Princípy tvorby softvéru
Architektúra softvérového systému

Robert Lukoťka
lukotka@dcs.fmph.uniba.sk
www.dcs.fmph.uniba.sk/~lukotka

M-255

Čo je to architektúra?

"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."

Philippe Kruchten, Grady Booch, Kurt Bittner, and Rich Reitman

Čo je to architektúra?

"The highest-level breakdown of a system into its parts; the decisions that are hard to change; there are multiple architectures in a system; what is architecturally significant can change over a system's lifetime; and, in the end, architecture boils down to whatever the important stuff is."

Martin Fowler

Čo je to architektúra?

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. Architecture is concerned with the public side of interfaces; private details of elements—details having to do solely with internal implementation—are not architectural."

Len Bass, Paul Clements, Rick Kazman

Čo je teda súčasťou architektúry systému?

- Fyzická a logická štruktúra aplikácie.
  - Logické delenie nodov (node = cca. zariadenie na ktorom beží SW).
  - Štruktúra nodov.
  - Logická štruktúra nodu (napr. komponenty, layers).
- Podmienky nasadenia, zvolené technológie (resp. typ technológie), paradigmy.
- Použité komunikačné kanály a protokoly.
- Interfacy ktorými časti aplikácie medzi sebou komunikujú.
- Určenie kľúčových častí systému.
- Kontrola nad behom programu
- Monitorovanie systémy, zálohovanie, . . .
- . . .

SW architektúra

Architektúra by mala:

- Expose the structure of the system but hide the implementation details.
- Realize all of the use cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.

Príklad: Veľká webová aplikácia

Logické delenie nodov: 3-úrovne (3-Tier architektúra):

- Prezentačná
- Business logic
- Dátová

Príklad: Veľká webová aplikácia - Prezentačná vrstva

- Štruktúra nodov vo vrstve: nezávislé, komunikujúce s logickou vrstvou (ktorá z pohľadu prezentačnej vrstvy vyzerá ako jeden node).
- Podmienky nasadenia, zvolené technológie: Browser, Javascript, . . .
- Použité komunikačné kanály: internet
- Interfacy ktorými časti aplikácie medzi sebou komunikujú: HTTP, stateless protokol, + konkrétne detajly
- Určenie kľúčových častí systému: . . .

Príklad: Veľká webová aplikácia - Logická vrstva

- Štruktúra nodov vo vrstve: Load balancer, nezávislé servery komunikujúce s datovou vrstvou ktorá z pohľadu logickej vrstvy vyzerá ako jeden node.
- Podmienky nasadenia, zvolené technológie: Cloud services, Java
- Použité komunikačné kanály a protokoly: internet, MySQL connection
- Interfacy ktorými časti aplikácie medzi sebou komunikujú: SQL
- Určenie kľúčových častí systému . . .

Príklad: Veľká webová aplikácia - Dátová

- Štruktúra nodov vo vrstve: Load balancer pre čítanie, single entry node pre zápis, replikujúce sa nody pre čítanie, ...
- Podmienky nasadenia, zvolené technológie: . . .
- . . .

Vzhľadom na veľkosť systému bude potrebné vybrať aj architektúru jednotlivých podsystémov.

Menšia monolythic aplikácia

- Fyzická a logická štruktúra aplikácie: single node, dve vrstvy - user interface, business logic; business logic sa sklada s komponentov.
- Podmienky nasadenia, zvolené technológie: Windows 8 +, Ubuntu 16.04 +, Python, PyGTK
- User interface: text user interface, bez inversion of control
- Interfacy ktorými časti aplikácie medzi sebou komunikujú . . .

Príklad z OOAM.

Interface
Interface

je kontrakt medzi komunikujúcimi stranami:

- Volajúca strana (vyžaduje/requires interface) má adresu volaného (web / memory) a dohodnutým protokolom (HTTP, C calling convention) volá funkciu interfacu.
- Volaná strana (poskytujúca/implements interface) vykoná nejakú akciu a prípadne dohodnutým protokolom vráti návratovú hodnotu.
-

Komunikácia medzi rozličnými nodami/komponentami má prebiehať iba za použitia striktne zvolených interfaceov.

- Stateless interface nevyžaduje poznať stav druhej strany.
- Inak súčasťou interfacu aj usage protocol (prechody medzi stavmi a v ktorom stave môže byť ktorá funkcia volaná)

SW architektúra - ešte raz

Architektúra by mala:

- Expose the structure of the system but hide the implementation details.
- Realize all of the use cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.

Basic questions

- How will the users be using the application?
- How will the application be deployed into production and managed?
- What are the quality attribute requirements for the application, such as security, performance, concurrency, internationalization, and configuration?
- How can the application be designed to be flexible and maintainable over time?
- What are the architectural trends that might impact your application now or after it has been deployed?

More questions to consider

- What are the foundational parts of the architecture that represent the greatest risk if you get them wrong?
- What are the parts of the architecture that are most likely to change, or whose design you can delay until later with little impact?
- What are your key assumptions, and how will you test them?
- What conditions may require you to refactor the design?

Software architecture principles

- Build to change instead of building to last.
- Model to analyze and reduce risk.
- Use models and visualizations as a communication and collaboration tool.
- Identify key engineering decisions.

Software architecture principles

- Do not try to get it all right the first time
- Design just as much as you can in order to start testing the design.
- Iteratively add details to the design over multiple passes to make sure that you get the big decisions right first, and then focus on the details.

Testing the architecture

- What assumptions have I made in this architecture?
- What explicit or implied requirements is this architecture meeting?
- What are the key risks with this architectural approach?
- What countermeasures are in place to mitigate key risks?
- In what ways is this architecture an improvement over the baseline or the last candidate architecture?

How to find the right architecture

- Determine the Application Type
- Determine the Deployment Strategy
- Determine the Appropriate Technologies
- Determine the Quality Attributes
- Determine the Crosscutting Concerns (authentication, logging, caching, . . .)

You cannot focus on everything.

Key scenario.

- It represents an issue—a significant unknown area or an area of significant risk.
- It refers to an architecturally significant use case.
  - Business Critical.
  - High Impact.
- It represents the intersection of quality attributes with functionality.
- It represents a trade-off between quality attributes.

How to find the right architecture

Break stuff into smaller stuff

- Keep in mind ordinary design principles
  - separation of concerns, single responsibility, principle of least knowledge, DRY, . . .
- Do not mix different stuff into layers/components/. . . (design patterns, component types, data types).
- Be explicit about how structural parts of the design communicate with each other, use abstraction.
- Define a clear contract for the parts.

Architektonické štýly a patterny

- Provide abstract framework for a family of systems
- Help communication

Typy of architektonických štýlov

Je v tom riadny bordel. Architektúra pokrýva veľa rôznych concernov a pohľadov.

- Deployment
- Structure
- Communication
- Domain
- ...

Key architectural styles and patterns

- Client/Server
  Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.
- Peer-to-peer
- Component-Based Architecture
  Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.
- Domain Driven Design
  An object-oriented architectural style focused on modeling a business domain and defining business objects based on entities within the business domain.

Key architectural styles and patterns

- Layered Architecture
  Partitions the concerns of the application into stacked groups (layers).
- N-Tier / 3-Tier
  Segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer.

Key architectural styles and patterns

- Service-Oriented Architecture (SOA)
  Refers to applications that expose and consume functionality as a service using contracts and messages.
- Message Bus
  An architecture style that prescribes use of a software system that can receive and send messages using one or more communication channels, so that applications can interact without needing to know specific details about each other.
- Microservices
- Model-View-Controller
- Object-Oriented

Key architectural styles

Benefits of respective styles

User interface and architecture

Representing and Communicating Your Architecture Design

Cover various views:

- Use case view
- Static view
- Dynamic view
- Physical view
- Development view

Ďalšie

- Oddeliť user interface ako vlastný komponent úplne dokonale?
- Kto má pod kontrolou vykonávanie programu? Inversion of control

Resources

Microsoft Application Architecture Guide