# Objectives



- Reminders
- Software Architecture

# Notes



- Representatives from Facebook will be visiting on the October 8-9
  - Give a tech talk in class that Wednesday
- First Sprint Goal and Backlog due today
- First sprint will begins
  - **Sprint 1** October 7 – October 18
  - **Sprint 2** October 21 – Novemeber 1
  - **Sprint 3** November 4 – November 22 (last day of class)
- We deliver working software at the end of each sprint
  - Projects will be evaluated in labs on 18/10, 1/11, and 22/11
- **Quiz 2** November 13

# Team Task



- Upload your sprint goal and sprint backlog to the moodle by tonight
- Designate a Scrum Master
- Take one of the user stories from the project description (this is your sprint goal)
- Write up a use case/s that could be a component of the selected user story
- Form a sprint backlog by breaking up the goal into smaller tasks (things that you could conceivably finish in a day)
- Note: Your sprint goal should include at least the following
  - That your application runs on a container
  - Frontend and Backend component
  - Demonstrates that you can successful connect to a database and send and receive data

# Callbacks



```
function mySequence() {
    $.get("myservlet",{},function (r) {
        show(r);
        $.get("myservlet?chain=" + r.expected, {}, function (r) {
            show(r);
            $.get("myservlet?chain=" + r.expected, {}, function (r) {
                show(r);
                $.get("myservlet?chain=" + r.expected, {}, function (r) {
                    show(r);
                    $.get("myservlet?chain=" + r.expected, {}, function (r) {
                        console.log("Number of steps advanced is " + r.steps);
                        show(r);
                    }, "json");
                }, "json");
            }, "json");
        }, "json");
    }, "json");
}
```

- This function shows how a sequence of requests can be implemented with callbacks and nested functions
- The goal of the exercise is to make the same sequence of requests but with Javascript promises instead of callbacks

# Solving with Promises



```
function mySequenceWithPromises() {
    var jqp = $.get("myservlet", showSuccess);

    jqp.then(function (value) {
        show(value);
        return $.get("myservlet?chain=" + value.expected, showSuccess);
    }).then(function (value) {
        show(value);
        return $.get("myservlet?chain=" + value.expected, showSuccess);
    }).then(function (value) {
        show(value);
        return $.get("myservlet?chain=" + value.expected, showSuccess);
    }).then(function (value) {
        show(value);
        return $.get("myservlet?chain=" + value.expected, showSuccess);
    }).then(function (value) {
        show(value);
    });
}
```

- The function rewritten using Promises
  - Recognize that get returns a promise
  - Chain promises together by calling `then` and returning

# The "4 + 1" View of Architecture[1]



- In 1994, Phillipe B. Kruchten published a very influential article on software architecture (citation below)
- This article became the basis for a popular agile process called the *Rational Unified Process*
- Proposed 5 different "Views" a team could use to gain insight into the operation of the system

[1] Philippe B Kruchten. "The 4+ 1 view model of architecture". In: *IEEE software* 12.6 (1995), pp. 42–50.

# Architecture vs. Detailed Design[2]



- *Requirements* tell us what a system should and should not do
- *Design* tells us how a system is implemented
  - What are the components?
  - How do they work individually?
  - How do they interact with each other?
- Divide the design into two phases: Architectural phase and a Detailed design phase
  - *Architecture*: High-level macro view of the system
  - *Detailed design*: Components described in finer detail, each functional requirement should be addressed by at least one component

[2] Frank Tsui, Orlando Karam, and Barbara Bernal. *Essentials of software engineering*. Jones & Bartlett Learning, 2016.

# Architecture in Software Processes[3]



- Architecture can be a link between requirements and design
- All systems have an architecture, even if that architecture has not been explicitly documented somewhere
- In a traditional process such as *waterfall* the design should be represented in a very detailed manner
  - Role of the developer is to basically *translate* the design into code
- In Agile processes, the developer may have greater responsibility for implementing the detailed design

[3] Frank Tsui, Orlando Karam, and Barbara Bernal. *Essentials of software engineering*. Jones & Bartlett Learning, 2016.

# Software Architecture



- The architecture of a system shows how it is structured, at a high-level of abstraction
  - A system may have more than one way in which it is structured
- Architecture focuses on external properties of modules (e.g. packages) rather than the internal operation of those modules
- Typically, a system is made up of a number of sub-systems. The architecture is the framework of control and communication between the sub-systems
- *Architecture in the small*: how an individual program is broken up into components
- *Architecture in the large*: enterprise systems that may consist of many programs in distributed locations

# Common Architectural Considerations[4]

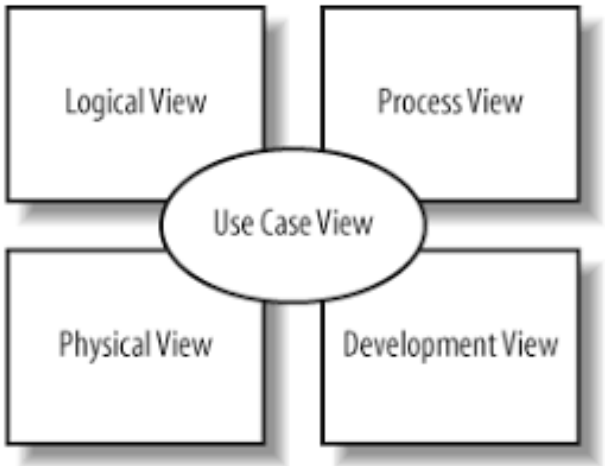
NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

- Is there a generic or pre-existing application architecture to follow?
- Is the system distributed?
- How will the architecture be documented?
- How will the system be decomposed into sub-systems and modules?
  - For example, main program and sub-programs
- *Characteristics*
  - Performance
  - Security
  - Safety
  - Availability
  - Maintainability

[4] Ian Sommerville. "Software Engineering. International computer science series". In: *ed: Addison Wesley* (2004).

# The "4 + 1" View of Architecture[5]






- Kruchten proposed four *views* of a system architecture plus an additional view to unify the others
  - *Logical View*
  - *Process View*
  - *Development View*
  - *Physical View*
- The +1 view is the *scenario view* which is equivalent to a description of the system in terms of use cases

[5] Philippe B Kruchten. "The 4+ 1 view model of architecture". In: *IEEE software* 12.6 (1995), pp. 42–50.

# Logical View



- Supports the functional requirements
- Shows the components of a system and their relationships
- Set of abstractions
- In Object-Oriented Design we can represent this view by UML class and sequence diagrams

# Process View

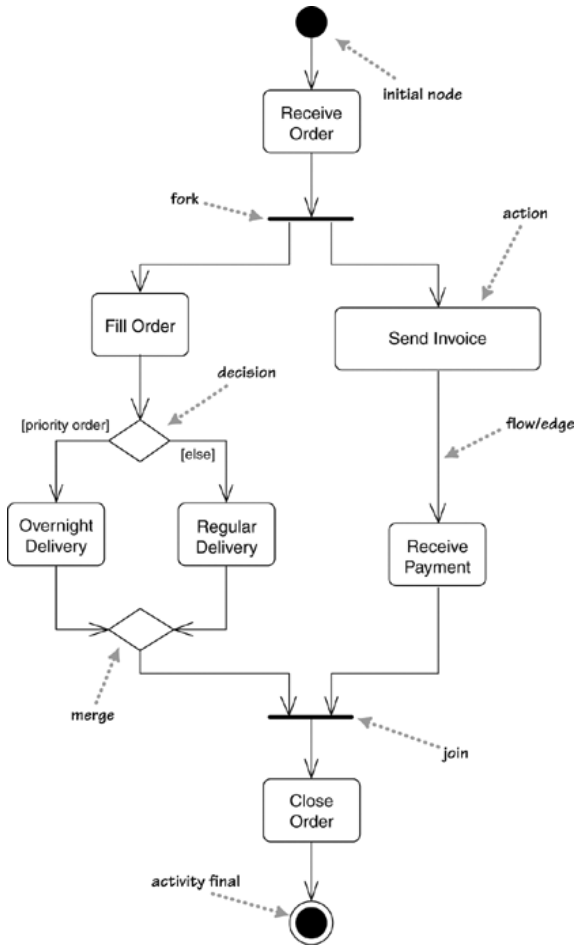


- Describes the processes of a system
  - In our case, we can think of these as the various threads of execution
- "Logical network of communicating programs"[6]
- In UML this view can be represented by an activity diagram

[6] Philippe B Kruchten. "The 4+ 1 view model of architecture". In: *IEEE software* 12.6 (1995), pp. 42–50.

# Activity Diagram[7]

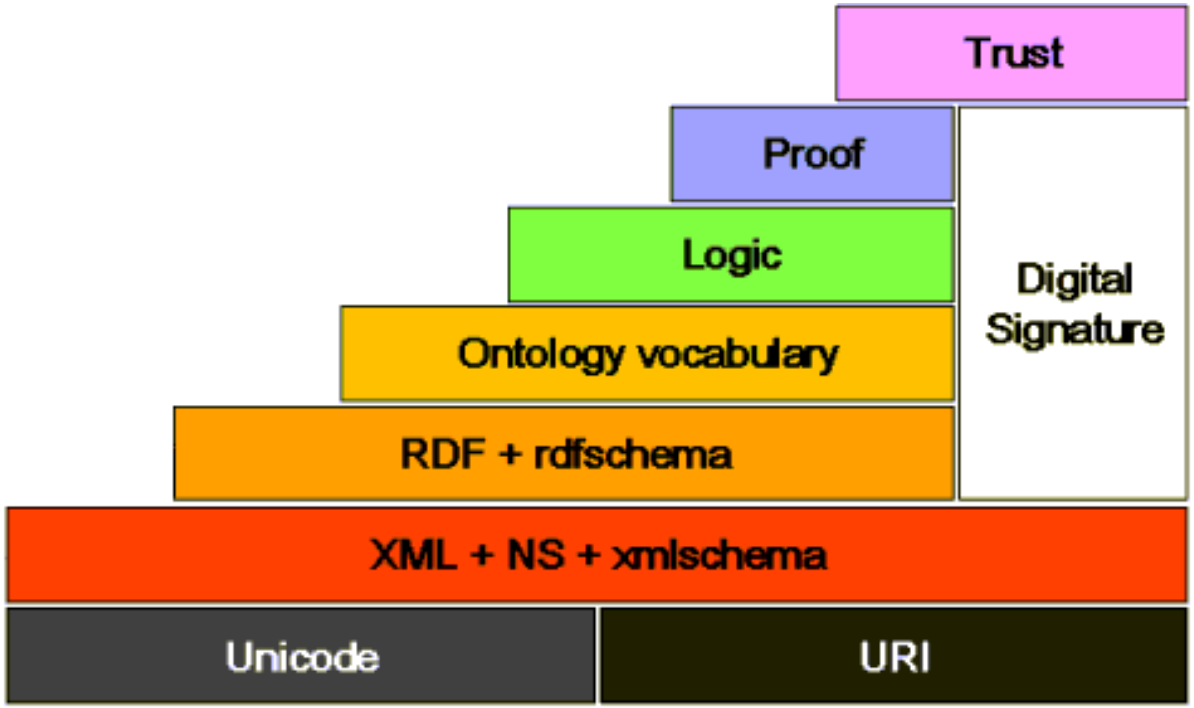





- Like a flowchart, can also describe concurrent processes, see Ch. 28 of Larman

[7] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

# Development View




- Organization of software modules (e.g. packages, libraries)
  - Which modules will be imported where?
  - How can work be divided?
- Kruchten recommends a *layered style* for the development view (example pictured, layers of the semantic web[8])

[8] image: w3.org

# Physical View



- Execution environment of the system
- Mapping the software to hardware resources
- In UML, this view can be communicated with a *deployment diagram*

# Scenario or Use Case View



- Shows the functionality of the system from an external point of view, or the point of view of the user
- Goals and scenarios
- Helps to explain the structures found in the other views
- Represent this with textual use-cases as well as UML use case diagrams
- **Task**: Read the article listed below[9]

[9] Philippe B Kruchten. "The 4+ 1 view model of architecture". In: *IEEE software* 12.6 (1995), pp. 42–50.

# Architectural Styles



- Pipes-and-filters
- Client-server
- Based on a common type of application
- Layered
- Event-driven
- MV*
- Repository/Database Centric

# Pipes-and Filters, Standard I/O



- A type of *Data-Flow Architecture*
  - Functional transformation of data, process input and produce output
- Input output scheme designed for interactive user terminals: standard input, standard output and standard error
- redirects and pipes allow you to control the source of input and chain together utilities
- `command < fn` allows you to redirect standard input
- `command > fn` allows you to redirect standard input
  - `date > now`: will create a file named "now" that contains the date
- pipes | allow you to chain together multiple utilities

# Client-Server Architecture[10]

NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

- System where data and data processing are distributed across a number of independent processes/components
- Stand-alone services provide specific functionalities
- Clients access these services (e.g. printing)
- Network infrastructure provides communication between the services and access for the clients

[10] Ian Sommerville. "Software Engineering. International computer science series". In: *ed: Addison Wesley* (2004).

# Layered Architecture[11]



- Models the interfaces of sub-systems
- Organizes system into a set of layers where each layer provides services to higher layers and utilizes the services of lower layers
- Supports incremental development, changes only affect adjacent layers
- May be difficult to find a clean separation in a real world problem

[11] Ian Sommerville. "Software Engineering. International computer science series". In: *ed: Addison Wesley* (2004).

# Application Types[12]

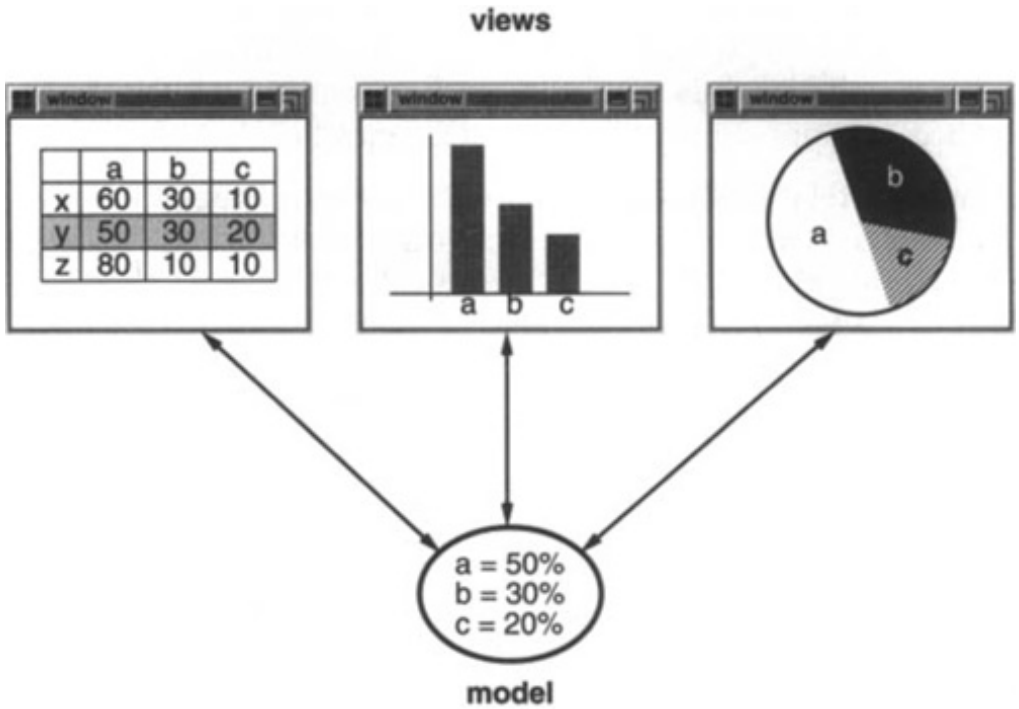
NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

- Data processing applications
  - Batch process data without user interaction
- Transaction processing applications
  - Process user requests asynchronously and update a central database
  - *E-commerce System*
- Event processing systems
  - System designed to respond to events from some wider context
- Language processing systems
  - Instructions are specified in a formal language that is interpreted by the application
  - *Compiler*

[12] Ian Sommerville. "Software Engineering. International computer science series". In: *ed: Addison Wesley* (2004).

# Model View Separation[14]






- Model-View Separation principle[13]: Domain objects should not have direct knowledge of UI objects

[13] Craig Larman. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Prentice Hall PTR, 2012.

[14] image credit: Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995

# Benefits of Model-View Separation[15]



- Better cohesion of model definitions
- Allows parallel development of UI and business logic
- Minimize impact of interface changes
- Easy development of new views
- Attach multiple views to the same domain object
- Separate execution of business logic
- Simple to change UI frameworks

[15] Craig Larman. *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Prentice Hall PTR, 2012.

# MV* in Javascript



- In the Javascript space: there are many implementations of MV* patterns
  - MVVM: Model-View-ViewModel
  - MVP: Model-View-Presenter
- Heavy Frameworks
  - Angular
  - React
- Lightweight Frameworks
  - Backbone (js)
  - Knockout

# Declarative Data-Bindings



- Modern Javascript frameworks usually implement so-called Declarative data-bindings
- Imperative Programming
  - State changes based on a sequence of commands
  - Example: "Clean and chop six onions. Brown *them* in a pan with butter."
- Declarative Programming
  - Describe what a program should do, instead of how it should be done
- Declarative data binding is just a way to link a Javascript variable with a UI element (often using an attribute like `data-bind`)

# Knockout JS



- Lightweight JS implementation of MVVM Pattern
- Based on...
  - Observables and dependency tracking
  - Declarative bindings
  - Templating
- *Model*: Domain layer accessed via AJAX
- *ViewModel*: JS representation of the UI data and operations
- *View*: The actual UI, HTML with declarative bindings that link to the view model

# Simple Example with KO



```
<script type="text/javascript">
var myViewModel;

$(document).ready(function () {
  myViewModel = { greetingName: ko.observable('Mark')};
  ko.applyBindings(myViewModel);
});
</script>
/////
<body>
<h2>KO Example</h2>
<p>
Hello, <b><span data-bind="text: greetingName"></span></b>.
How are you today?
</p>
</body>
```

- Framework sets up the pub/sub mechanism for us, we only need to call `myViewModel.greetingName("new value")` to update the view

# Observable Array



```
function AppViewModel() {
    var self = this;

    self.people = ko.observableArray([
        { name: 'Bert' },
        { name: 'Charles' },
        { name: 'Denise' }
    ]);

    self.addPerson = function() {
        self.people.push({ name: "New at " + new Date() });
    };

    self.removePerson = function() {
        self.people.remove(this);
    }
}

ko.applyBindings(new AppViewModel());
```

- From Knockout Documentation, observable array