Information systems and complexity

Large-scale complex IT systems

I. Sommerville et al., (2012)

Large scale II - holder ikke med rent teknisk perspektiv. Det er reductionism, da ser man ikke helheten i infrastrukturen.
Inherent complexity; great number of parts, great number of possible connections, and feedback loops.
Coalition of systems.

Argues that traditional software engineering is not sufficient to under Large-Scale complex IT systems because it has the philosophical notion of reductionism -> does not allow them to see the complexity of the system.
A system cannot be understood by itself, and has to be seen in the context of the systems dynamic operating environment.

- Coalition of systems
- Complex systems
    - Inherit complexity -> iboende kompleksitet
    - Epistemic complexity -> mangel på kunnskap om systemet, gjør at man ikke forstår og det virker komplekst
- Reductionism
    - Software engineering has focused reducing and managing epistemic complexity

Key insights

- Coalition of systems, in which the system elements are managed and owned independently, pose challenging new problems for systems engineering.
- When the fundamental basis of engineering - reductionism - breaks down, incremental improvements to current engineering techniques are unable to address the challenges of developing, integrating and deploying of large-scale complex IT systems.
- Developing complex systems requires a socio-technical perspective involving human, organizational and political factors, as well as technical factors.

Keywords: reductionism, coalition, inherent complexity, epistemic complexity, large-scale complex IT systems

Balancing the Local and the Global in Infrastructural Information Systems

K. Rolland, E. Monteiro (2002)

- Balance between sensitiveness to local contexts and a need to standardize across contexts
- Can be connected to platforms

- The platform core can be the global infrastructure which one make local adjustments on (for example apps)

Keywords: information infrastructure, standardization of global work practices, improvisation, globalisation, local infrastructure, global infrastructure

Additional readings
Orlikowski & Lacono (2001) - Research Commentary: Desperately Seeking the "IT" in IT Research—A Call to Theorizing the IT Artifact
Lee (2004) - Thinking about social theory and philosophy for information systems

Platform ecosystems fundamental concepts

The architecture of platforms: A unified view

C. Baldwin & C. Woodard (2008)

Definition of platform:
"*a set of stable components that supports a variety and evolvability in a system by constraining the linkages among the other components*"
"*A platform system consists of a core, its compliments, and the interfaces between them"*
-> A set of core components (the platform), a variety of apps

Purpose of article: relation between platforms and the systems in which they are embedded.
Better understand firms and industries where platforms play an important role.
Platform architecture
Useful when the underlying system is complex, but needs to adapt to changing tastes and technologies.
Modularization that partitions the systems into a set of components, complementary set of components that can vary.

Reuse and share core components and functionality

The fundamental feature of platform architecture is that certain components remain fixed over the life of the platform, while others are allowed to vary in cross-selection or change over time.

Interfaces - e.g. the interfaces between components creates specific thin crossing points in the network of relationships between the elements in the system

Keywords: platforms, architecture, multi-sidedness, interface, modules, evolvability

Platform Ecosystems, chapter 1 The Rise of Platform Ecosystems

A. Tiwana (2013)

En software platform defineres i teksten på følgende måte: "*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.*". Et plattform-økosystem består av to hovedkomponenter - selve plattformen og tilhørende apper.

Real platforms also have :Multi-sidedness. And interactions between different sides through the core of the platform.

Videre diskuterer Tiwana hva som er en plattform, og hva som ikke er en plattform. Det er ofte slik at de fleste suksessfulle plattformer begynner som et stand-alone produkt. Tiwana argumenterer også for at en plattform ikke kan være "one-sided", men må ha flere sider - den skal fasilitere interaksjonen mellom (minst) to forskjellige grupper (for eksempel utviklere og sluttbrukere).

Istedenfor at produktene konkurrerer mot hverandre, konkurrerer plattformens økosystemer mot hverandre.

Tiwana snakker om fem drivere som har bidratt til dette skiftet til plattformbaserte virksomheter; deepening specialization, packetization, ubiquity, internet of things og software embedding.

Table 1.1 Core Elements of a Platform Ecosystem

Element	Definition	Example
Platform	The extensible codebase of a software-based system that provides core functionality shared by apps that interoperate with it, and the interfaces through which they interoperate	iOS, Android Dropbox, Twitter AWS Firefox, Chrome
App	An add-on software subsystem or service that connects to the platform to add functionality to it. Also referred to as a module, extension, plug-in, or add-on	Apps Apps Apps Extensions
Ecosystem	The collection of the platform and the apps specific to it	
Interfaces	Specifications that describe how the platform and apps interact and exchange information	APIs Protocols
Architecture	A conceptual blueprint that describes how the ecosystem is partitioned into a relatively stable platform and a complementary set of apps that are encouraged to vary, and the design rules binding on both	–

Keywords: platform, platform ecosystem, drivers, deepening specialization, packetization, ubiquity, internet of things, software embedding

Platform Ecosystems, chapter 2 Core Concepts and Principles

A. Tiwana (2013)

Platform lifecycle - A multifaceted characterization of whether a technology solution, a platform, an app, or the entire ecosystem, is in its pre- or post-dominant design stage; its current stage along the S-curve; and the proportion of the prospective user base that has already adopted it

- *Dominant design* - A technology solution that implicitly or explicitly becomes the gold standard among competing designs that defines the design attributes that are widely accepted as meeting the user's needs
- *S-curve* - A technology's life-cycle that describes its progression from introduction, ascent, maturity and decline phases
- *Leap-fogging* - Embracing a disruptive technology solution and using it as the foundation for the firm's market offering in lieu of an incumbent solution in the decline of the S-curve
- *Diffusion curve* - A description of whether a technology solution is in the stage of having attracted the geeks, early majority, early adopters, late majority or laggards to its user base
- ***Multisidedness*** - The need to attract at least two different groups who can interact more efficiently through a platform than without it
- ***Network effects*** - A property of a technology solution where every additional user makes it more valuable to every other user on the same side (same-sided network effects) or on the other side (cross-sided network effects)
    - Two properties: direction (positive/negative) and sidedness
    - *Negative network effects* - When every additional user makes it less valuable for every other user
        - More users - more complex - possibly more chaos and
    - *Same-sided* - When adding an additional participant to one side of the platform changes its value to all other participants on the same side
        - Positive: More facebook users adds value to other facebook users
        - Negative: adding one more driver to an already busy highway, decreases the appeal to other users
    - *Cross-sided* - When adding an additional participant to one side of the platform changes its value to users on an other side of the platform
        - Example: More people buying ipad, more developers wants to develop apps
- ***Multihoming*** - When a participant on either side participates in more than one platform ecosystem
    - Example: owning a android phone and an iphone
- ***Tipping*** - The point at which a critical mass of adopters makes positive network effects take off
- *Lock-in* - The way in which a platform can make it more desirable for existing adopters to not jump ship to a rival
- *Competitive durability* - The degree to which the adopters of a technology solutions continue to regularly use it long after its initial adoption
- ***Envelopment*** - When a platform swallows the market of another platform in an adjacent market by adding its functionality to its existing bundle
- ***Architecture*** - A conceptual blueprint that describes components of a technology solution, what they do,and how they interact

- ***Governance*** - Broadly, who decides what in a platform's ecosystem. This encompasses partitioning of decision-making authority between platform owners and app developers, control mechanisms, and pricing and pie-sharing structures

Guiding principles

The Red Queen Effect	Refererer til det økende presset om å tilpasse seg fortere kun for å overleve i markedet. Drevet av økende evolusjonær pace fra rivaliserende tech-løsninger. "Escalation in system thinking".
The Chicken-or-egg problem	En P kan ikke tiltrekke seg utviklere uten å ha stor brukerbase, og en stor brukerbase vil ikke kunne oppstå når det ikke finnes et bredt spekter av apper.
The penguin problem	Oppstår når en ny P introduseres, som har et stort potensiale for network effects, men som ingen adopter fordi de ikke vet om andre gjør det også. "Bandwagon effect". Akutt når den forespeilede brukergruppen er den installerte basen i en rivaliserende P.
Emergence	Selvforsterkende order som oppstår på bakgrunn av handlingene til appDevs og P-owners som forfølger egne interesser basert på egen kompetanse, men som kontinuerlig tilpasser seg feedback basert på hva andre i ecosystemet gjør.
The seesaw problem	Apper må sømløst integreres i en P, men det som leder til P-integrasjon kan også virke påtrengende på autonomien til appDev på måter som motvirker emergent innovasjon rundt P. P-owners needs to manage a balance between devs autonomy and ecosystem-wide integration.
The Humpty Dumpty problem	Refererer til å separere apper fra dets P for å kunne oppgradere, men dette kan gjøre det vanskelig å sette sammen igjen. Dette kan ha domino-effekter, da en endring i en del av økoosystemet kan påvirke andre deler. Solution: SW-arch.
The mirroring principle	Den organisatoriske strukturen av en P's ecosystem skal speile dets arkitektur. Løst koblet org = løst koblet arch.
Coevolution	Å simultant tilpasse arch og governance av en P/app for å vedlikeholde alignment mellom dem. Sammen fungerer de som en motor for ecosystem-evolusjon.
The Goldilocks rule	Ideen om at hvis man blir gitt tre graderte valg, så vil mennesker alltid helle mot det midterste. I stedet for å tilby en gratis og en betalt versjon av en app, vil en appDev kunne generere høyere summer ved å tilby tre versjoner.

Key insights

- Strategies that are appropriate for managing platforms and apps vary with where they are in their evolutionary life-cycle
- Software platforms have intrinsically different properties compared to other types of platforms
- Architectures of technology solutions provide the blueprint for mass coordination
- Governance can amplify or diminish the advantages of good architecture
- Evolutionary pace of a platform is relative to its rivals

Keywords: platforms, dominant design, S-curves, diffusion curves, multisidedness, network effects, multihoming, architecture, governance

Platform Ecosystems, chapter 5 Platform Architecture

A. Tiwana (2013)

Platform architecture.
Complexity. Platform ecosystems are usually seen as complex systems.
Structural and behavioral complexity.
Structural: interconnections between its parts are difficult to describe.
Behavioral: aggregate behavior is difficult to predict or control.

Architecture - tool to tackle structural complexity and governance.
Partitioning. Subsystems autonomous from others.
System integration. Coordination of development activities among app devs and the platform owner.

Apps - internal and external microarchitectures.

Modularization, have pros and cons for both platform owner and app devs. Mainly cons: performance and cost. Mainly pro: distributed innovation, variety in apps, greater app evolvability.

Platform and apps can have strong coupling within and weak coupling between.
Visible info and hidden info in both platform as well as in app.
Two levels of architecture:
Platform Architecture Ecosystem architecture (App microarchitecture)

Apps micro-architecture:
Presentation logic (interface between app and end-user).
Application logic (functionality)
Data Access logic (process to interact with data)
Data storage

- Complexity constrain innovation
    - Architecture can reduce structural complexity, but not behavioral complexity

- Architecture is a platform's DNA that imprints evolvability
- Architecture precedes organization
- Architecture partitiones and reintergrates a complex system
- Apps have internal and external microarchitectures
- Platform architectures have four desirable properties - simple, resilient, maintainable and evolvable
    - Simple: architecture should be easy to describe, main components, how they are partitioned and interact.
    - Resilient: keep dependencies to a minimum. Defect in one app should not affect others
    - Maintainable: allow changes in components without breaking others. Stable interface.
    - Evolvable: support new functionality/innovation. Stable interface.
- Modularization endows these desirable properties to architectures
    - Decoupling
- Decoupling in architectures follow two simple rules
    - 1) Partitioning the system into low-variety, high-reusable functions that go into the platform core and the high-variety, low-reusable functions to remain outside
    - 2) Specify assumptions that apps can make about the platform and vice versa
- A platform's interface follows three criteria
    - Precise, frozen and versatile
- A platform's interface is like traffic lights
    - Coordinate so everyone follows the same rules
    - Connected to platform governance

Keywords:platforms, complexity, architecture, app microarchitecture

Balancing platform control and external contribution in third-party development: the boundary resources model

Ghazawneh & Henfridsson (2013)

Keywords: platform, third-party development, boundary resources model, APIs, resourcing, securing.
Boundary resources.
Enable: innovation, design and development of new func to the platform.
Control: the platform and its evolution in some desired direction.

Resourcing happens when:

1. platform owners design boundary resources for extending the scope and diversity of the platform
2. Third-party developers use these resources in their application development.

Securing occurs when: platform owner designs boundary resources for controlling the platform.

Inherent tension between resourcing and securing.

- Boundary resources model can be seen as a useful lens with which to understand this balancing act.

Keywords:platforms, third-party development, boundary resource model, API, resourcing

Additional readings
Reuver & Sørensen (2017) - The digital platform: a research agenda
Gillespie (2010) -The politics of 'platforms'

Innovation

Organizing for innovation in the digitized world (bør lese)

Yoo, Boland & Lyytinen (2012)

Technical perspective on platforms.

Platforms extendibility. Opening for other parties to develop, promotes innovation. Core does not need to be stable, only the interface. Platform core is continuously evolving. Three types of components in any platform system:

1. The complements, which change over time. Apps.
2. Core components, remain stable as the complements change
3. Interfaces, design rules that allow the core and complements to operate as one system. Ie boundary resources.

The important part of the core is the interface, not the parts behind it.

Combination of stability and variety is accomplished through the "stable, yet versatile" interfaces.

Generative innovation: a comparison of lightweight and heavyweight IT

Bygstad (2016)

Innovation and platforms. Generative innovation, generative: take already existing components, recombining and create new functionality. Eg. assignment 2, take 2 APIs connect and make something new.

Lightweight and heavyweight IT.

Heavy - big large scale of the system, put into the core of the system.
Light - easy to incorporate, eg. apps, extension to the core.

Additional readings
Msiska & Nielsen (2017) - Innovation in the fringes of software ecosystems: the role of socio-technical generativity

Design

Distributed development to enable user participation: Multilevel design in the HISP network

O. H. Titlestad, K. Staring, J. Braa (2009)

A bit more general. Technology, and a more social view. Boundary spanners and Scaffolding concepts.

Scaffolding - Social set up around the platform. In innovation systems you are never finished, so the scaffoldings built around has to be sustainable.

Scaffolding not only the social activities supporting the local activities, it can also be documentation, which makes the activities easier.

Core - bundled apps, eg data-entry in DHIS2. Look at the environment we are in. Trying to implement this into developing countries, eg. doctors diary or commodity dispensing, or monitoring antibiotics. Make this system fit with the other domains. It already can store data, try to use the existing technology, but also have to fit with new domains. Talk to users, understand requirements etc.

- If the domains are too far away from the generic application - make custom apps,
- If not to far - use bundle apps and create what you need.

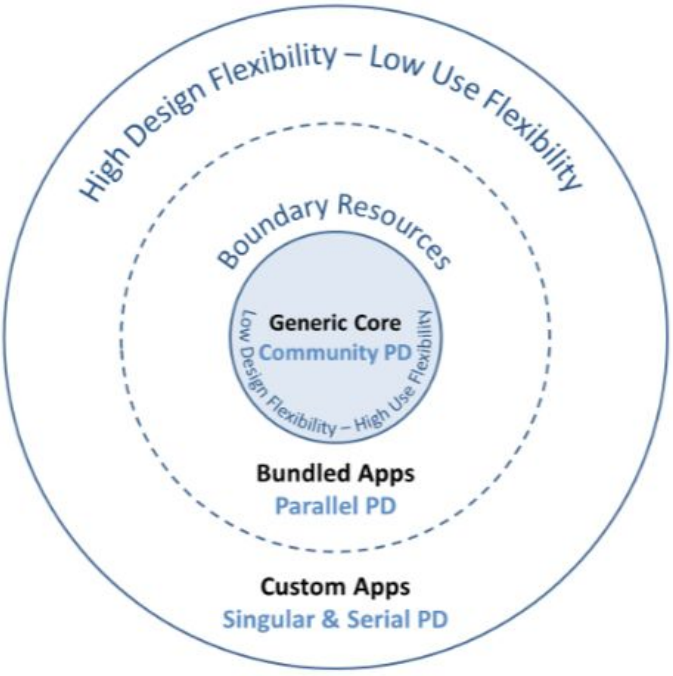
Boundary spanner- Magnus var da en boundary spanner, appear both in the core and also in the local implementation.

P for Platform: Architectures of large-scale participatory design

L. Roland, T. Sanner, J. Sæbø, E. Monteiro (2017)

Participatory design.
Development of requirements to the generic core as a community effort.
4 types of participatory design (PD) along the dimension of scale:

- singular PD: principles and techniques associated with PD. one-to-one relation between dev and users.
    - eg. prototypes, mock-ups, workshops, ++
    - eg. early dev period of DHIS.
- serial PD: across sites and over time.
    - eg. version 2 of DHIS2.
- parallel PD: greater scale. collective filtering of requirements and local appropriation of generic features rather than custom design.
    - eg. workshops and the circulation of software implementers, case studies, software documentation and the software itself.
    - short visits with, workshops with user representatives
- community PD: broader community negotiates generic features. Local adaptations for specific needs without involvement of core dev team.



Layered architecture show that PD may take place even with a large base of heterogeneous users and settings.

Software engineering beyond the project - Sustaining software ecosystems
Y. Dittrich (2014)

DHIS2 relevant.
Design is distributed and needs to be coordinated across heterogeneous design constituencies that, together with the software, build a product specific to socio-technical ecosystems.
Innovation takes place across the whole ecosystem > therefore important to keep contact with users and other actors.

Holistic perspective on the ecosystem when designing.

Software ecosystems challenge some of the very core assumptions of traditional engineering, some of the challenges:

- keeping contact with other actors in the ecosystem
- techniques to support multilevel development and evolution
- managing an overlay of development cycles with different rhythms
- documentation and modelling support for continuous development

Programming and open source

Principled design of the modern Web architecture
Fielding & Taylor (2002)

HTTP - define existing protocol.
REST principles, rest is an architectural style.

- Coordinated set of architectural constraints that attempts to minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations.
- Enables caching
- Enables reuse of interactions, dynamic substitutability of components, ++

REST architectural elements:

- Key aspect: Data elements, eg. URL, HTML document, JPEG image, metadata (source link), cache-control +++
    - Info in REST is a resource.
    - REST uses a resource identifier.
    - Representation: sequence of bytes, plus rep. Metadata to describe those bytes.
    - Control data - define purpose of messages, action being requested or meaning of response.
- Connectors: encapsulate activities. All REST interactions are stateless.
    - Client
    - Server
    - Cache
    - Resolver (DNS lookup library)
    - Tunnel (SOCKS. SSL, after HTTP CONNECT)
- Components, roles
    - User agent - eg web browser
    - Origin server - govern namespace for a requested resource
    - Gateway - reverse proxy, client determines when it will use a proxy

REST: layered system. Architecture can consist of hierarchical levels.
Components only communicate with their neighbors.

+ Reduce system complexity
+ Improve efficiency by ex caching
- Adds overhead and latency

Code on demand - simplifies clients, but reduces visibility.
Concludes:

- REST architectural style has succeeded in guiding the design and deployment of modern Web architecture.
- Model for design guidance
- Has been a test for architectural extensions to the Web protocols
- To date, no significant problems caused by it.

Discusses also how to further develop a new protocol, without loosing the advantages of REST.

RESTful Web Services: Principles, Patterns, Emerging Technologies

Puatasso (2013)

REST - taking full advantage of the HTTP protocol.
Skriver mye samme som Fielder & Taylor.
Maturity model:

- Level 0: HTTP as a tunnel. Exchange XML documents over HTTP POST request and responses.
- Level 1: Resources. Uses multiple identifiers to distinguish different resources.
- Level 2: HTTP Verbs. Already in HTTP: POST. Adding: GET, DELETE, PUT.
- Level 3: Hypermedia. Hypermedia controls within resource representations.

Maturity level of a service - affects the quality attributes of the architecture of the service.
Only level 0 > basic ability to comminicate and exchange data. Not secure. Difficult to evolve and scale.
REST vs WS - ikke relevant? (ikke lest p. 35 og ut).

Richardson Maturity Model

Fowler (2010)

Maturity model. https://martinfowler.com/articles/richardsonMaturityModel.html

Chapter 4. Free and Open Source Software, in INF5780 Compendium Autumn 2015: Open Source, Open Collaboration and Innovation.

Liester (2015)

Open source.
FOSS - Free and Open Source Software
Criterias/Ten rules of the OSI (Open Source Initiative):

- Free redistribution. License shall not restrict any party form selling or giving away the software as a component (...).
- Source code. Program must include source code, and distribution must include source code as well as compiled form.
- Derived works. Licence must allow modifications and derived works.
- Integrity of the author's source code. License may restrict source-code from being distributed in modified form only if license allows the distribution of "patch files".
- No discrimination against person or groups. License must not discriminate.
- No discrimination against fields of endeavor. Can be used in genetic research as well as business.
- License must not restrict other software.
- License must be technology-neutral. Provision of license cannot be done based on individual technology or style of interface.

Other types (Not FOSS):
Freeware - free to use, source code not available.
Negware - as freeware, but boxes will disappear after fee is paid.
Adware - as freeware, but display advertisement.
Crippleware - as freeware, but functionality restricted in free version.
Shareware - free trial period, then paid.

History - code sharing used to be the norm.
Unix..

From lecture:
Commercial - not free at all
Shareware - free, redistributable. Not unlimited use, source code not available or modifiable.
Freeware - as above.
Royalty-free libraries - free, redistributable, unlimited use, source code available, but not modifiable.
Open source - free, redistributable, unlimited use, source code available and modifiable

Fra forelesning:
Four freedoms for software:
0. Freedom to run the program, for any purpose
1. Freedom to study how the program works, and change it to make it do what you wish
2. Freedom to redistribute copies so you can help your neighbor
3. Freedom to distribute copies of your modified version to others. By doing this you can give the whole community a chance to benefit from your changes.

Exploiting and Defending Open Digital Platforms with Boundary Resources: Android's Five Platform Forks.

Karhu, Gustafsson & Lyytinen (2018)

Boundary resources.
Def: Open digital platform (ODP) - extensible core that is open for third parties to contribute improvements or add complements (de Reuver 2017 a research agenda).
Open-platform strategies and their effects.
Platform forking forms a competetive platform strategy that directly attacks and deminishes the host platform's competetive advantage by opportunistically exploiting the platform core and the complementary resources.
Boundary resources are critical in extracting (or inhibiting) all four types of rents, and are, therefor, a key determinant for an ODP's competitive advantage.
Defending against forking through platform governance

- ODP governance too loose.
- Need control to defend against competing platforms.
- ODP managers must pay closer attention to how they design boundary resources.
- Avoid forking is difficult because boundary resources are mostly designed at initial stages.
    - Eg. if Google had restricted app distribution on Android for third-party app stores, Amazon would have been unable to create a forked platform with its own app store.

Conclude: ODP ecosystem, not only cooperative environment but also as a hostile competitive environment demanding proactive strategizing (ie avoid forking).

Other (additional)
Kornbluh (2018) - The Internet's Lost Promise (Foreign Affairs Magazine)