Fundamentals of Software Architecture

AMICT 2014

Dr. Antti-Pekka Tuovinen

University of Helsinki

Why *Software Architecture*?

- Sofware used to be just one piece of a system
- Software is everywhere
- Applications need to work on global scale

- 20 years of active research, decades of practice

Outline of the talk

- What is Software Architecture?
- Why and when is Software Architecture important?

SOFTWARE ARCHITECTURE

What is it?

- Definition (Software Engineering Institute / Carnegie-Mellon Universisty)

> ” The software architecture of a system is the *set of structures needed to reason about the system*, which comprise software elements, relations among them, and properties of both.”

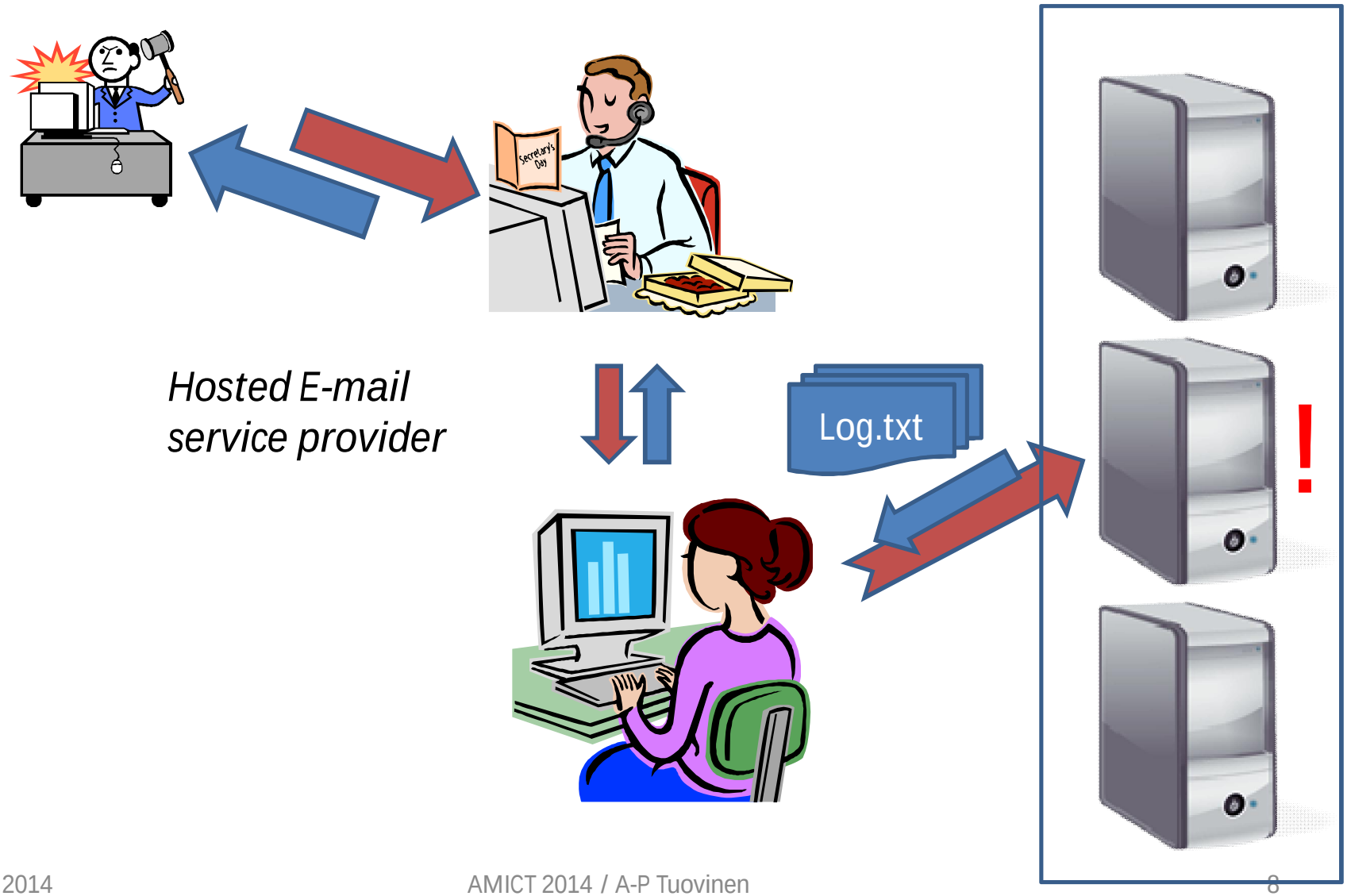
Ok, but what does it *mean*?

- The definition does not tell us what those structures actually are
  - Not just one structure, but a *set* of structures
  - Any *appropriate* collection of elements (code modules, run-time components, virtual machines,...) and their relationships (dependencies, connections) can form a structure
  - Different types of systems have different structures
  - Structures are abstract – they show only selected, essential details needed for reasoning about the system

...huh?

- The key thing is that the structures enable us to *reason about the macro-level qualitites of the software system* that are important for a stakeholder
  - Client/customer, end-user, developer, maintainer, managers, business-owner, authorities, ...
  - Suitability, dependability, efficiency, performance usability, modifiability, scalability, security, external dependencies, ...
  - The question to ask is: if the system is built the way that the structures show, will it have the desired qualities?
- Software architecture is a *means to an end* (a tool) – it does not have value of its own (like art has)

Example - Rackspace



System under study

- A system for storing, accessing and searching the data in the log files produced by Email servers for technical customer support
- Mission goal & requirements
  - Support solving customers' email problems by making the data in the email server logs easily accessible
  - The log data available via the system needs to be fresh to solve acute problems
  - The log data should be kept for a period of time to support analysing past problems
- Rackspace built three versions of the system that had different architectures

V. 1

- *Local log files on the servers*
  - The techinal support person needs to ask as a system operator/engineer to log in to the customer's email server in person and search in the log files on the server machine to figure out what's wrong
  - To make this easier, they wrote one script that would automatically connect to a number of servers and execute a *grep* command locally to search in the log files
  - The operators could control the search by changing the arguments given to the *grep* command
- Problems:
  - Executing the searches on the server machines started to impact their performance negatively
  - They always needed an operator to do a search; the technical support persons could not do it by themselves

V. 2

- *Central database for all log data*
  - Email servers send every few minutes their most recent log data to a central database server that stores the data in a relational database (moving the log data off the email servers)
  - Technical support persons have a web-UI to execute pre-programmed queries on the database server
  - Because there were more and more updates all the time, they started to use batch inserts at every 10 minutes so that the database server could handle all the requests with an acceptable performane
- Problems:
  - When the amount of data and the number of queries kept increasing, the database server wash pushed to its limit, which lead to frequent failures
  - The searches were getting slower, data was lost due random failures (there were no backups), and only few days worth of log data could be kept in the database

V. 3

- *Indexing cluster*
  - The log data was streamed from the email servers into a cluster of commodity servers storing the data into a distributed (Hadoop DFS) file system (with triple copies of data)
  - A Map-Reduce –job indexes the individual log files and builds a complete index of all the data every 15 minutes
  - Technical support personnel has a web interface to search the log data, as before
  - Searching the index is fast but programming a new kind of search takes some hours
  - Complete backups, log data is kept for six months
- No problems ☺

Comparing the solutions

	V. 1	V. 2	V. 3
Functionality			
Scalability – amount of data and queries	bad	poor	Very good
Delay – time to wait before new data is available	No delay	10 min.	15 min.
Flexibility – new searches	good	good	satisfactory

Reflections

- All three versions of the system provide basically the *same functionality (service)* to its users
  - In all cases, the same data is made available to the technical support personnel
- The technical solution is not based on functional requirements
  - The architecture is a *separate choice* from the functions provided by the system

Reflections

- The architecture of the system is mostly determined by the required *qualities* (a.k.a. ’non-functional requirements’)
  - Scalability, latency/delay, modifiability
- Scalability is by far the most important quality of the example system
  - A poorly scalable system can not cope with the amount data and the number of queries and totally fails to serve its users

Reflections

	V. 1	V. 2	V. 3
Functionality	✓	✓	✓
Scalability – amount of data ad queries	bad	poor	Very good
Delay – time to wait before new data is available	No delay	10 min.	15 min.
Flexibility – new searches	good	good	satisfactory

Reflections

- As we improve the scalability of the system in the 3. version, the other qualities suffer a little
  - The delay in the availablity of new data grows up to 15 min
  - Modifiability is slightly worse because it takes longer time to implement new kind of queries
- We *trade-off qualities* against each other (there is no free lunch)
  - Priorisation of the qualities in the example system:

Scalability > Delay > Modifiability

Arhitectural design decisions

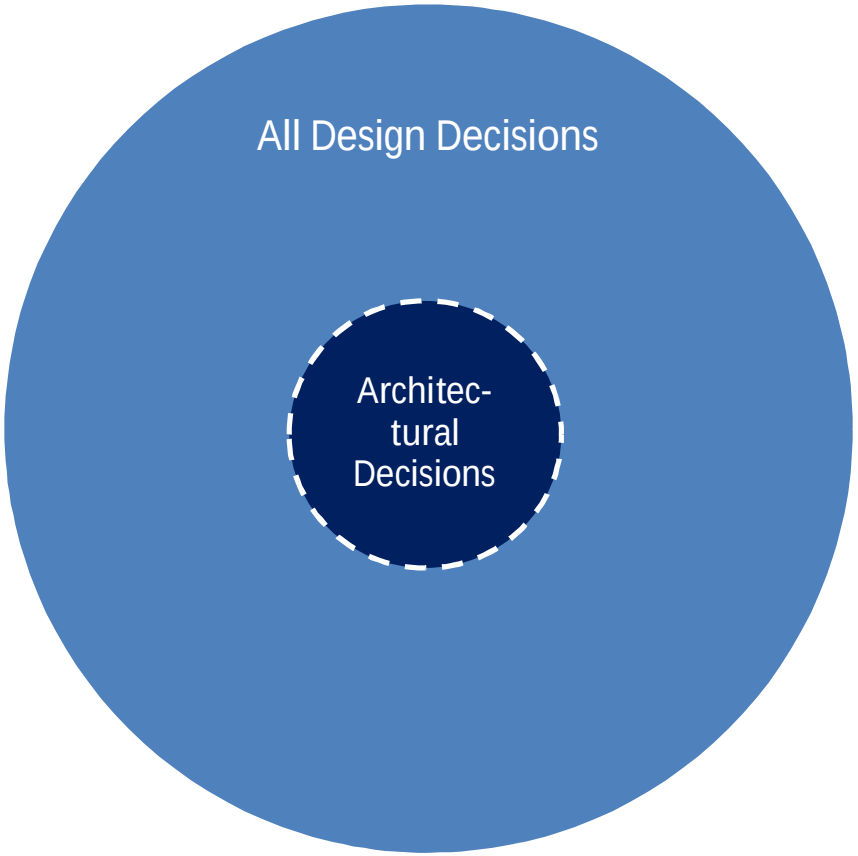
- The aspects of the technical solutions discussed above are the key choices that determine the architecture of the three versions of the example system
- So, we give an alternative definition for software architecture: *SWA is the set of design decisions that are important for achieving the overall qualities of a system*
  - Based on those design decisions, we can build a system that meets its quality requirements
- The *domain* and the *specific requirements* of a system determine which design decisions are architectural

Typical architectural design decisions

- Partitioning of the system under design (SUD) into subsystems/main components and determining their role, functionality and mutual dependencies and collaboration (separation of concerns)
- Identifying the interfaces of the SUD (UI, APIs) and separating them from their implementation (information hiding)
- Decisions that impact the ease of development and maintainability
- Allocation of the software elements into the run-time environment, which affects the performance, dependability and security of the SUD
- The storage and access solutions for the data managed by the SUD
- Use of technologies/platforms and reference architectures that they promote or dictate

Architecture vs. *Design*

Drawing the line between Architecture and Design is not always easy



But it is usually possible to recognize the design decisions that affect qualities

WHY AND WHEN IS SOFTWARE ARCHITECTURE IMPORTANT?

SWA is important, because

- Architecture acts as a skeleton of a system
  - Every system has an architecture – it is better to choose it consciously
  - There is no single, absolutely right architecture, but there are *more or less suitable* skeletons for the job
  - The skeleton (architectural style) determines the basic capabilities of the system

SWA is important, because

- Architecture influences quality attributes
  - Architecture enables or inhibits qualities such as performance of security
- Architecture is (mostly) independent of functionality
  - But: an unsuitable architecture can make it difficult and expensive to implement functionality (e.g. a very fine grained separation of concerns gets in the way of implementing functions)

SWA is important, because

- Architecture constrains systems (guide rails)
  - Architecture may place constraints on the (detailed) design and implementation that guide the development to the desired direction
  - Constraints help the developers
    - Transferring of wisdom and experience from experts without full transfer of knowledge
    - Promoting conceptual integrity[1] by reducing ”needless creativity” of developers in places where it would be harmful (reducing accidental complexity)
    - Can enforce run-time behaviors that would be difficult to deduce from the code directly

[1] *”A single good idea consistently applied is better than several brilliant ideas scattered across a system”* (Fred Brooks)

When is Architecture important?

- The solution space is small
  - It is hard to find *any* acceptable technical solution
  - There is a lot of *essential complexity* in the design problem
- System failures can cause significant damage
  - People will get hurt or die, equipment or the environment may be damaged, money is lost
- Difficult quality requirements
  - It is *really hard* to make an on-line information system that scales up to a billion users

When is Architecture important?

- New domain
  - A new application domain will have some new kind of problems and technologies that the software developer is unfamiliar with
- Product lines
  - A software product line comprises of products that share a significant amount of implementation (code) and that have a common architecture
  - The common architecture has to be designed to support the common features of the products, and, at the same time, it has to cater for the needed variations in product specific features

When is Architecture important?

- Make a thought experiment: think, what would be the consequences of getting the architecture wrong?
  - If the system is small and simple, your architecture choices won't make a big difference and there is no need to spend too much time on architecture
  - If the system (or the project) has significant risks, it is wortwhile to focus on architecture to mitigate the risks

Summary

- Software Architecture is the set of design decisions that are important for achieving the overall qualities of a system
- Do just enough architecture work to control the technical and project risks of the system to be developed

References

- *Fairbanks, G.*: Just Enough Software Architecture - A Risk-Driven Approach. Marshall & Brainerd, 2010
- *Bass, L., Clements, P., Kazman, R.*: Software Architecture in Practice, Third Edition. Addison-Wesley Professional Series, 2012
- *Hoff, T.*: How Rackspace Now Uses MapReduce and Hadoop to Query Terabytes of Data. HighSclability.com, January 30, 2008 http://highscalability.com/how-rackspace-now-uses-mapreduce-and-hadoop-query-terabytes-data