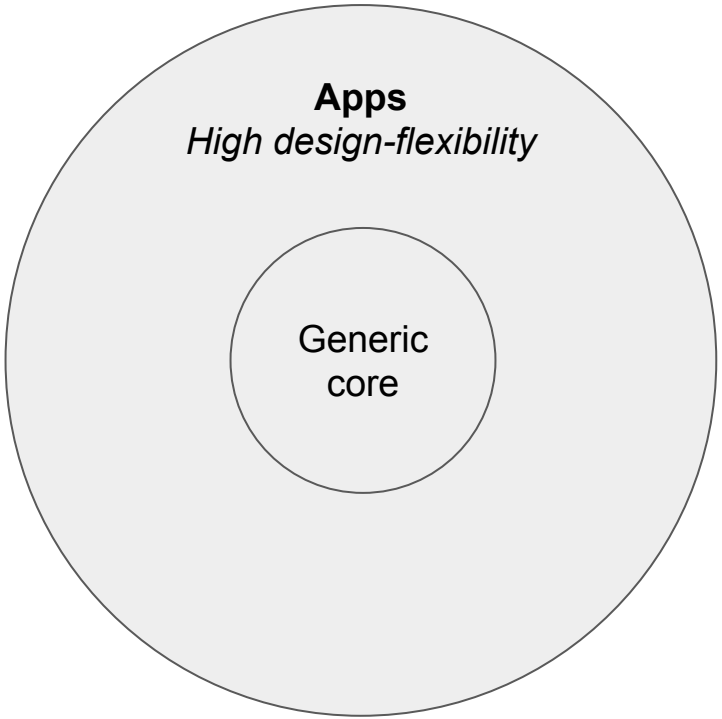
Alt 3: Combination

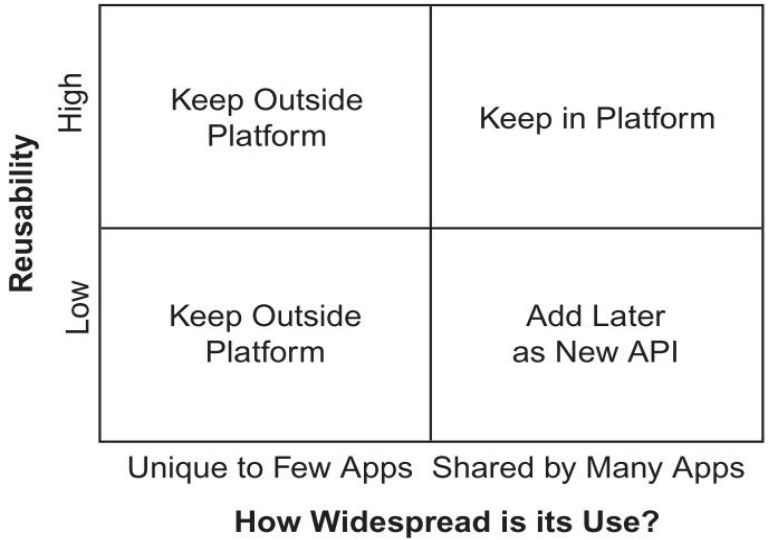
- Platform architecture can support this

- Enable innovation
- Usability and local relevance and utility
- Scalability
- Adaptability

Platform design flexibility

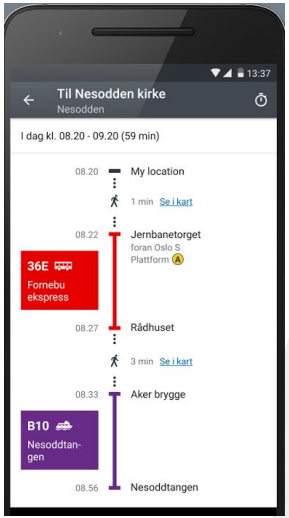
Opening up the software architecture for the development of third-party apps could be one way of balancing variety and standards

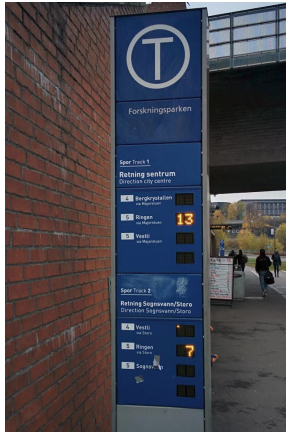




Tiwana 2013

Variety and standards: Ruter

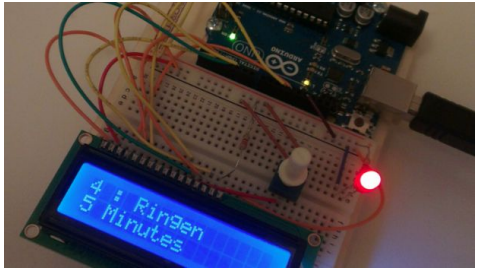


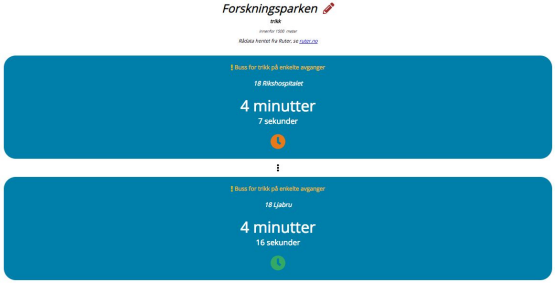


Apps

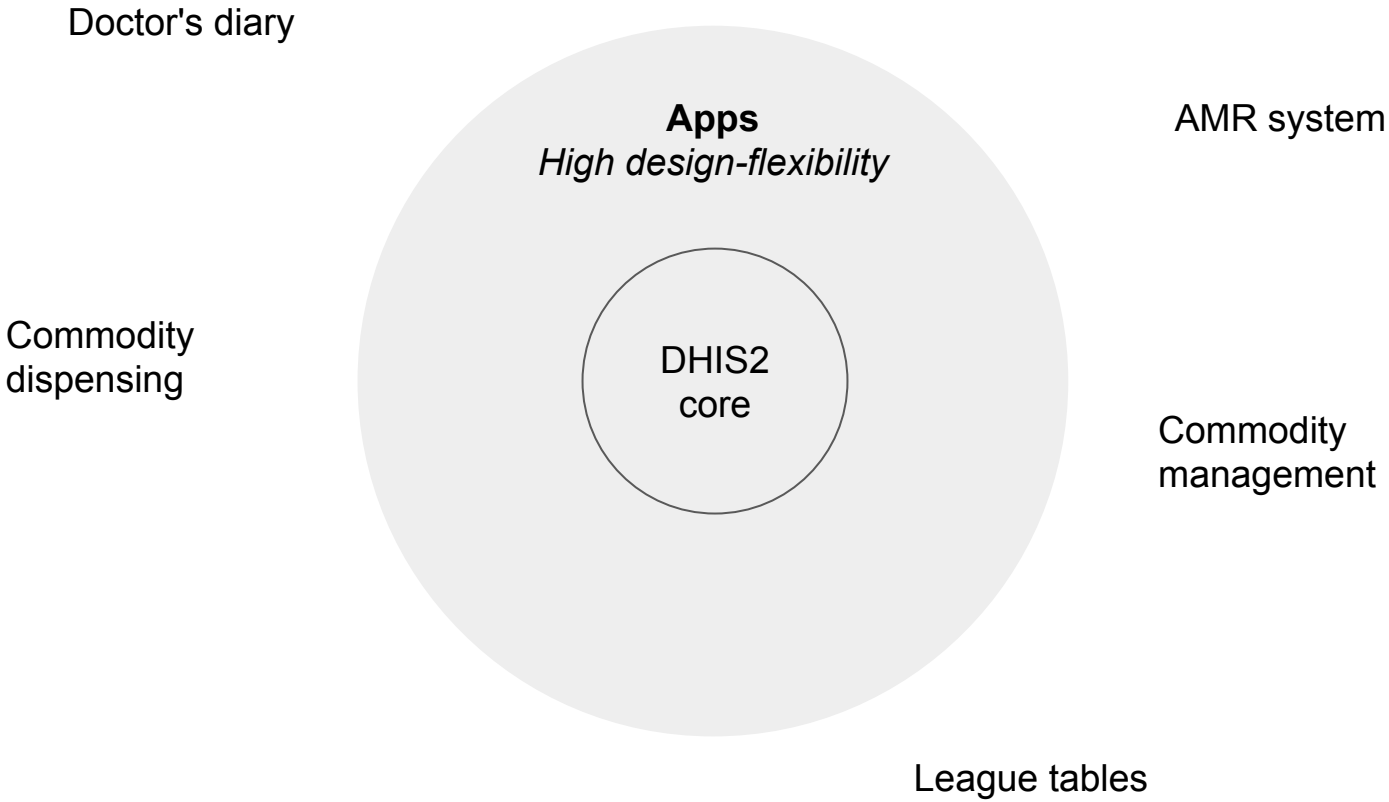
High design-flexibility

Ruter platform core





Variety and standards: DHIS2



Variety and standards: Case - AMR in India

- Old system was too generic (off-the-shelf software)
- Built their own tailored to their domain and needs
- Happy with this → works well with the users as it is designed specifically for the use-case.

- Problem with scalability
- Communication with other systems
- Takes lots of time to create data presentation tools (graphs, maps, etc.)

- Move to DHIS2?

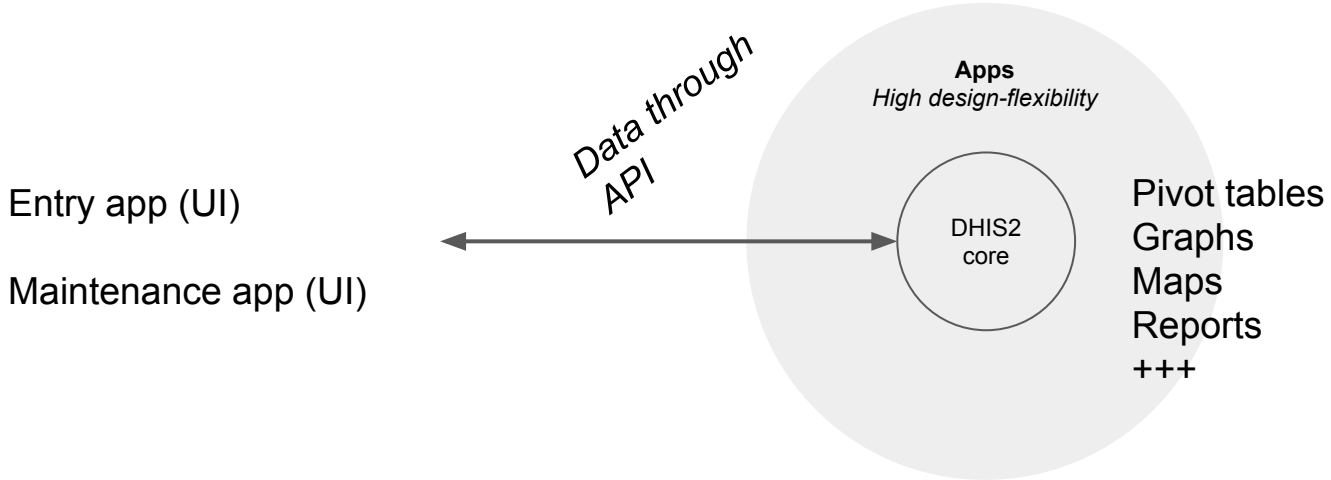
Variety and standards: Case - AMR in India

The specific

- *Domain language*
- *Work practices*
- *Legacy designs*
- *User's mental models*

The generic

- *Data storage*
- *Scalability*
- *Integration*
- *Data presentation*



Programming / implementation / licensing

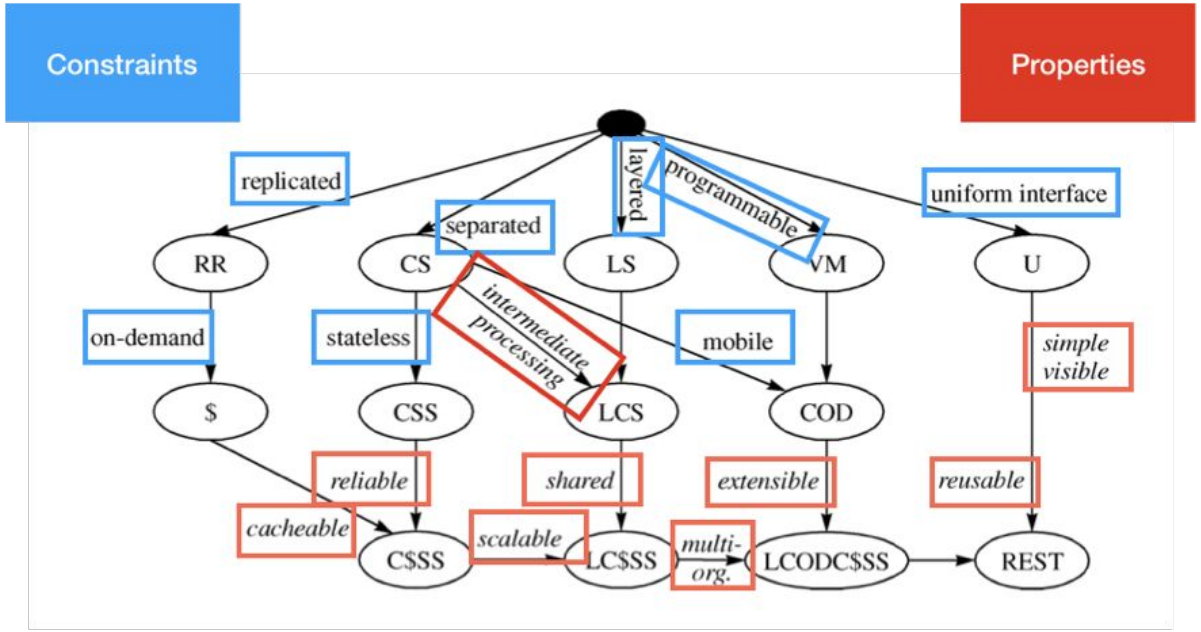
Programming / implementation / licensing

- Relevant languages and frameworks in front-end web development
- AJAX and APIs
- REST architectural style
- Licensing: free and open source software

Front-end programming

REST architectural style

- REST is an architectural style - a set of architectural constraints, that can (a bit simplified) be thought of as "rules" for what is allowed within an architecture



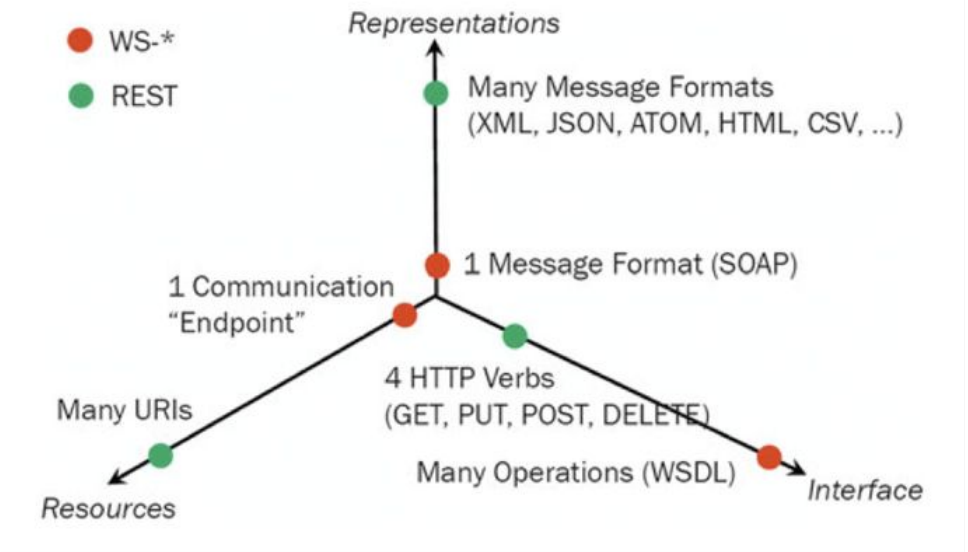
Source: Fielding and Taylor (2002)

REST architectural style

- “Uniform interface” constraint for interacting with resources the most immediately relevant for creating/interacting with a RESTful API/web service:
  - **Addressability** - all resources are identified by one identifier mechanism
  - **Universal semantic** - a small set of standard methods support all interactions and apply to all resources
  - **Resource representations** - resources are manipulated through their representations
  - **Self-describing messages** - interactions happen through request and response message that contain both data and metadata
  - **Hypermedia** as engine of application state (HATEOAS) - resources include links to related resources, enabling decentralised discovery. Application state is kept on client, resource state on server.

REST architectural style

- A maturity model for REST defines how “RESTful” a web service is, according to how fully it makes use of the affordances of the HTTP protocol
- “Traditional” web services (XML-RPC) use HTTP primarily for transport



Source: Pautasso 2014

Free and Open Source Software

- Differentiate between FS and OSS, which has different origins and purposes
  - FS focus on the freedoms of users to do what they want with the software
  - OSS focus on improving software development - faster, cheaper, better quality through sharing, re-useability, peer reviews etc

NAV sine applikasjoner er finansiert av skattebetalerne, og skal derfor i utgangspunktet være åpne for innsyn slik at skattebetalerne kan ha tillit til at det vi leverer er juridisk og teknologisk tilfredsstillende. Som stor samfunnsaktør er det viktig at vi bidrar til Open source-miljøet, spesielt når vi nyter godt av deres arbeid selv. NAV IT

- FOSS enables a range of different business models
- Authors of software is given copyright to their work, and software licenses govern how others can use the work
- FOSS licenses can be permissive or restrictive – related to FS vs OSS. A “free software” viewpoint implies licenses that are restrictive and viral so that software remains FOSS.

FOSS and platform ecosystems

Openness	Boundary resources	Actor who shares	Shared resources	Type of sharing	Platform owner's rational
Access	API, app store	Complementors (e.g. app developers)	Complement, e.g. apps	Shared for distribution	Generate network effects; extract value from complementaries
Resource	Open-source license	Platform owner	Platform core, e.g. AOSP	Shared IPR	Strategic forfeiture of IPR while recovering costs elsewhere

- Platform openness can encourage growth of the platform ecosystem
- Platform forking is the creation of a new competing platform from an existing platform's resource base - strategic rather than just technical
- Boundary resources, including (open source) licenses, are important both to enable platform growth and for control/governance

Final exam

Final exam

- 4th of December
- Individual
- Four hours
- No help resources
- Digital (Silurveien)

Final exam

Questions are based on what we have talked about in the lectures and the mandatory readings.

- 8 - 10 short answer questions (2 - 4 % each)
  - Answer the question brief with a few sentences.
- 3 - 5 long answer questions (8 - 12 % each)
  - Discussion. Try to use all relevant concepts from the course readings.

Theoretical weekly assignments provide an indicator.

- Some of these may be used in the final exam.

Course evaluation

Remember to fill out the course evaluation!

- The course is “new”
- What was good?
- What can be improved, and how?

https://goo.gl/vNTisZ