

Lecture 12

Software architecture, maintenance & evolution

07.4.2014

Week	Lecture	Exercise
10.3	Quality in general; Quality management systems	Patterns
17.3	Dependable and safety-critical systems	ISO9001
24.3	Work planning; effort estimation	Code inspections
31.3	Version and configuration management	<i>Effort estimation</i>
7.4	Role of software architecture; product families; software evolution	?
14.4	Specifics of some domains, e.g. web system and/or embedded and real time systems	Break?
21.4	Easter	Break?
28.4	Software business, software start-ups	?
5.5	Last lecture; summary; recap for exam	?

Content of today's lecture

- Remaining topics from last week
- Role of software architecture
 - Architecture and processes and project management
- Software evolution
 - We used to have a course about this

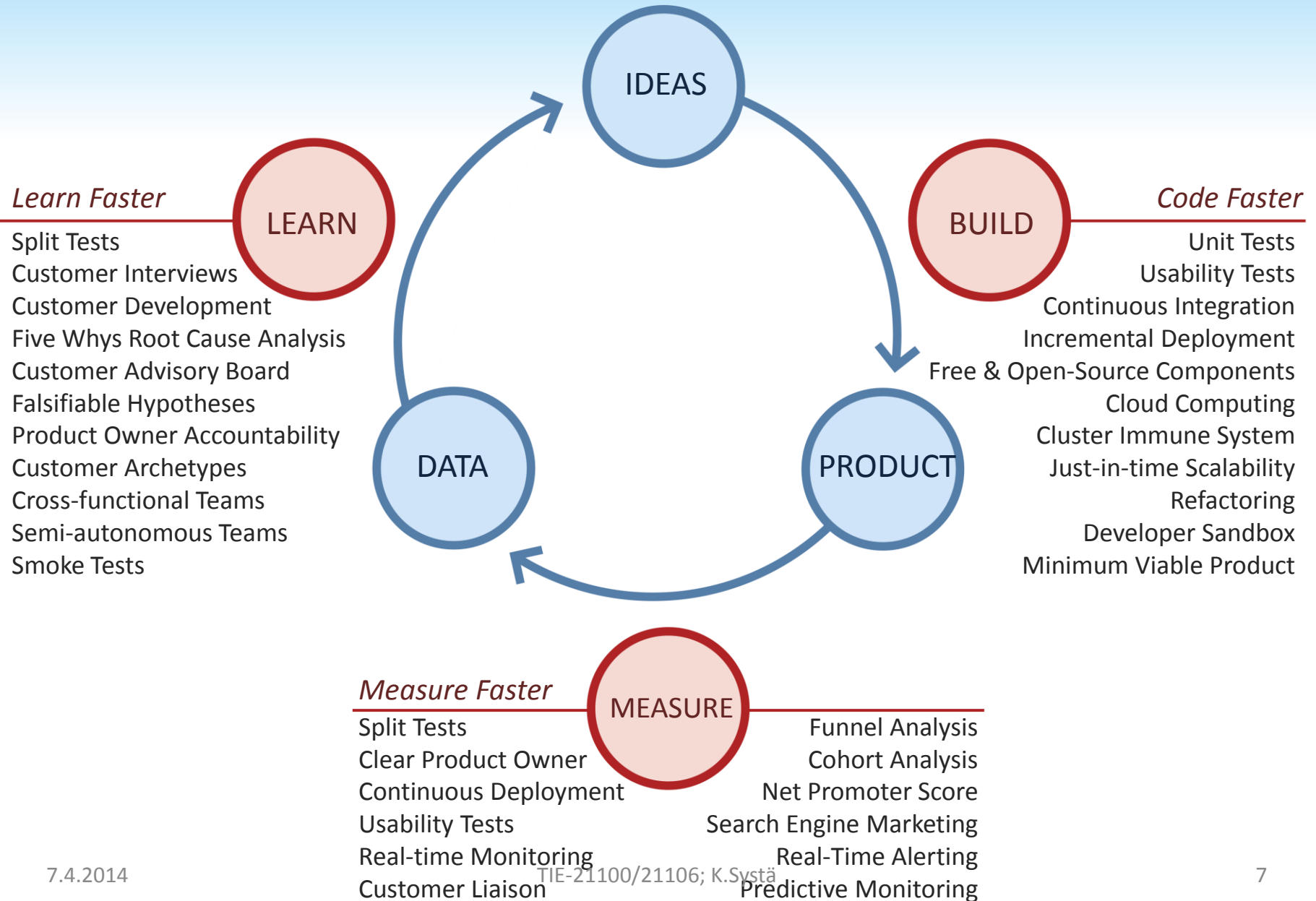
**31.03.2014 WE RUN OUT OF
TIME AND STOPPED HERE.
FOLLOWING SLIDES WILL BE
COVERED 7.4**

Release management

About release management

- Releases go to external customers/users the vendor should be able to answer questions on that particular release.
- Often include major and minor releases
 - Powerpoint 14.0.7116.5000 (32-bit)
 - Thunderbird 17.0.11
- Customer-specific and mass-market SW impose different challenges
- When problem occurs the HW configuration should be available
- Full traceability is expected
- Releases should be well tested, well documented, ...
- Installation/deployment need to be planned
- Updates need to be planned
 - Technical, commercial

But on the other hand, sometimes ...



Continuous delivery/deployment; A/B testing

- Sometimes it is important to get fast feedback from market
 - Lean Startup
- Also part of DevOps
- Used for development of customer software and Internet-services
- A/B testing (split testing):
 - Randomly give different users different versions of the system and systematically compare.

One claim

(<http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment>)

CONTINUOUS DELIVERY



CONTINUOUS DEPLOYMENT



Key points of release management

- System building is the process of assembling system components into an executable program to run on a target computer system.
- Software should be frequently rebuilt and tested immediately after a new version has been built.
 - This makes it easier to detect bugs and problems that have been introduced since the last build.
- System releases include executable code, data files, configuration files and documentation.
- Release management involves making decisions on system release dates, preparing all information for distribution and documenting each system release

Summary the whole topic

- Configuration management is the management of evolving system
- Main processes deal with change management, version management, system building and release management
- Change management assesses proposals and deciding
- Version management keeps track of the different versions
- System building is about assembling into executable system
 - executable code, data files, configuration files and documentation
 - How about scripting languages?
- Systems are frequently rebuilt

Pointers to material

- Haikala & Mikkonen: Chapter 13 "tuotteenhallinta" (Product management)
 - A bit short chapter – reading of additional material recommended
- Sommerville: Chapter 25
- Software Engineering knowledge (language bit boring)
<http://www.computer.org/portal/web/swebok/swebokv3>
- About continuous integration
 - <http://www.martinfowler.com/articles/continuousIntegration.html>
- About continuous deployment:
 - <http://gofore.github.io/continuous-deployment/#/>

Role of software architectures

TIE-21300 Ohjelmistoarkkitehtuurit, periodit 3-4 2013-2014

[Opinto-opas](#)
[tut.ot.ohar](#)

[Pääsivu](#)
[Aikataulu](#)
[Luennot](#)
[Harjoitukset](#)
[Harjoitustyö](#)
[Tenttiarkisto](#)
[Arvostelu](#)
[Kirjallisuutta](#)
[Linkkejä](#)
[Keskusteluryhmä](#)
[Paikalliset sivut](#)
[Tulokset](#)

[Tulostusversio](#)
[Sivukartta](#)

[Ohjelmistotekniikan
opinnot.](#)

Luentoajat

Kurssin luennot ovat 3. periodilla keskiviikkoisin 10-12 salissa TB109 ja torstaisin 14-16 TTY:n salissa TB103, 4. periodilla (10.3. alkaen) keskiviikkoisin 10-12 TTY:n salissa TB109.

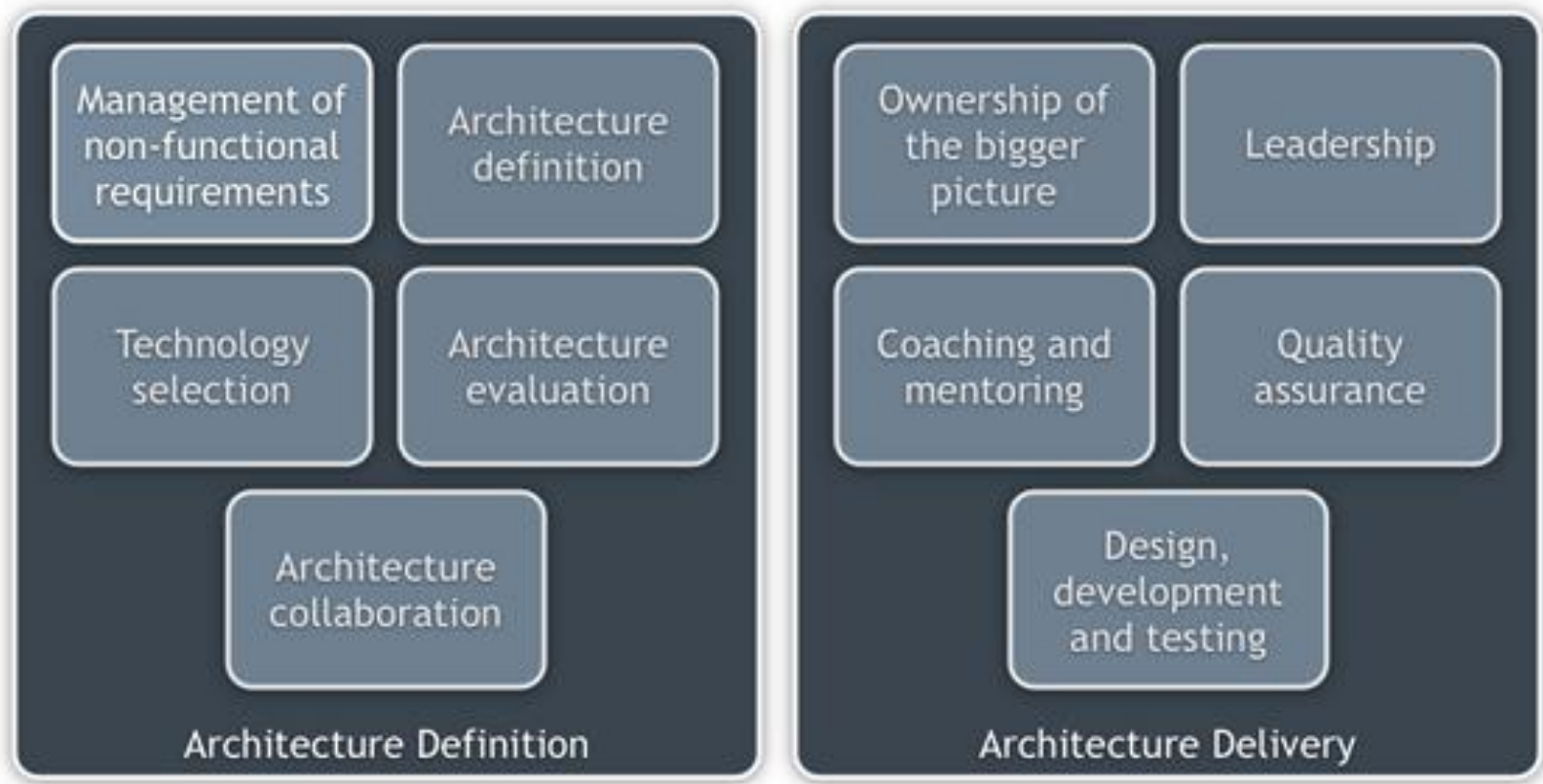
Muut OSCu-hankkeen opetustahot osallistuvat luennoille videoneuvottelun välityksellä.

Luennot 2014 kevät

Aihe	Luento	Kalvot	Tallenteet
Aloitusluento, johdanto arkkitehtuurijuttuihin	8.1.	Kurssin esittely Johdantoa arkkitehtuuriin	Samuel hukkasi tallennukset, katso Jomppaa, se on komeampikin. Jomppa 1/4
Lisää arkkitehtuurista ja vaatimuksista	9.1.	Arkkitehtuurijohdantojatkot	kalvot
Arkkitehtuurin dokumentointi, UML-kaaviot	16.1.	kalvot	luentotallenne(puolikas?)
Vierailuluento 1: Janne Viitala, Sandvik	17.1.	kalvot	luentotallenne
Komponentit	22.1.	kalvot	luentotallenne
Suunnittelumallit	22.1.	kalvot	luentotallenne
Vierailuluento, Mika Siikarla, Bitwise - arkkitehtuurityö käytännössä	29.1.	kalvot	luentotallenne (2012)
Harjoitustyön esittely, arkkitehtuurityylit	30.1.	harkkatyoesittely, arkkitehtuurityylit (abstrakti tehdas suunnittelumallikalvoissa)	luentotallenne
Arkkitehtuurityylit jatkuvat	5.2.	kalvot	luentotallenne
Sessio 1: lainsäädöntö & suunnittelu	6.2.	ennakkotehtävät	
Pilveä ja koneita	12.2.	kalvot	luentotallenne
sessio 2: MDD & adaptiivisuus	13.2.	ennakkotehtävät	materiaalia sessiosta
Vierailuluento, Janne Sinivirta, Nitor Creations Ltd. - Lean Architecture	19.2.	luentotallenne -->	Agile Finland -luento samasta aiheesta TTY:n luento (Mac ja Adobe connect enähtäessopivia...)

Role of architect (the person)

<http://www.codingthearchitecture.com/pages/book/role.html>



Definition

(<http://csse.usc.edu/csse/TECHRPTS/1995/usccse95-500/usccse95-500.pdf>)

A software system architecture comprises

- A collection of software and system components, connections, and constraints.
- A collection of system stakeholders-need statements.
(a collection of system requirements)
- a rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system requirements

Other definitions

Sommerville

- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is **architectural design**.
- The output of this design process is a description of the **software architecture**.

Swebok

- System architecture: an interacting combination of elements to accomplish a defined objective. These include hardware, software, firmware, people, information, techniques, facilities, services, and other support elements,
- Software architectural design (sometimes called high-level design): develops top-level structure and organization of the software and identifies the various components.
- Software detailed design: specifies each component in sufficient detail to facilitate its construction.

Two (three) levels

Sommerville

- **Architecture in the small** is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components.
- **Architecture in the large** is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies.
- **Enterprise architecture** Enterprise architecture is the organizing logic for business processes and IT infrastructure reflecting the integration and standardization requirements of the company's operating model. The operating model is the desired state of business process integration and business process standardization for delivering goods and services to customers.[]

Advantages of explicit architecture

- Stakeholder communication
 - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
 - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
 - The architecture may be reusable across a range of systems
 - Product-line architectures may be developed.

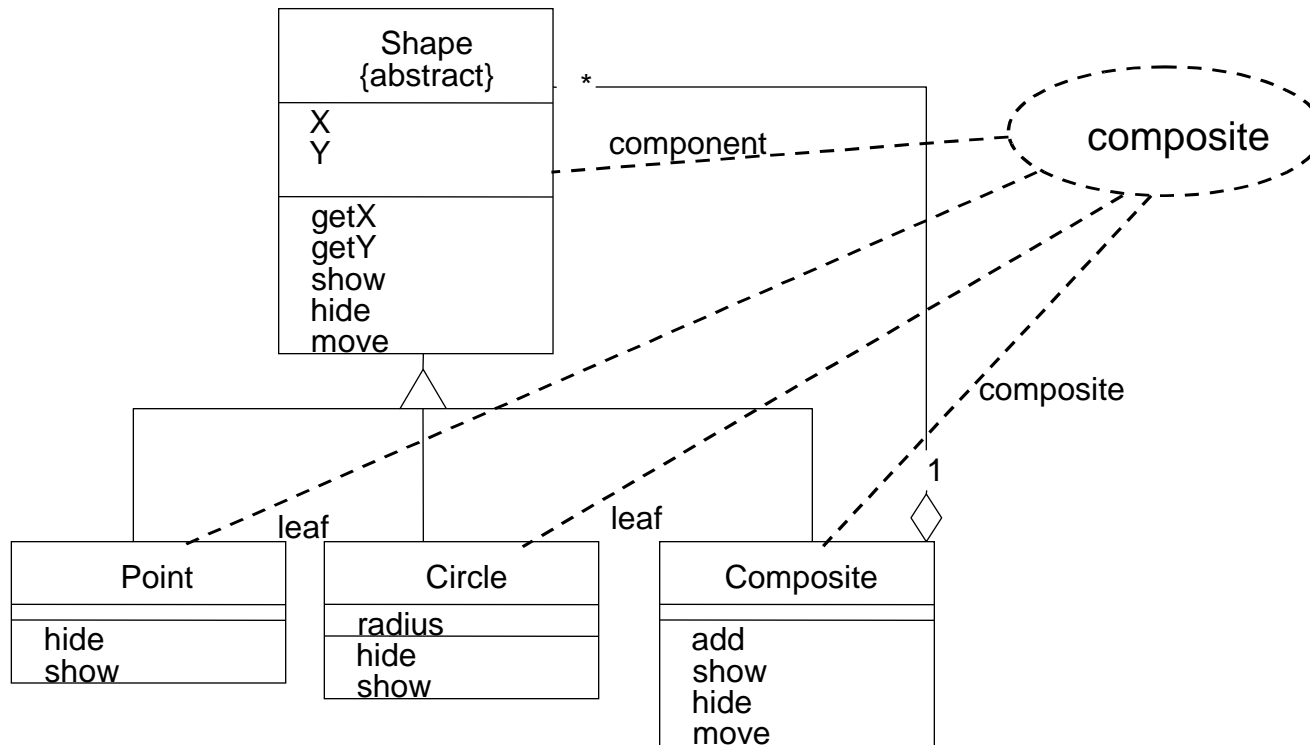
Architectural representations

- Simple, informal block diagrams showing entities and relationships are the most frequently used method for documenting software architectures.
- But these have been criticised because they lack semantics, do not show the types of relationships between entities nor the visible properties of entities in the architecture.
- Depends on the use of architectural models. The requirements for model semantics depends on how the models are used.

Box and line diagrams

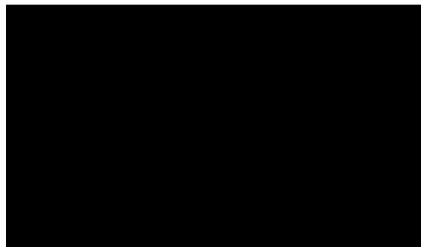
- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- However, useful for communication with stakeholders and for project planning.

UML



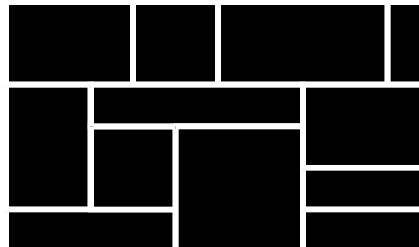
What is architecture design (translated from Haikala&Mikkonen)

Specification

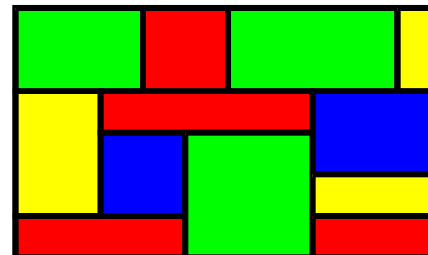


**Product from customer
point of view**

Architecture design



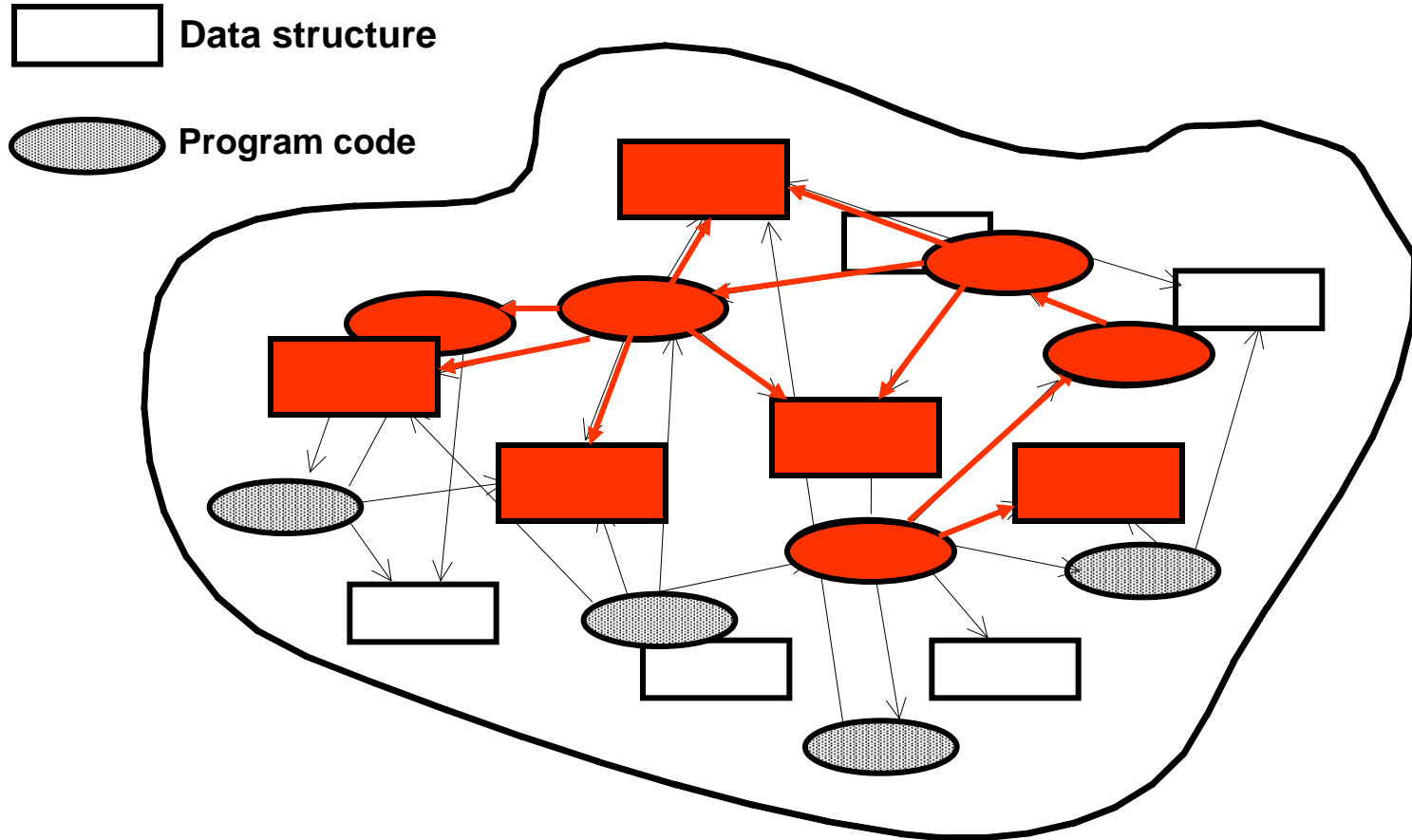
Product from implementation point of view
architecture principles
component. interfaces,
database solutions,
distribution
-process and communication structure,
-Technology selections, ...



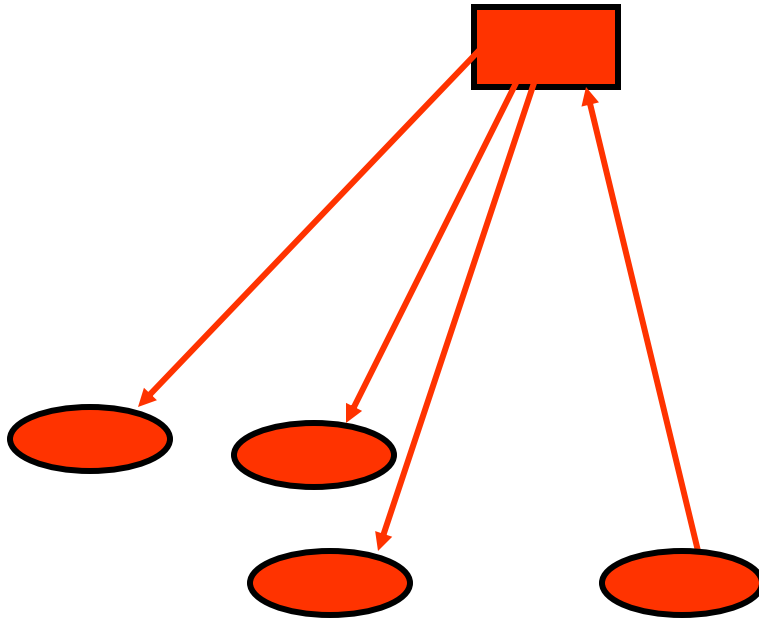
Implementation



Spagetti



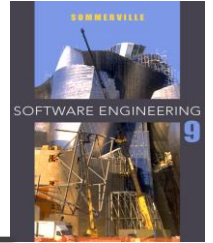
Structured programming



Architectural design decisions

- ✧ Is there a generic application architecture that can be used?
- ✧ How will the system be distributed?
- ✧ What architectural styles are appropriate?
- ✧ What approach will be used to structure the system?
- ✧ How will the system be decomposed into modules?
- ✧ What control strategy should be used?
- ✧ How will the architectural design be evaluated?
- ✧ How should the architecture be documented?

Architectural design decisions



- ✧ Architectural design is a creative process so the process differs depending on the type of system being developed.
- ✧ However, a number of common decisions span all design processes and these decisions affect the non-functional characteristics of the system.
- ✧ The architectural decisions are in a key role
 - The reasoning should also be documented

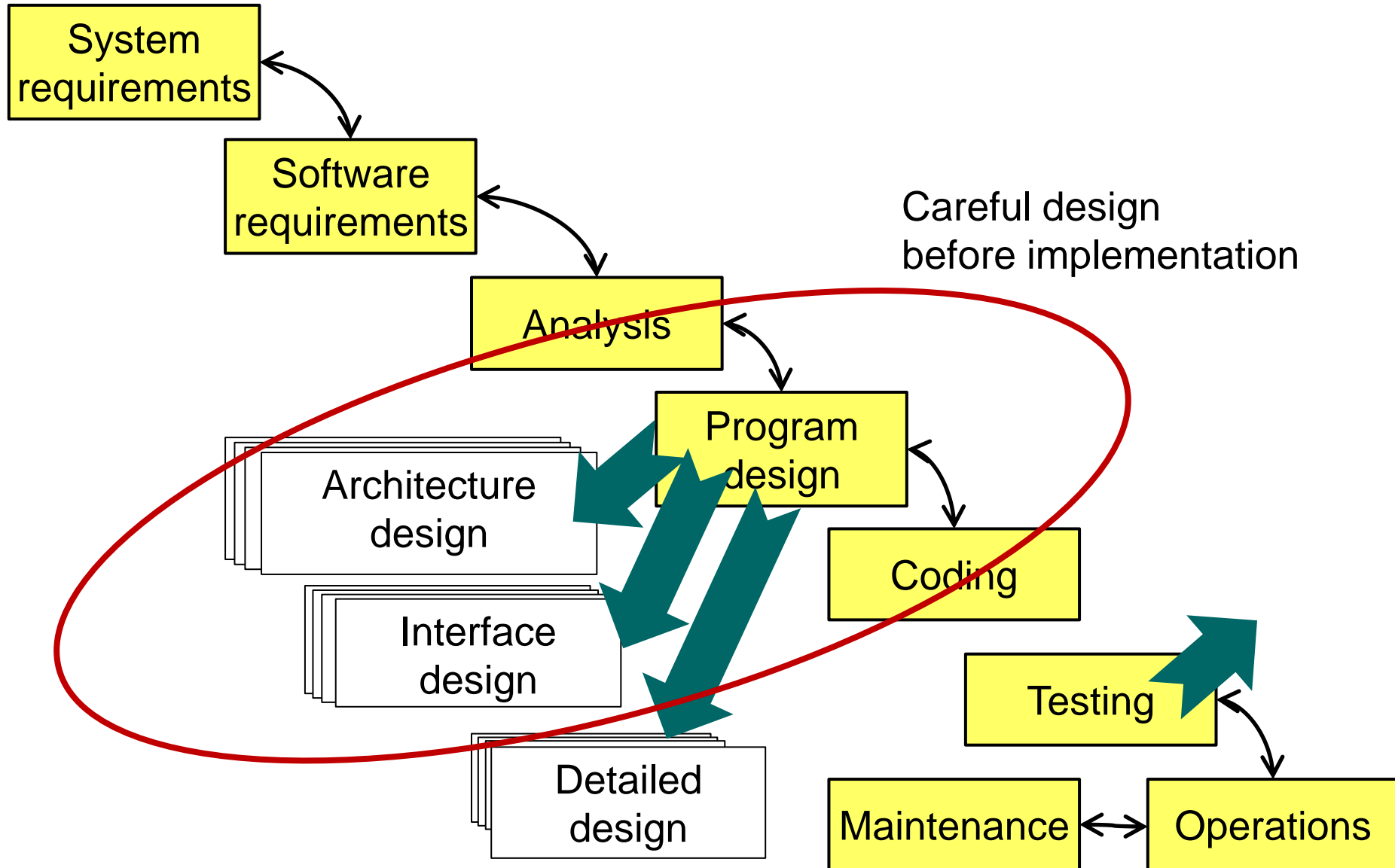
4 + 1 view model of software architecture

- ✧ A logical view, which shows the key abstractions in the system as objects or object classes.
- ✧ A process view, which shows how, at run-time, the system is composed of interacting processes.
- ✧ A development view, which shows how the software is decomposed for development.
 - Haikala&Mikkonen: toteutusnäköymä; implementation view
- ✧ A physical view, which shows the system hardware and how software components are distributed across the processors in the system.
 - Haikala&Mikkonen: sijoittelunäköymä (deployment view)
- ✧ Related using use cases or scenarios (+1)

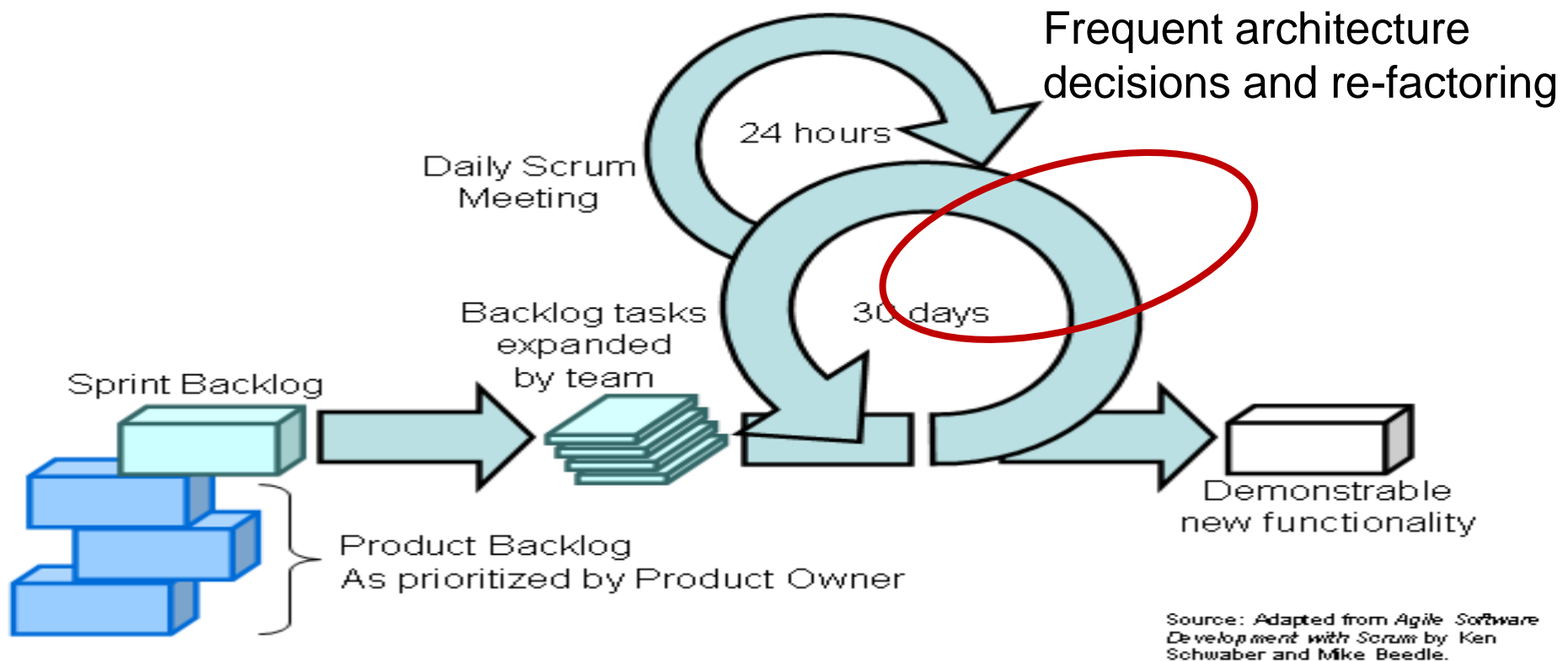
Goals of architectural design

- Performance
 - e.g. no extra communication
- Understandability
 - Developers and stakeholders understand
 - KISS (Keep It Simple, Stupid)
 - Should be familiar
- Maintainability
 - Separation of concerns
 - Changes are local
- Work division

Architecture and waterfall



Architecture and Scrum

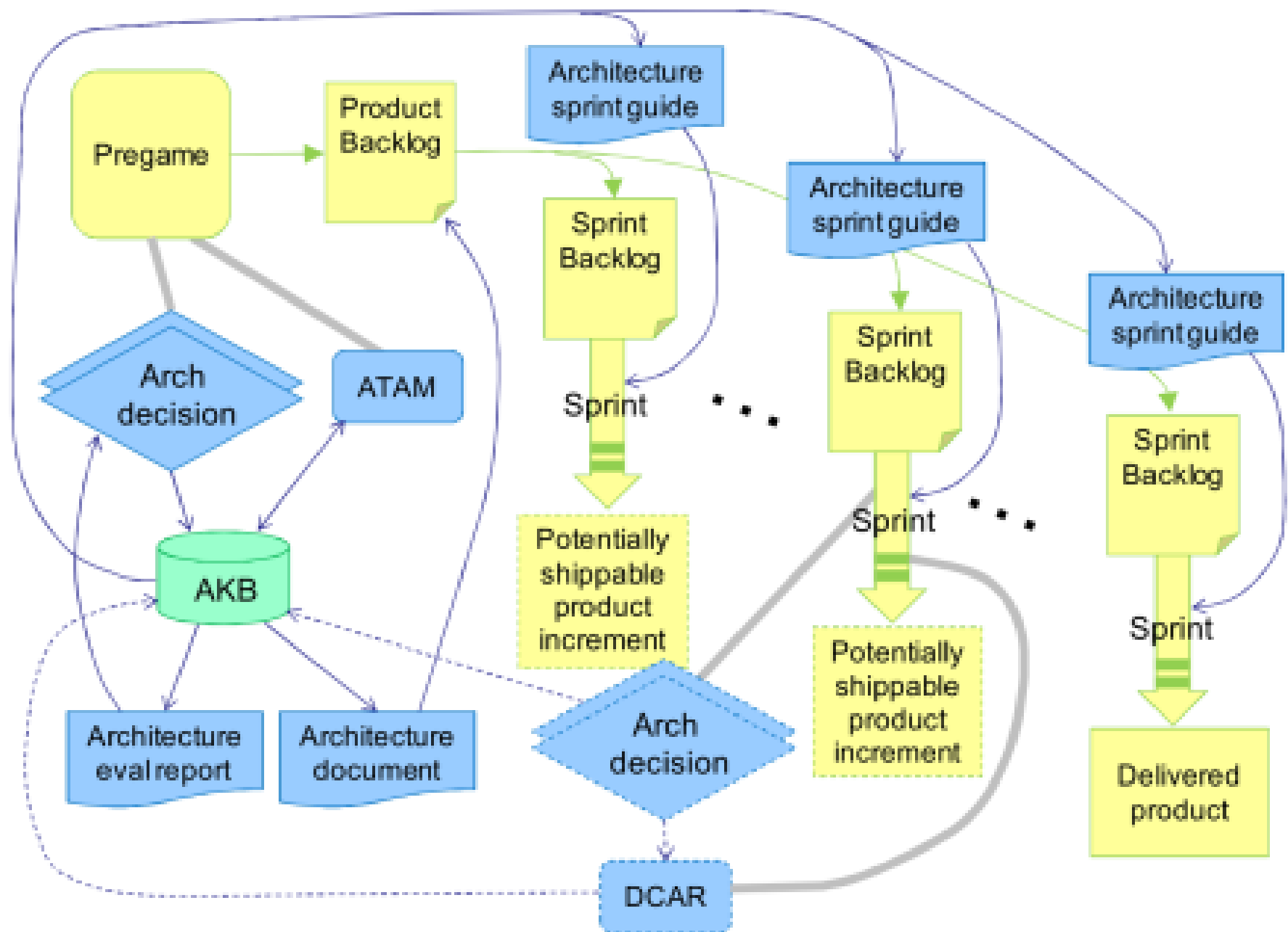


Architecture design in Agile processes

- The idea of most Agile processes is that system is developed iteratively.
- The architecture selected for first sprint is not necessarily valid for remaining sprints
 - ⇒ Refactoring is needed
 - Could use separate re-factoring sprints, or
 - add re-factoring tasks to backlog
- Often it is necessary decide the main architecture principles in a "pregame"
 - Especially big projects that require several teams

Veli-Pekka Eloranta and Kai Koskimies. 2012. Aligning architecture knowledge management with Scrum. In *Proceedings of the WICSA/ECSA 2012 Companion Volume (WICSA/ECSA '12)*. ACM, New York, NY, USA, 112-115.

- architectural knowledge management (AKM)
- architectural knowledge base (AKB)
- Architecture evaluation methods ATAM ja DCAR



Conway's law

- organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations
- Conway, Melvin E. (April 1968), "How do Committees Invent?", Datamation **14** (5): 28–31,
 - Reprint available:
<http://www.melconway.com/research/committees.html>

Software maintenance and evolution

Once upon a time

OHJ-3100 Ohjelmien ylläpito ja evoluutio - Luennot 2012

Muutoksia: 12.6.2012 Sivu perustettu

[Opinto-opas](#) Luennot perustuvat kurssikirjaan, jonka sisällysluettelo ja muut tiedot löytyvät [Materiaali](#)-sivulta.

[Pääsivu](#) Luentojen aikataulu päivitetään tälle sivulle sitä mukaa kun se on tiedossa.

[Arvostelu](#)

[Harjoitukset](#) Kalvoista on kaksi eri tulostusvaihtoehtoa: 2 kalvoa/sivu (2 k/s) ja 6 kalvoa/sivu (6 k/s).

[Harjoitustyö](#)

Luennot

Luentojen aikataulu:

[Materiaali](#)

[Osallistumisrajoitukset](#) **I-periodi**

[Palaute](#)

[Uutisryhmä tut.ot.evo](#) ti 28.8.2012 Kurssin järjestelyt ([2 k/s](#)) ([6 k/s](#)), Peruskäsitteitä ([2 k/s](#)) ([6 k/s](#))

[Ohjelmistotekniikan](#)

to 30.8.2012 Analysointitekniikoita ([2 k/s](#)) ([6 k/s](#))

[opinnot](#)

ti 4.9.2012 Metriikat

to 6.9.2012 Metriikat [jatkoa] ([2 k/s](#)) ([6 k/s](#))

ti 11.9.2012 Ylläpitotoiminnot ([2 k/s](#)) ([6 k/s](#))

to 13.9.2012 Ylläpidettävät ohjelmat ([2 k/s](#)) ([6 k/s](#))

ti 18.9.2012 Ohjelmien ymmärtäminen ([2 k/s](#)) ([6 k/s](#))

to 20.9.2012 Takaisinmallinnus

ti 25.9.2012 Takaisinmallinnus [jatkoa] ([2 k/s](#)) ([6 k/s](#))

to 27.9.2012 Uudistaminen ja uudistamismallit ([2 k/s](#)) ([6 k/s](#))

ti 2.10.2012 Ohjelmistoarkkitehtuurien ylläpito ([2 k/s](#)) ([6 k/s](#))

to 4.10.2012 Harjoitustyön esittely

ti 9.10.2012 Olioiden tunnistaminen ([2 k/s](#)) ([6 k/s](#))

to 11.10.2012 Kielikonversio ([2 k/s](#)) ([6 k/s](#))

Lehman, M. M.. On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle. Journal of Systems and Software 1: 213–221, 1980.

- Claim: 30%-70% on top of initial development is spent for maintenance
- Lehman divides software into three categories
 - S-type programs are those that can be specified formally
 - P-type programs cannot be specified but an iterative process is used to find a working solution;
 - E-type programs are embedded in the real world and become part of it thereby changing it

Laws of evolution (by Lehman)

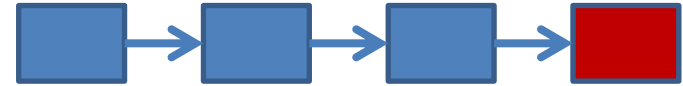
- **Continuing Change.** All E-type systems need continuous adaptation or they become progressively less satisfactory.
- **Increasing Complexity.** Because all E-type systems evolve, their complexity increases unless work is done to reduce it.
- **Self-Regulation.** E-type system evolution process is self-regulating with distribution of product and process measures close to normal.
- **Conservation of Organizational Stability.** The average effective global activity rate in an evolving E-type system is invariant over a product lifetime.
- **Conservation of Familiarity.** As an E-type system makes everything associated with it evolve, developers, sales personnel, users, must maintain mastery of its content and behavior to achieve satisfactory evolution. Because too rapid growth diminishes that mastery, the average incremental growth remains invariant as the system evolves.
- **Continuing Growth.** The functional content of an E-type system must be continually increased to maintain user satisfaction.
- **Declining Quality.** The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
- **Feedback System.** E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

Another view to evolution

- Software is changed because
 - It has bugs or poor performance
 - It needs new features
 - Environment changes
- Existing software is also reused in other projects (at least partly)

Example: SW of Smart phone

- New products
 - with new SW features,
 - utilizing improved HW



have been developed over several years

Dear developer,

Thank you for your improvement ideas, thus (sic) Symbian is in maintenance mode and no new features will be implement[ed] without extremely good reason (business case). We have written down your ideas for future development if there is a chance that new features will be released.

*Kind Regards
Nokia Developer support*

Terms

- Legacy software (perintöohjelmat)
- Program comprehension (ohjelmien ymmärtäminen)
- Program analysis (ohjelmien analysointi)
- Software metrics (metriikat, mittaaminen)
- Refactoring
- Reuse engineering, reuse re-engineering
- Wrapping (kääriminen)

Program analysis

- Needed for comprehension
- Architecture is not known, not documented, or documentation is out of date
- Sometimes "reverse engineering" is needed
 - New documents about program structure, data, control flow....
-

Material

- Role of architect:
<http://www.codingthearchitecture.com/pages/book/role.html>
- About continuous integration
 - <http://www.martinfowler.com/articles/continuousIntegration.html>
- About continuous deployment:
 - <http://gofore.github.io/continuous-deployment/#/>
- Good article about architecture " On the Definition of Software System Architecture"
<http://csse.usc.edu/csse/TECHRPTS/1995/usccse95-500/usccse95-500.pdf>
- Lehman, M. M.. On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle.
Journal of Systems and Software 1: 213–221, 1980

Week	Lecture	Exercise
10.3	Quality in general; Quality management systems	Patterns
17.3	Dependable and safety-critical systems	ISO9001
24.3	Work planning; effort estimation	Code inspections
31.3	Version and configuration management	<i>Effort estimation</i>
7.4	Role of software architecture; product families; software evolution	?
14.4	Software business, software start-ups	Break?
21.4	Easter	Break?
28.4	Specifics of some domains, e.g. web system and/or embedded and real time systems	?
5.5	Last lecture; summary; recap for exam	?