Software System Architectures (NSWI130)
C4 model and diagrams

- Martin Nečaský, Ph.D.
- Department of Software Engineering
- Faculty of Mathematics and Physics
- Charles University in Prague

Structurizr DSL

- a DSL (Domain Specific Language) for documenting software architecture based on the C4 model
- Try, install locally (see previous lecture), or sign up to the Structurizr cloud service
- GitHub repository with reference documentation, cookbook, etc.
- Code from this lecture is available in NSWI130 repository

```
workspace [name] [description] {
  …
}
```

optional

```
workspace extends <file|url> {
  …
}
```

required

- workspace is top level construct which contains a software architecture model and architectural views

```
workspace {
  model {
    …
  }

  views {
    …
  }
}
```

- Workspace must contain a model which defines architectural elements and relationships
- Workspace can also contain architectural views showing architectural elements and relationships

```
model {
  <id> = softwareSystem <name> [description] [tags]
}
```

- Defines a software system.
- description: a short description of the software system focused on its responsibilities
- tags: adds one or more tags (separated by ‘,’) to an element (useful, e.g., for styling)

```
model {
  <id> = person <name> [description] [tags]
}
```

- Defines a person (user, actor, role, …).

```
model {
  <id> -> <id> [description] [technology] [tags]
}
```

- Defines a uni-directional relationship between two elements.
- technology: a technology the relationship will be implemented on

```
views {
  systemContext <software system identifier> [key] [description] {
    include *
  }
}
```

- Defines a system context view (diagram) for the specified software system.
- include: specifies elements and relationships in the view
  - Wildcard identifier * operates differently for different views
  - In this case it specifies the software system + all directly connected people and software systems

sampleworkspace01.dsl

```
views {
  theme <default|url>
}
```

- Reuse of predefined visual stylesheet for presenting views as architectural diagrams
- default: a keyword for the default stylesheet
- url: web address of an external stylesheet

```
views {
  styles {
    element|relationship <tag> {
      … visual properties …
    }
  }
}
```

- definition of own styles using various visual properties

sampleworkspace02.dsl

```
softwareSystem {
  <id> = container <name> [description] [technology] [tags]
}
```

- Defines a container within a software system.

```
views {
  container <software system identifier> [key] [description] {
    include *
  }
}
```

- Defines a container view (diagram) for the specified software system.
- include *
  - All containers within the software system + all people and software systems directly connected to those containers

sampleworkspace03.dsl

```
container {
  <id> = component <name> [description] [technology] [tags]
}
```

- Defines a component within a container.

```
views {
  component <container identifier> [key] [description] {
    include *
  }
}
```

- Defines a component view (diagram) for the specified container.
- include *
  - All components within the container + all people, software systems and containers directly connected to those components

sampleworkspace04.dsl

```
group <name> {
  …
}
```

- Defines a named grouping of elements which will be rendered as a boundary around those elements.

sampleworkspace05.dsl

```
deploymentEnvironment <name> {
  …
}
```

- Defines a deployment architecture of the static elements (software systems and containers).
- Shows how the static elements are deployed at runtime as instances

```
deploymentEnvironment <name> {
  deploymentNode <name> [description] [technology] [tags] [#] {
    …
  }
}
```

- Defines a single deployment node which comprises other deployment nodes or instances of software systems or containers
- # - number of instances (integer)

```
deploymentEnvironment <name> {
  deploymentNode <name> [description] [technology] [tags] [#] {
    softwareSystemInstance <id>
  }
}
```

- Defines an instance of a software system

```
deploymentEnvironment <name> {
  deploymentNode <name> [description] [technology] [tags] [#] {
    containerInstance <id>
  }
}
```

- Defines an instance of a container

```
views {
  deployment <software system id> <environment> {
    include *
  }
}
```

- Defines a deployment view (diagram) for the specified software system.
- include *
  - All deployment nodes and instances within the deployment environment

sampleworkspace06.dsl

```
deploymentEnvironment {
  deploymentNode {
    infrastructureNode <name> [description] [technology] [tags]
  }
}
```

- defines an infrastructure node, which is typically something like a load balancer, firewall, DNS service, etc.

sampleworkspace07.dsl

```
dynamic * [key] [description] {
  …
}
```

- defines dynamic view (diagram)
- useful to show how elements in your static model collaborate at runtime to implement a user story, use case, feature, etc.

```
dynamic * [key] [description] {…}
dynamic <software system id> [key] [description] {…}
dynamic <container id> [key] [description] {…}
```

- *: all people and software systems
- software system id: people and other software systems connected to the system + its containers
- container id: people, software systems and other containers connected to the container + its components

```
dynamic {
  <id> -> <id> [description] [technology]
}
```

- defines an instance of a relationship from the model
- there can be multiple instances of the same relationship
- the order defines the order in the diagram

sampleworkspace08.dsl

```
model {
  softwareSystem {
    !docs docs
  }
}
```

- Defines a written documentation for a software system
- Documentation written in Markdown

sampleworkspace09.dsl
&&
docs directory