7 INTRODUCTION TO SOFTWARE ARCHITECTURE

course “software requirements and architecture”

Nov 2022

contents

contents

example 1

feature: call subscriber

old telephone system

- archit.: centralised hw switch
- qualities: reliable, works during power outages, emergency calls get location information

skype

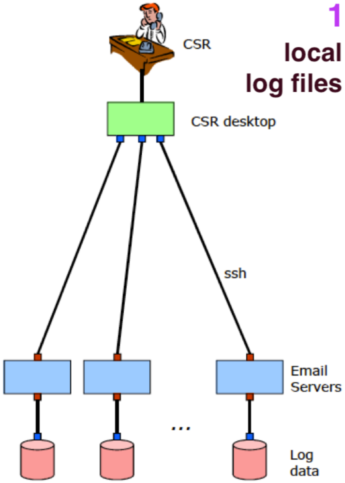
- archit.: peer-to-peer sw
- qualities: scales without central hardware changes, easy to add new features

example 2

- Rackspace is a real company that manages hosted email servers.
- Customers call up for help when they experience problems.
- To help a customer, one must search the log files that record what has happened during the customer's email processing.
- The volume of emails handles kept increasing, so three systems were built to handle the customer queries.
- Three similar systems with different architectures.

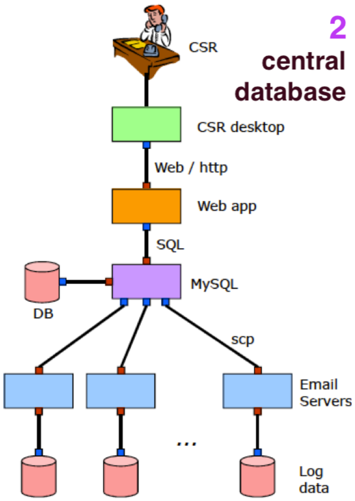
example 2

- There are already several email servers generating log files.
- A script uses `ssh` to connect to each machine and executes a `grep` query on the mail log file.
- Engineers control the search results by adjusting the grep query.
- The number of searches increases.
- The overhead of running those searches on the email servers becomes noticeable.
- It requires an engineer to perform the search.



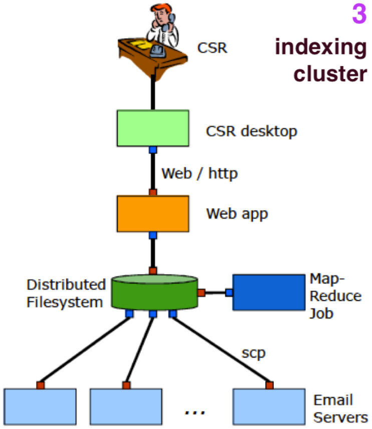
example 2

- Every few minutes, each email server sends its recent log data to a central machine where it is loaded into a relational DB.
- CSRs has access to the DB data via a web-based interface.
- Rackspace is now handling hundreds of email servers.
- The challenge becomes how to get the log data into the DB as quickly and efficiently as possible.



example 2

- This solution saves log data into a distributed file system and parallelizes the indexing of log data.
- It uses ten commodity machines.
- Log data from the email servers is streamed into the Hadoop distributed file system, which keeps 3 copies of data on different disks.
- Jobs to stream log data run every 10 min and take 5 min to complete.
- Index results are about 15 min stale.



contents

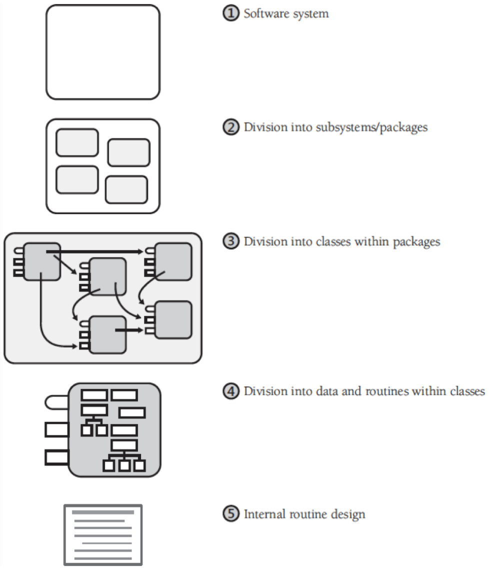
principles

- A principle is a comprehensive and fundamental law, doctrine or assumption.
- Software design principles are key notions that provide the basis for many approaches/concepts.
  - abstraction
  - coupling and cohesion
  - decomposition and modularization
  - encapsulation and information hiding
  - separation of interface and implementation
  - sufficiency, completeness and primitiveness
  - separation of concerns

design

- Software design is the conception or invention of a scheme for turning a specification for computer software into operational software.
- Design is the **activity** that links requirements to coding and testing.
- A good top-level design provides a structure that can safely contain multiple lower-level designs.
- Good design is indispensable on large projects.

design levels


① Software system
② Division into subsystems/packages
③ Division into classes within packages
④ Division into data and routines within classes
⑤ Internal routine design

software architecture

set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both. (Clements et al., 2010)

fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution. (ANSI/IEEE Std 1471-2000)

software architecture

- The static structures of a system define its internal design-time elements and their arrangement.
- The dynamic structures of a system define its run-time elements and their interactions.
- The fundamental properties of a system manifest in two different ways:
  - The **externally-visible behaviour** defines the functional interactions between the system and its environment
  - A **quality property** is a non-functional property.

functional requirements and software architecture

- Architecture has emerged as a crucial part of the development process.
- A software architecture is the realization of early design decisions to decompose the system into parts.
- If two architects are given the same requirements, in general, they produce different architectures.
- Functional requirements do not determine architecture.

what affects architecture

- Architectures are influenced by:
  - the stakeholders
  - the developing organization
  - the background and experience of the architects
  - the technical environment
- An architecture affects the factors that influence it.
- A software architecture is the result of technical, business and social influences.
- Software architecture is about the design of the system and the impact it has on its qualities.
- The architecture acts as the skeleton of the system, constrains it, and affects its quality attributes.

software architecture

- Applying software architecture ideas requires a conscious shift to embrace its abstractions.
- If you do not consciously choose the architecture of your systems, then it end up being a big ball of mud.
- A big ball of mud ("spaghetti code") is a software system that lacks a perceivable architecture.
- Some teams design up-front and other teams design as they build.
- Every system has an architecture, whether or not it is documented and understood.

risks

- Each project faces risks, so there is no single correct way to do software architecture.
- One must evaluate the risks on each project.
- Often there is no architecture work, since there is no risk when a proven architecture is reused.
- The harder the problem is, the more one needs to pay attention to the architecture choices.
- The architecture choices are most important when:
  - The solution space is small
  - The risk of failure is high
  - There are difficult quality attribute requirements
  - A new domain is being considered
- A software's design consists of the decisions and intentions that are in the heads of the developers.

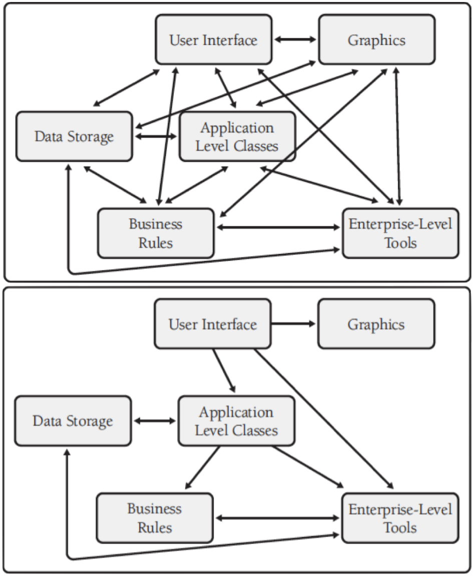
architecture design and detailed design

- The design phase can be partitioned into two parts:
  - Architecture design is the macroscopic parts of the design, such as modules and how they are connected.
  - Detailed design covers everything else.
- The output of design is:
  - a set of artefacts that record the decisions that were taken
  - the rationale for each non-trivial decision is explained
- Every system comprises elements and relations among them.

dependencies

- Each component plays a specific role in the system.
- The components collaborate to provide the required functionality.
- Minimizing dependencies between components is important to create a loosely coupled architecture.
- By eliminating unnecessary dependencies, changes are localized and don't propagate through the system.
- It is necessary to decide about how the components communicate data and control information.

coupling

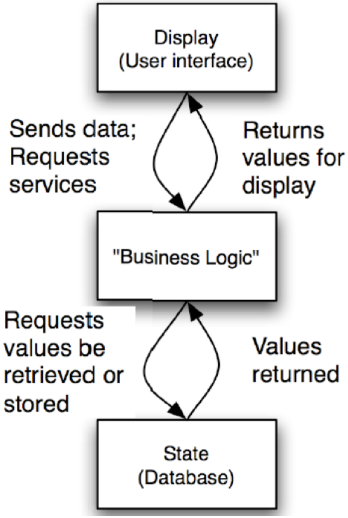

User Interface
Graphics
Data Storage
Application Level Classes
Business Rules
Enterprise-Level Tools
User Interface
Graphics
Data Storage
Application Level Classes
Business Rules
Enterprise-Level Tools

contents

skeletons

skeletons

- A 3-tier software architecture enables IT systems to localize changes and handle transactional loads.
- A cooperating-processes architecture is well suited to operating systems because it isolates faults.
- It is probable that a distributed VOIP network (Skype) uses a peer-to-peer architecture.


Display
(User interface)
Sends data; Requests services
Returns values for display
"Business Logic"
Requests values be retrieved or stored
Values returned
State
(Database)

quality attributes

- Developers must pay attention to functionality, but a system has also quality attributes.
- A system's architecture enables or inhibits qualities such as security or performance.
- Functional requirements that change are a challenge, but evolving quality attributes can force drastic changes.
- Example: a system to support 100 users may be impossible to scale up to 100,000 without a drastic architectural change.

architecture is orthogonal to functionality

- Architecture and functionality can be balanced.
- Architecture is mostly orthogonal to the system's functionality.
- There is no single best architecture, but architectures are more suited to some tasks than others.
- A system's architecture can be changed, but yet keeping its functionality.
- The same architecture can be used on a system with different functionality.
- A poor architecture choice can make functionality and quality attributes difficult to achieve.

quality attributes

- Architecture is likely to require more attention in systems with large scale or high complexity.
- When the system is simple, its architecture is unlikely to sink the project, so one pays little attention to it.
- There is no such thing as an inherently good or bad architecture.
- An architecture is more or less fit for some purpose.
- Architectures can be evaluated but only in the context of specific stated goals.
- There are, however, good rules of thumb.

what makes a good architecture?

- The architecture should be the product of a small group of architects with an identified leader (avoid "design by committee" anti-pattern).
- The architect should base the architecture on a prioritized list of quality attributes.
- The architecture should be documented using views.
- The views should address the concerns of the most important stakeholders.
- The architecture should be evaluated for its ability to deliver the system's important quality attributes.
- The architecture should lend itself to incremental implementation.

presumptive architectures

- A presumptive architecture is a family of architectures that dominates a particular domain.
- Developers in that domain may have to justify a choice that differs from the presumptive architecture.
- Presumptive architectures are similar to reference architectures.
- A reference architecture is a family of architectures that describes an architectural solution to a problem.
- Presumptive architectures succeed because they are a good match for the common risks in the domain.
- IT developers who use the presumptive N-tier architecture will almost always do fine.

Summary

- There is no single correct way to do software architecture.
- Architecture is mostly orthogonal to the system's functionality.
- There is no single best architecture.
- There is no such thing as an inherently good or bad architecture.
- The architecture should be the product of a small group of architects with an identified leader.

bibliography

- Fairbanks G; *Just-enough software architecture: A risk-driven approach*, Marshall & Brainerd, 2010. [chapters 1-2]