Design – Modularity & High Level Design

Thomas Fritz

Many thanks to Reid C. Holmes, Elisa Baniassad and Gail C. Murphy

Agenda

1. Design Introduction
2. Modularity & Abstraction
3. Software Architecture (& Exercise)
4. Diagrams
5. Software Design Exercise (if there is time)

Note:

- *For exams you will have to know UML syntax*
- *Fill in quiz by 10:20am*

Examinable skills

By the end of this class, you should be able to…

- Explain the importance of software design
- Describe what modularity is and why it matters
- Describe the role of software architecture in software design and the design decisions it addresses
- Sketch an architecture of a software system
- Understand the use of diagrams in software development
- Read and create a design for a given system and specify it in correct UML component/deployment/class diagram syntax

Introduction to Design

Learning Objectives

Be able to:

- Describe what software design is
- Explain the importance of software design

What is clothing design?

Why might a designer decide to design such a jacket?

What might have influenced the designer?

Picture from www.arcteryx.com

What is bridge design?

Golden Gate Bridge

Inputs:

Constraints:

What is design?

*What is design? What makes something a design problem? It's where you stand with **a foot in two worlds** – the world of technology and the world of people and human purposes – and you try to bring the two together.*

- Mitchel Kapor, A Software Design Manifesto (1991)

Kapor goes on to say…

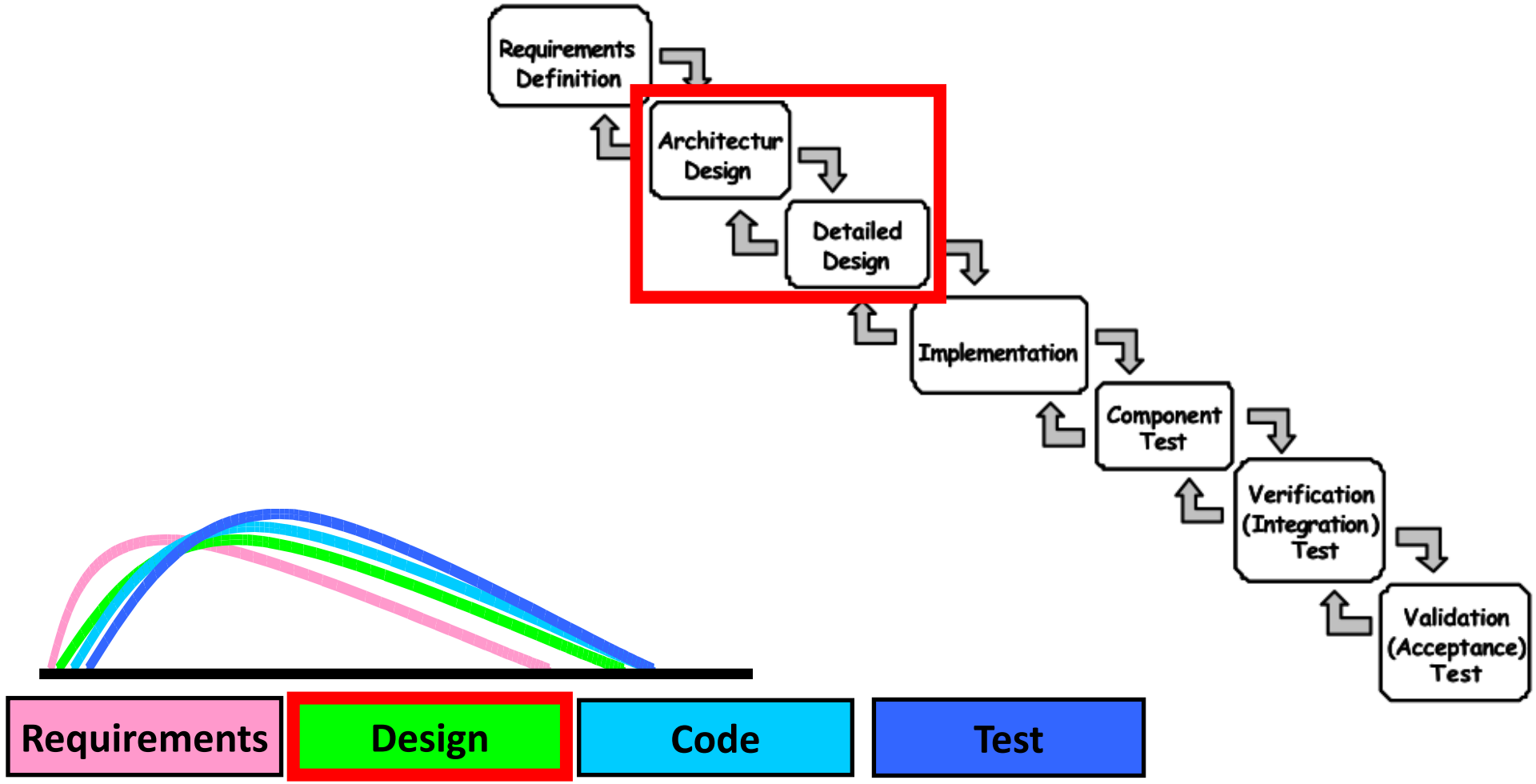
Design disciplines are concerned with making artifacts for human use. Architects work in the medium of buildings, graphic designers work in paper and other print media, industrial designers on mass-produced manufactured goods, and software designers on software. ***The software designer should be the person with overall responsibility for the conception and realization of the program.***

Why design? (why not just code)

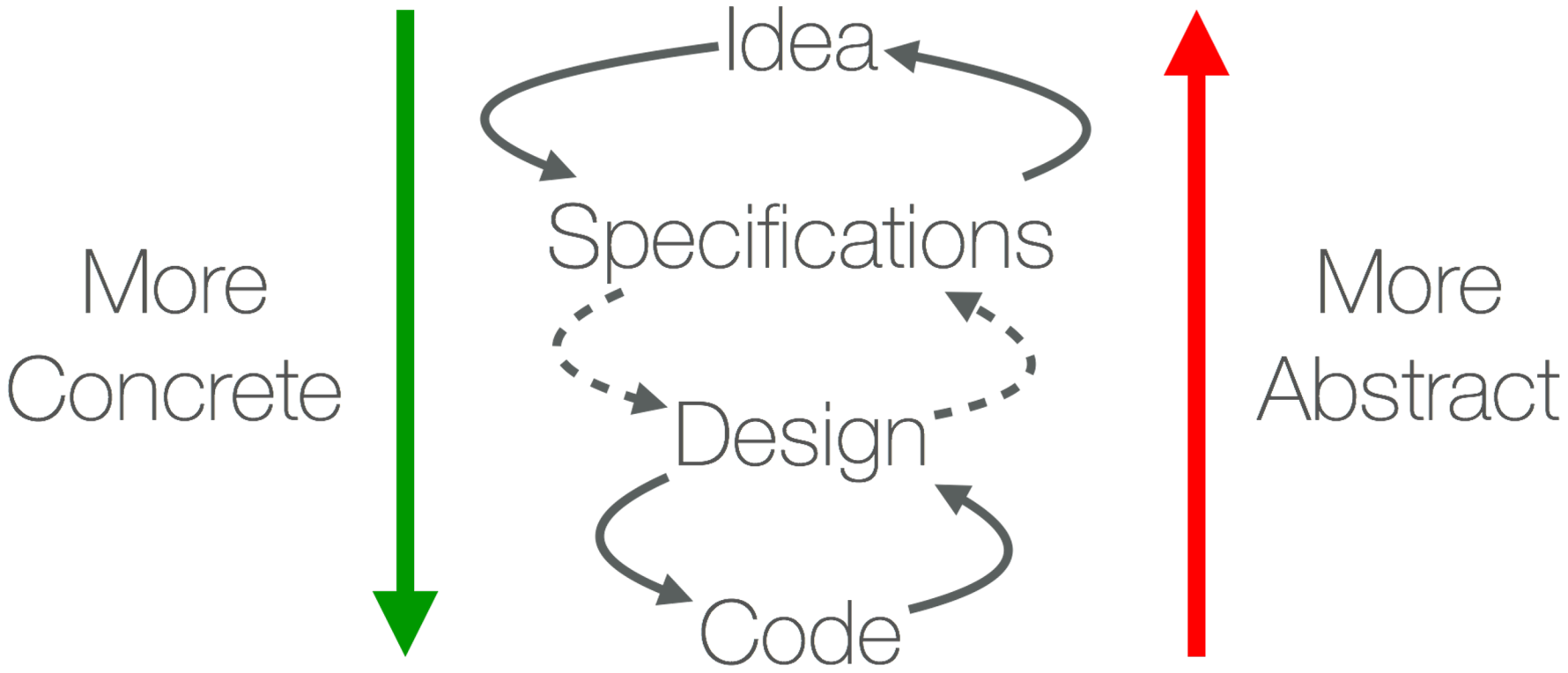
Why design?

- Facilitates communication
- Eases system understanding
- Eases implementation
- Helps discover problems early
- Increases product quality
- Reduces maintenance costs
- Facilitates product upgrade

Design in the process



Design in the process



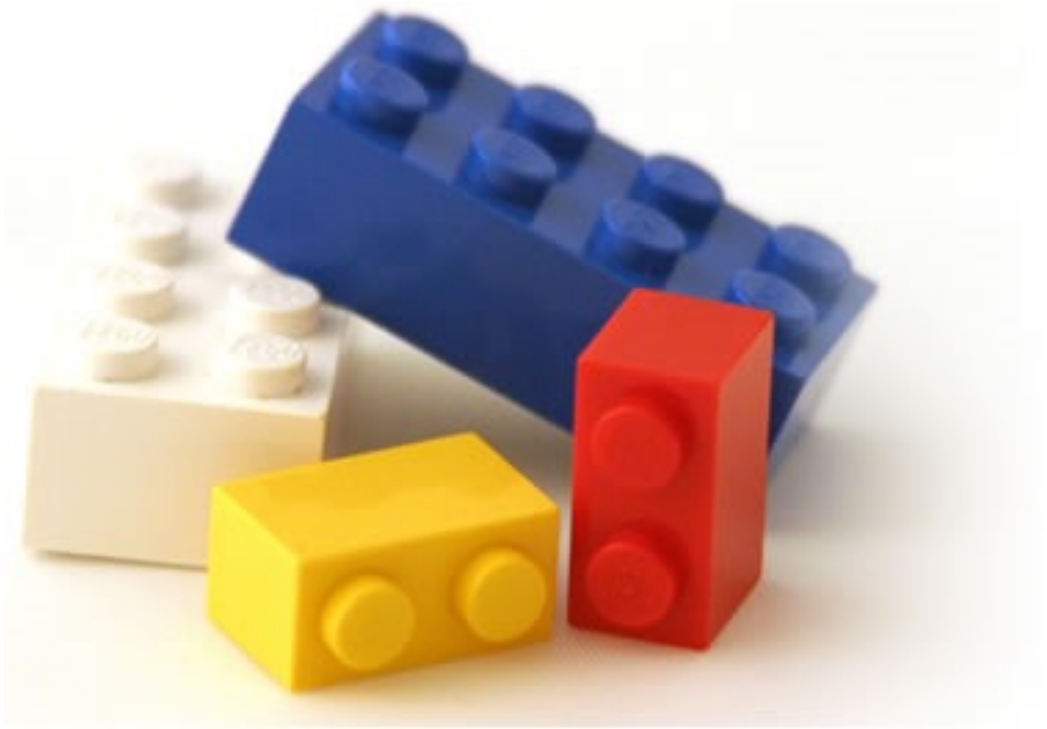
Modularity and Abstraction

Learning Objectives

Be able to:

- Describe what modularity is, why it matters and what it enables
- Describe the role of abstraction, information hiding and interfaces in software development

Software design - Modularity



The goal of all software design techniques is to break a complicated problem into simple pieces.

Modularity

- Involves two ideas
- One idea: interdependence within and independence across modules

> *"A module is a unit whose structural elements are powerfully connected among themselves and relatively weakly connected to elements in other units. Clearly there are degrees of connection, thus there are gradations of modularity"*
>
> - McCelland and Rumelhart 1995

Modularity

- Second idea is captured by the three terms: abstraction, information hiding and interface

  A complex system can be managed by dividing it up into smaller pieces and looking at each one separately. When the complexity of one of the elements crosses a certain threshold, that complexity can be isolated by defining a separate abstraction that has a simple interface. The abstraction hides the complexity of the element: the interface indicates how the element interacts with the larger system"

  - Baldwin and Clarke 2000

Modularity - Abstraction, Information Hiding, Encapsulation and Interfaces

"Abstraction, information hiding, and encapsulation are very different, but highly-related, concepts. One could argue that abstraction is a technique that help us identify which specific information should be visible, and which information should be hidden. Encapsulation is then the technique for packaging the information in such a way as to hide what should be hidden, and make visible what is intended to be visible."

-- Edward V. Berard

Example: a book in an online book store


```
class Book {
        private double fPrice;
        …
        public double getPrice() {…}
        public String getTitle() {…}
        public String getISSN() {…}
        private double calculateBookTax() {…}
}
```

What is the (a) encapsulation, (b) abstraction, and (c) interface in this example?

Photo by cottonbro from Pexels

Why Modularity?

1. Do you have any good examples of modularity in real life?
2. Why is modularity valuable/useful/good?

Why Modularity?

- Minimize complexity
- Reusability
- Extensibility
- Portability
- Maintainability
- …

- **Creates options**

  Decisions that are hidden inside modules and that can be changed create options

Modularity and Option Example

Let’s say Interface A has a function like:

- sendMessage(to, msg)

Do we have any options on implementing the Messaging module?

Modularity in Software

It used to be that when you wrote a program, you wrote most of it from scratch, including the data structures.

1. How do you go about writing a program today?
2. Why? And are there any disadvantages?

Software Supply Chains

- Software we write today is often dependent on a chain of software modules we may or may not even know about
- As a developer, I may make a dependence on a module (e.g. Google Maps), but we don't know what Google Maps depends on

- Sometimes these chains can have unintended consequences…

The 17 lines that “broke the Internet”

http://arstechnica.com/information-technology/2016/03/rage-quit-coder-unpublished-17-lines-of-javascript-and-broke-the-internet/

- March 2016
- Some definitions:
  - npm – default package manager for Node.js (OS cross-platform JS runtime environment for executing JS code server side)
  - Npm has > 2 Mio modules
- Story
  - Programmer Koçulu has project named “kik” on npm with a few modules
  - “kik” app (a company) asks Koçulu to rename the project
  - Koçulu refuses and kik (company) threatens legal action
  - npm Inc. decides to give the name to Kik (the company)
  - Koçulu quits npm and deletes all his modules on npm; including “leftpad”, 17 lines of code that right-justify text
  - Turns out many large projects depend on left pad
  - Code starts to break

Modularity – Summary

- Provides a whole lot of value in software industry
- Involves:
  - Abstraction, information hiding, encapsulation and interfaces
  - Interdependence within the module and independence across modules
- Allows: parallel work on independent parts of the system
- Is essential for enabling development of complex systems
- Fundamental driver for software design

Software Architecture Design – Exercise

Learning Objectives

Be able to:

- Sketch an architecture of a software system
- Describe the kinds of design decisions software architecture addresses

Software architecture

- The set of principal design decisions about the system
- Encompasses:
  - structure,
  - behavior,
  - non-functional properties
- Subset of the overall design of the system
- Helps enable further independent design work

Many thanks to Gail C. Murphy

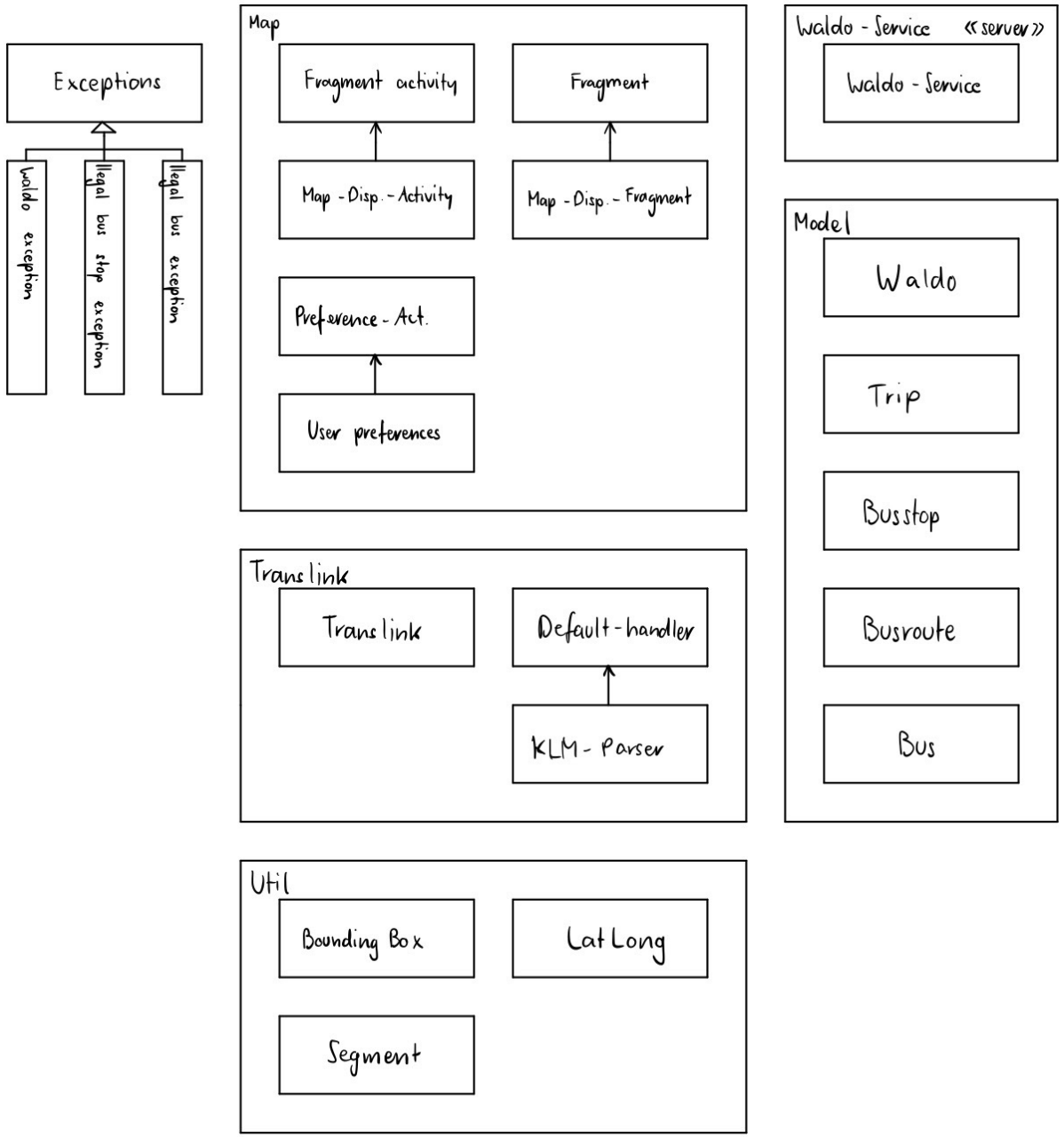
Software Architecture Exercise [https://bit.ly/3Tdco07] (20mins)

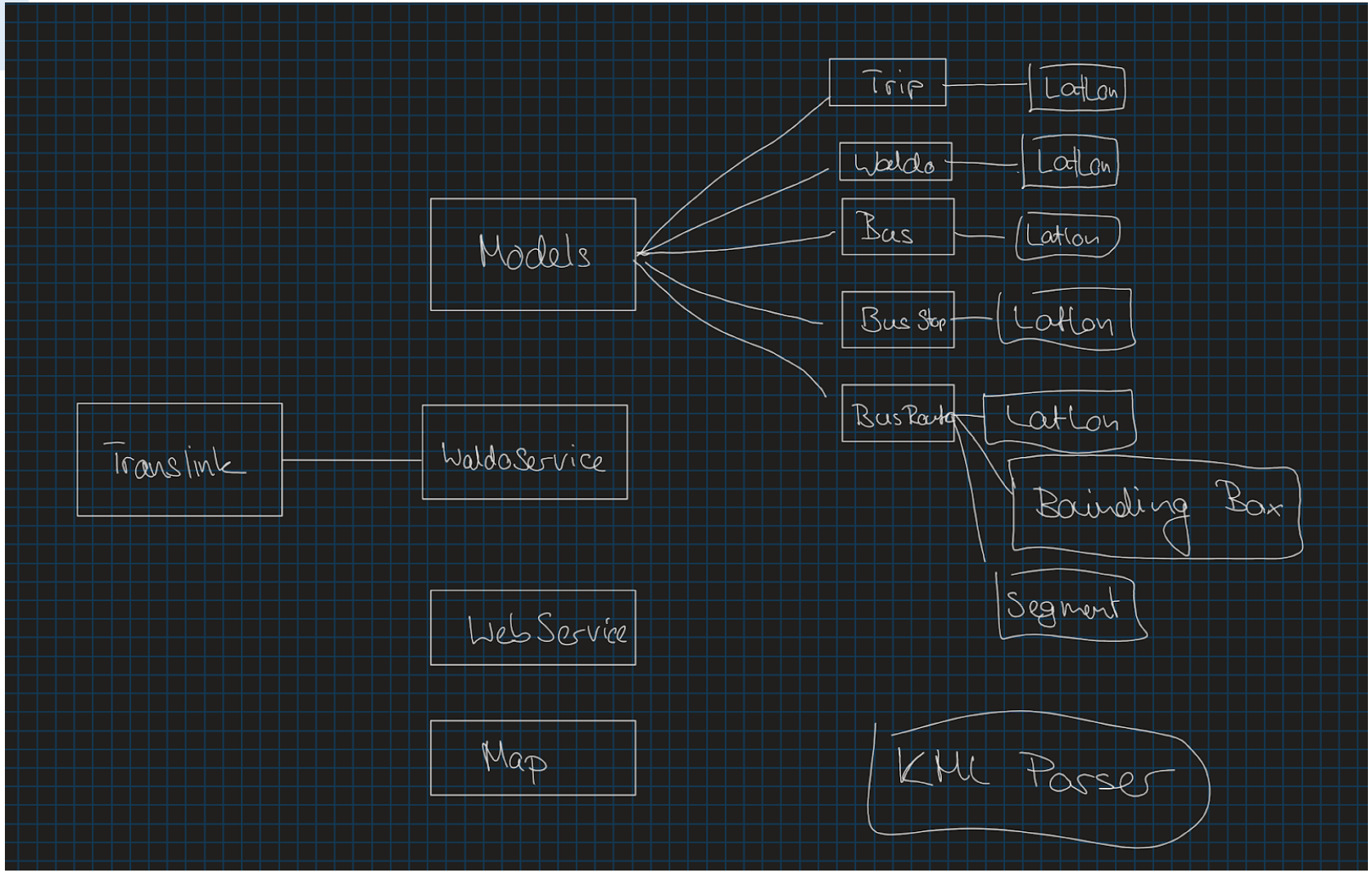
- Each download Java package **Waldo-src.zip** (originally UBC), on OLAT

In a group

- Read/Analyze the code and sketch / draw diagrams to describe the software architecture of the system and *copy & paste it in google document*
- Warning: Groups will be randomly asked to come up and describe their designs
- There is no “right” answer here…

Android application called Waldo. The Waldo applications lets you (1) track other users of the Waldo application within a certain geographic area, (2) plots your location and the locations of other users on a map, and (3) uses live bus information from Translink (Vancouver) to determine the best bus to use to reach a selected user's location from your current location


Exceptions
Waldo exception
Illegal bus stop exception
Illegal bus exception
Map
Fragment activity
Fragment
Map - Disp. - Activity
Map - Disp. - Fragment
Preference - Act.
User preferences
Translink
Translink
Default - handler
KLM - Parser
Util
Bounding Box
LatLong
Segment
Waldo - Service «server»
Waldo - Service
Model
Waldo
Trip
Busstop
Busroute
Bus


Trip
LatLon
Waldo
LatLon
Models
Bus
LatLon
Bus Stop
LatLon
Bus Route
LatLon
Bounding Box
Segment
Translink
WaldoService
Web Service
Map
KML Parser


ca.ubc.cpsc210.waldo.map::MapDisplayFragment
-LOG_TAG = "MapDisplayFragment": String
-ICICS = new GeoPoint(49.261182, -123.2488201): GeoPoint
-CENTERMAP = ICICS: GeoPoint
-sharedPreferences: SharedPreferences
-mapView: MapView
-mapController: MapController
-userLocationOverlay: SimpleLocationOverlay
-busStopToBoardOverlay: ItemizedIconOverlay<OverlayItem>
-busStopToDisembarkOverlay: ItemizedIconOverlay<OverlayItem>
-waldosOverlay: ItemizedIconOverlay<OverlayItem>
-routeOverlays: List<PathOverlay>
-selectedStopOnMap: OverlayItem
-selectedBus: OverlayItem
-locationManager: LocationManager
-locationListener: MyLocationListener
-locationProviderFine: String
-locationProviderCoarse: String
-transLinkService: TransLinkService
-waldoService: WaldoService
-selectedWaldo: Waldo
-userName: String
+onActivityCreated(Bundle savedInstanceState): void
-initializeLocationManager(): void
-initializeWaldo(): void
+onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState): View
-resetOverlays(): void
-clearOverlays(): void
-clearAllOverlaysButWaldo(): void
+onDestroyView(): void
+onDestroy(): void
+onResume(): void
+onPause(): void
+updateMyLocation(Location location): void
+onSaveInstanceState(Bundle outState): void
+findWaldos(): void
+clearWaldos(): void
-createBusStopToBoardOverlay(): ItemizedIconOverlay<OverlayItem>
-createBusStopToDisembarkOverlay(): ItemizedIconOverlay<OverlayItem>
-createWaldosOverlay(): ItemizedIconOverlay<OverlayItem>
-createPathOverlay(): PathOverlay
-createLocationOverlay(): SimpleLocationOverlay
-plotEndPoints(Trip trip): void
-plotRoute(Trip trip): void
-plotWaldos(List<Waldo> waldos): void
-createSimpleDialog(String msg): AlertDialog
ca.ubc.cpsc210.waldo.map::MapDisplayActivity
-fragment: MapDisplayFragment
-sideDrawerMenu: DrawerLayout
-sideDrawerMenuList: ListView
-sideDrawerToggle: ActionBarDrawerToggle
-menuItems = { "Find Waldos", "Clear Waldos", "Settings" }: String[]
-SideDrawerItemAdapter = new BaseAdapter() {
@Override
public int getCount() {
return menuItems.length;
}
@Override
public Object getItem(int position) {
return null;
}
@Override
public long getItemId(int position) {
return 0;
}
@Override
public View getView(int position, View convertView, ViewGroup parent) {
View retval = LayoutInflater.from(parent.getContext()).inflate(R.layout.side_drawer_item, null);
TextView title = (TextView) retval.findViewById(R.id.drawer_item_content);
ImageView icon = (ImageView) retval.findViewById(R.id.drawer_item_icon);
String menuItem = menuItems[position];
title.setText(menuItem);
switch(position) {
case 0:
icon.setImageResource(R.drawable.ic_map);
break;
case 1:
icon.setImageResource(R.drawable.ic_upload);
break;
case 2:
icon.setImageResource(R.drawable.ic_settings);
}
return retval;
}
}: BaseAdapter
+onCreate(Bundle savedInstanceState): void
+onOptionsItemSelected(MenuItem item): boolean
+onCreateOptionsMenu(Menu menu): boolean
+onConfigurationChanged(Configuration newConfig): void
-selectItem(int position): void
ca.ubc.cpsc210.waldo.translink::TranslinkService
-APIKEY = "OR/NsuCHtbEeShvml83": String
-routes: Set<BusRoute>
-stops: Set<BusStop>
+TranslinkService(): ctor
+getBusStopsAround(LatLon location, int radius): Set<BusStop>
-parseBusStopsAroundFromJSON(String input): Set<BusStop>
+getBusEstimatesForStop(BusStop stop): void
-parseBusEstimatesFromJSON(BusStop stop, String input): void
-makeJSONQuery(StringBuilder urlBuilder): String
+parseKML(BusRoute route): void
+addBusStop(BusStop stop): void
+lookupRoute(String number): BusRoute
+getBusStops(): Set<BusStop>
+addToRoutes(Set<BusRoute> routes): void
+translateRoutes(List<String> routesAsParsed): Set<BusRoute>
+clearModel(): void
+findRoutesInCommon(Set<BusStop> stopsNearMe, Set<BusStop> stopsNearWaldo): List<BusRoute>
+selectRouteToTake(String routingType, Set<BusStop> stopsNearMe, Set<BusStop> stopsNearWaldo, List<BusRoute> routesInCommon, String travelDirectionEW, String travelDirectionNS): Trip
-selectTripClosestToDest(Set<BusStop> stopsNearMe, Set<BusStop> stopsNearWaldo, List<BusRoute> routesInCommon, String travelDirectionEW, String travelDirectionNS): Trip
-selectNextBusFromMe(Set<BusStop> stopsNearMe, Set<BusStop> stopsNearWaldo, List<BusRoute> routesInCommon, String travelDirectionEW, String travelDirectionNS): Trip
-findEndStop(Set<BusStop> stopsNearWaldo, BusRoute route): BusStop
-inCorrectDirection(String direction, String travelDirectionEW, String travelDirectionNS): boolean
+checkIfAtSameBusStop(Set<BusStop> stopsNearMe, Set<BusStop> stopsNearWaldo): BusStop
ca.ubc.cpsc210.waldo.model::BusRoute
-number: String
-stops: Set<BusStop>
-buses: Set<Bus>
-routeMapLocation: String
-bounds: BoundingBox
-segments: List
+BusRoute(String number): ctor
+addBus(Bus nextBus): void
+clearBuses(): void
+getBuses(): Set<Bus>
+getRouteNumber(): String
+setRouteMapLocation(String routeMapLocation): void
+getRouteMapLocation(): String
+setBounds(String north, String south, String east, String west): void
+getBounds(): BoundingBox
+addSegment(Segment seg): void
+getSegments(): List
+equals(Object other): // Compare bus routes based on bus number boolean
+hashCode(): int
+toString(): String
ca.ubc.cpsc210.waldo.waldowebservice::WaldoService
-waldos: List<Waldo>
-BROADWAY_GRANVILLE = new LatLon(49.263772, -123.138371): LatLon
-KERRISDALE = new LatLon(49.235, -123.1627401): LatLon
-WALDO_WEB_SERVICE_URL = "http://kramer.nss.cs.ubc.ca:8080/": String
-key: String
-userName: String
+WaldoService(): ctor
+initSession(String nameToUse): String
+getFixedWaldos(): List<Waldo>
+getRandomWaldos(int numberToGenerate): List<Waldo>
+getWaldos(): List<Waldo>
+getMessages(): List<String>
-parseInitializeResult(String s): String
-initError(String s): boolean
-parseMessages(String s): List<String>
-parseRandomWaldos(String s): List<Waldo>
-makeJSONQuery(StringBuilder urlBuilder): String
ca.ubc.cpsc210.waldo.model::BusStop
-number: int
-name: String
-latlon: LatLon
-routes: Set<BusRoute>
-descriptionToDisplay: String
+BusStop(int number, String name, LatLon latlon, Set<BusRoute> routes): ctor
+BusStop(int number): ctor
+getNumber(): int
+getName(): String
+getLatLon(): LatLon
+getRoutes(): Set<BusRoute>
+setDescriptionToDisplay(String description): void
+getDescriptionToDisplay(): String
+equals(Object other): boolean
+hashCode(): int
+toString(): String
-routesAsString(): String
ca.ubc.cpsc210.waldo.translink::KMLParser
-sb: StringBuilder
-name: String
-route: BusRoute
-north: String
-south: String
-east: String
-west: String
+KMLParser(BusRoute route): ctor
+endDocument(): void
+startElement(String uri, String localName, String qName, Attributes attributes): void
+endElement(String uri, String localName, String qName): void
+characters(char[] ch, int start, int length): void
-parseSegment(String coordinates): Segment
ca.ubc.cpsc210.waldo.util::LatLon
-lat: double
-lon: double
-WIGGLE = .01: double
+LatLon(String lat, String lon): ctor
+LatLon(double lat, double lon): ctor
+getLatitude(): double
+getLongitude(): double
+isIllegal(): boolean
+inBetween(LatLon pointOfInterest, LatLon point1, LatLon point2): boolean
+toString(): String
+equals(Object other): boolean
+distanceBetweenTwoLatLon(LatLon point1, LatLon point2): double
ca.ubc.cpsc210.waldo.model::Trip
-startStop: BusStop
-endStop: BusStop
-direction: String
-routeToUse: BusRoute
-walkingDistance: boolean
+Trip(BusStop start, BusStop end, String direction, BusRoute route, boolean walkingDistance): ctor
+inWalkingDistance(): boolean
+getStart(): BusStop
+getEnd(): BusStop
+getDirection(): String
+getRoute(): BusRoute
ca.ubc.cpsc210.waldo.model::Waldo
-name: String
-lastUpdated: Date
-lastLocation: LatLon
+Waldo(String name, Date lastUpdated, LatLon lastLocation): ctor
+getName(): String
+getLastUpdated(): Date
+update(Date date): void
+update(LatLon latlon): void
+getLastLocation(): LatLon
+toString(): String
ca.ubc.cpsc210.waldo.util::BoundingBox
-north: double
-south: double
-east: double
-west: double
+BoundingBox(): ctor
+BoundingBox(double n, double s, double e, double w): ctor
+getNorth(): double
+getSouth(): double
+getEast(): double
+getWest(): double
ca.ubc.cpsc210.waldo.model::Bus
-route: BusRoute
-direction: String
-stop: BusStop
-minutesToDeparture: int
+Bus(BusRoute route, String direction, BusStop stop, int minutesToDeparture): ctor
+getDirection(): String
+getStop(): BusStop
+getMinutesToDeparture(): int
+getRoute(): BusRoute
ca.ubc.cpsc210.waldo.util::Segment
-points: List<LatLon>
+Segment(): ctor
+addPoint(LatLon pt): void
+iterator(): Iterator<LatLon>
ca.ubc.cpsc210.waldo.map::UserPreferences
+onCreate(Bundle savedInstanceState): void
ca.ubc.cpsc210.waldo.exceptions::IllegalBusStopException
+IllegalBusStopException(String msg): ctor
ca.ubc.cpsc210.waldo.exceptions::WaldoException
+WaldoException(String msg): ctor
ca.ubc.cpsc210.waldo.exceptions::IllegalBusException
+IllegalBusException(String msg): ctor

Software Architecture

Learning Objectives

Be able to:

- Describe the role of software architecture in software design
- Define, provide examples of and recognize components and connectors in an architectural design
- Describe the kinds of design decisions software architecture addresses

Software architecture

- The set of principal design decisions about the system
- Encompasses: structure, behavior, non-functional properties
- Subset of the overall design of the system
- Describes the gross structure of the system
  - Main components and their behavior (at architectural level, e.g. client/server, web service, software package)
  - Connections between the components / communication
  - Configurations: bind components and connectors together in a specific way

Components

- Elements that encapsulate processing and data at architectural level
- Definition
  - Architectural Entity that:
    - Encapsulates subset of functionality
    - Restricts access via explicit interface
    - Has explicit environmental dependencies
    - e.g. client/server, web service, software package

Connectors

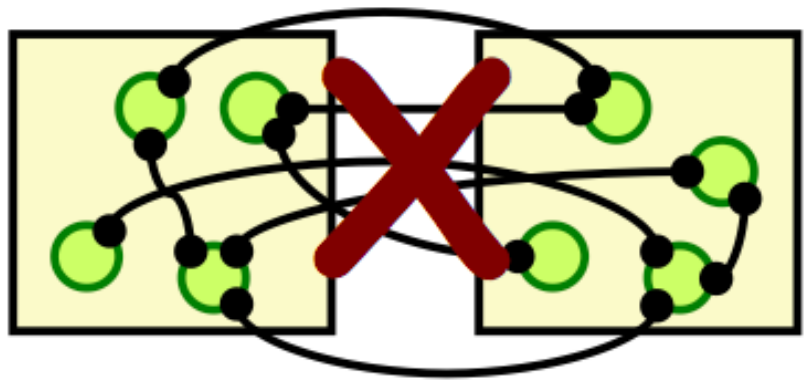
- Definition
  - An architectural entity tasked with effecting and regulating interactions between components
- Connectors are often more challenging than components in large heterogenous systems
- Often consists of method calls, but can be much more
- Frequently provide application-independent interaction mechanisms.

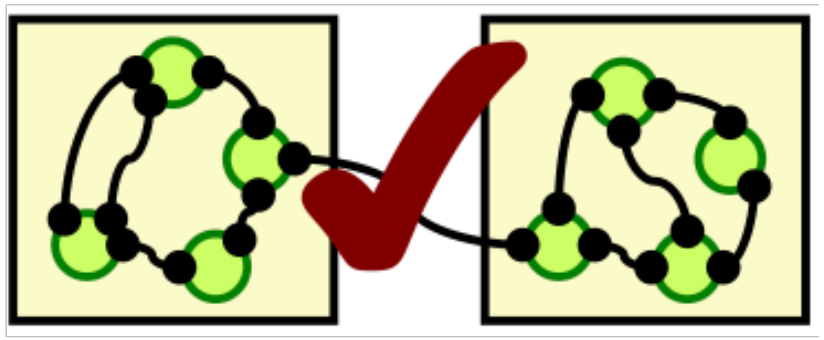
Configurations

- Bind components and connectors together in a specific way
- Definition
  - An architectural configuration, or topology, is a set of specific associations between the components and the connectors of they system's architecture
- Differentiates a bag of components and connectors from an implementable system

Topological goals

- Minimize **coupling** between components
  - The less components know about each other, the better (also known as information hiding)
- Maximize **cohesion** within each component
  - Components should be responsible for a logical service; extraneous functionality should not be present





[Steve Easterbrook: http://www.cs.toronto.edu/~sme/CSC302/notes/04-package-diagrams.pdf]

Diagrams

Learning Objectives

Be able to:

- Read and create basic UML class, component and deployment diagrams

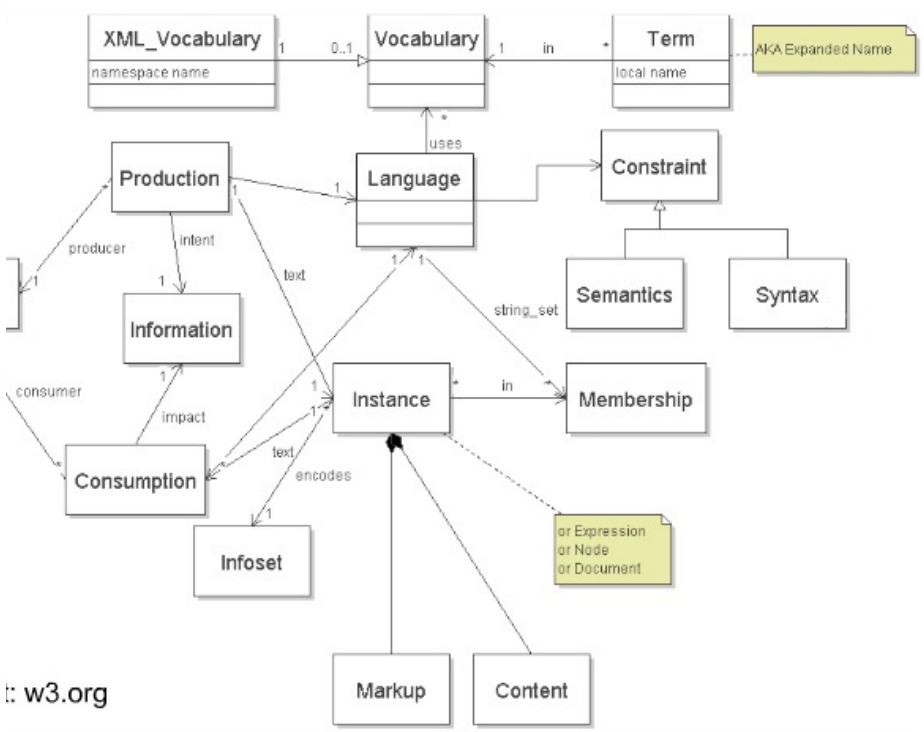
Describing architecture

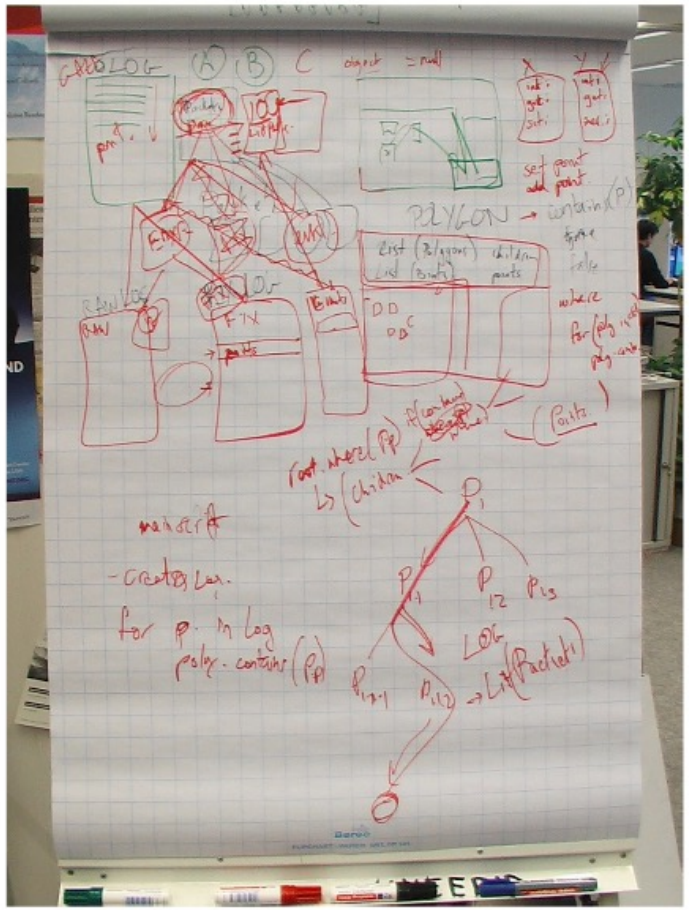
- No “one” way to describe or express software architecture

- Typically, several diagrams to capture different views of system

Class Activity – discuss with your neighbor (3mins)

Which of these diagrams is more useful to software developers?





Cherubini, Venolia, Ko, DeLine

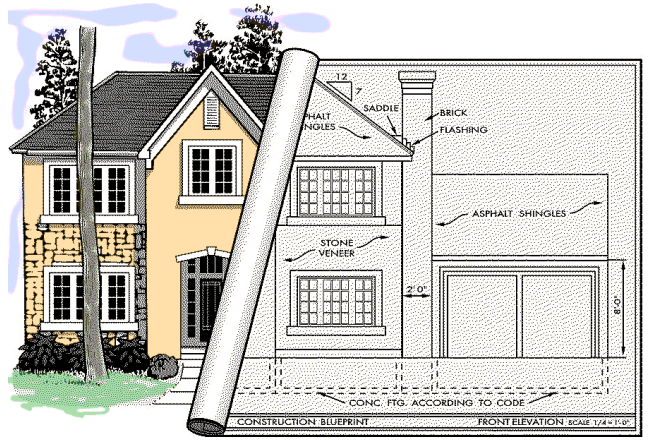
Hallway art at Microsoft

"Let's go to the whiteboard: How and why developers use drawings."

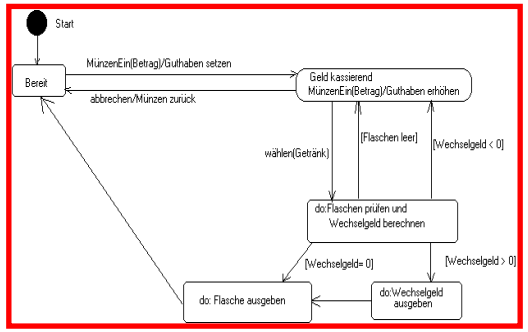
- *Cherubini, Venolia, DeLine, Ko*

Unified Modeling Language (UML)

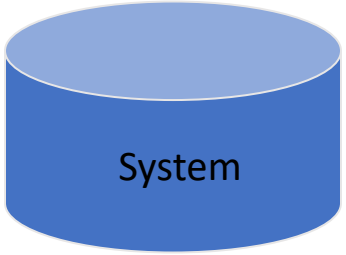
A “diagram language” to visualize, specify, construct and document a software system (historically object-oriented)



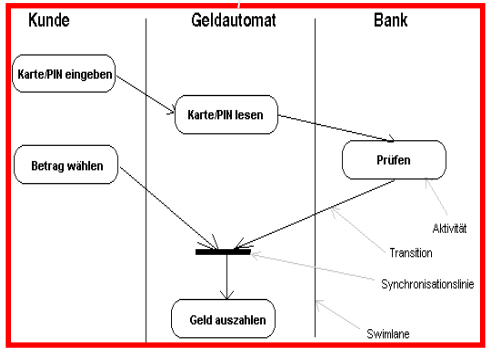
Behavior / Dynamic



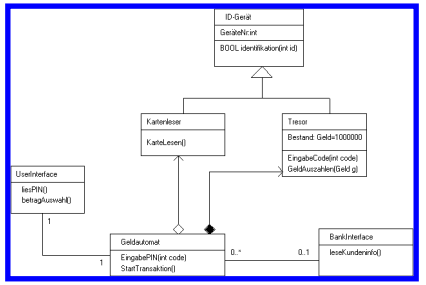
State machine diagram



...

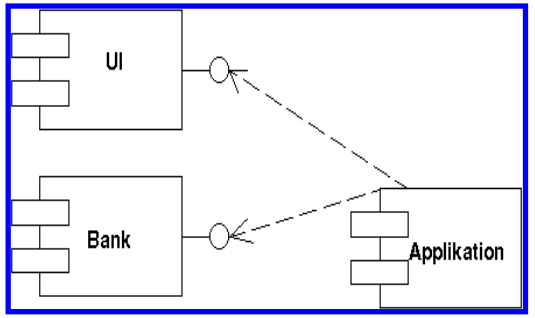


Activity diagram



Class diagram

Structural / Static



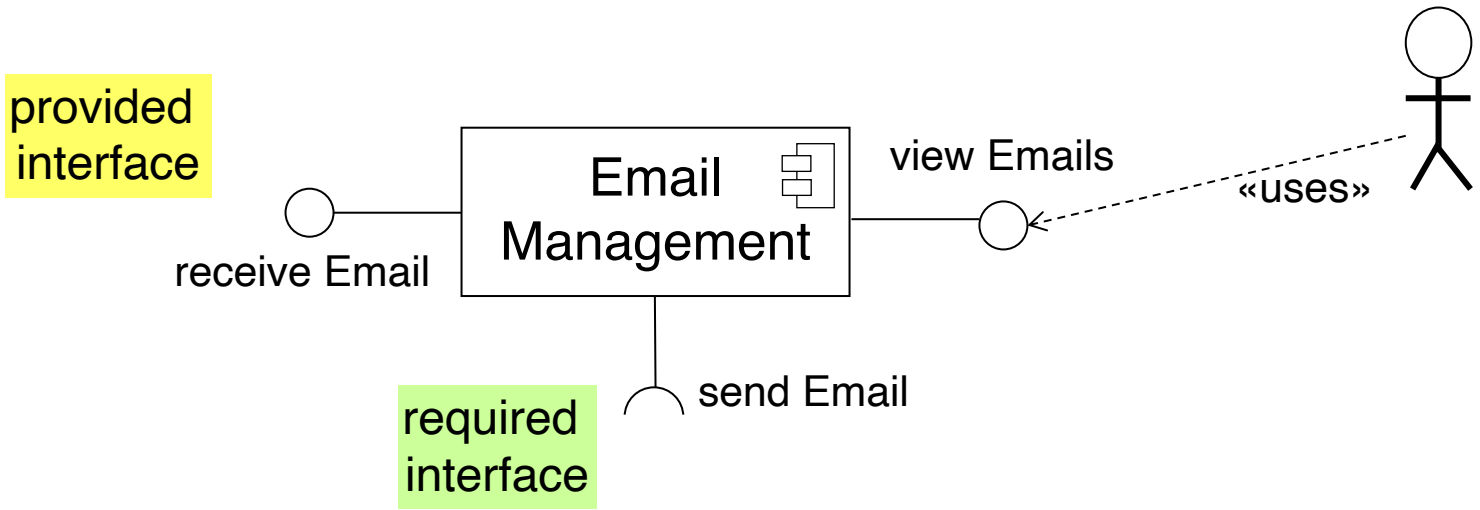
Component diagram

Some of the views – UML Component Diagram

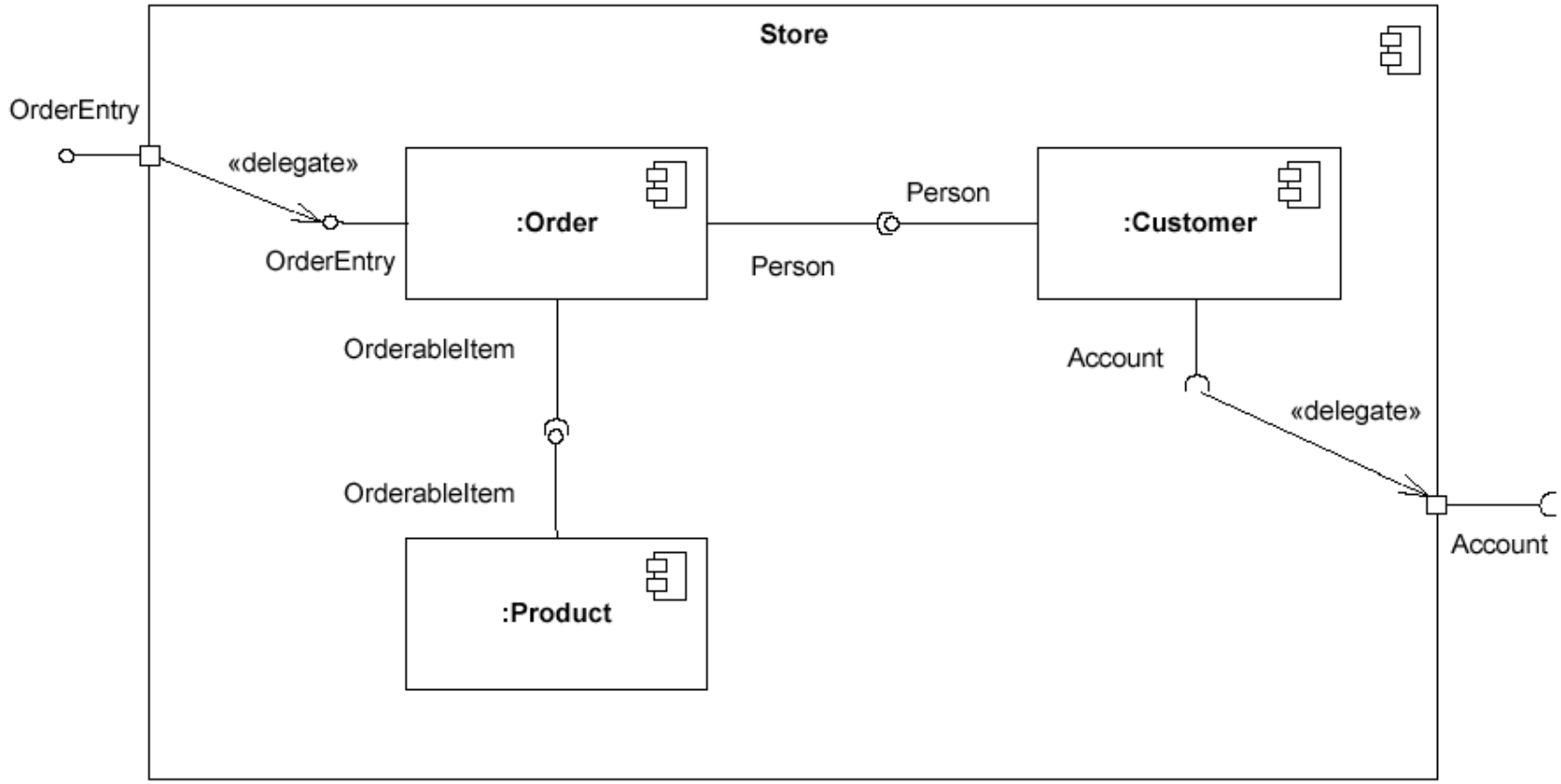
- Focus on core software components
  - Where you had to make a major choice in a framework or library
  - Core high-level components of your application

- Some of the symbols to use (there are also others)
  - rectangle for component, <<component>>,
  - lollipop interface with name, <<requires>>

UML Component Diagram - example



UML Component Diagram - hierarchy

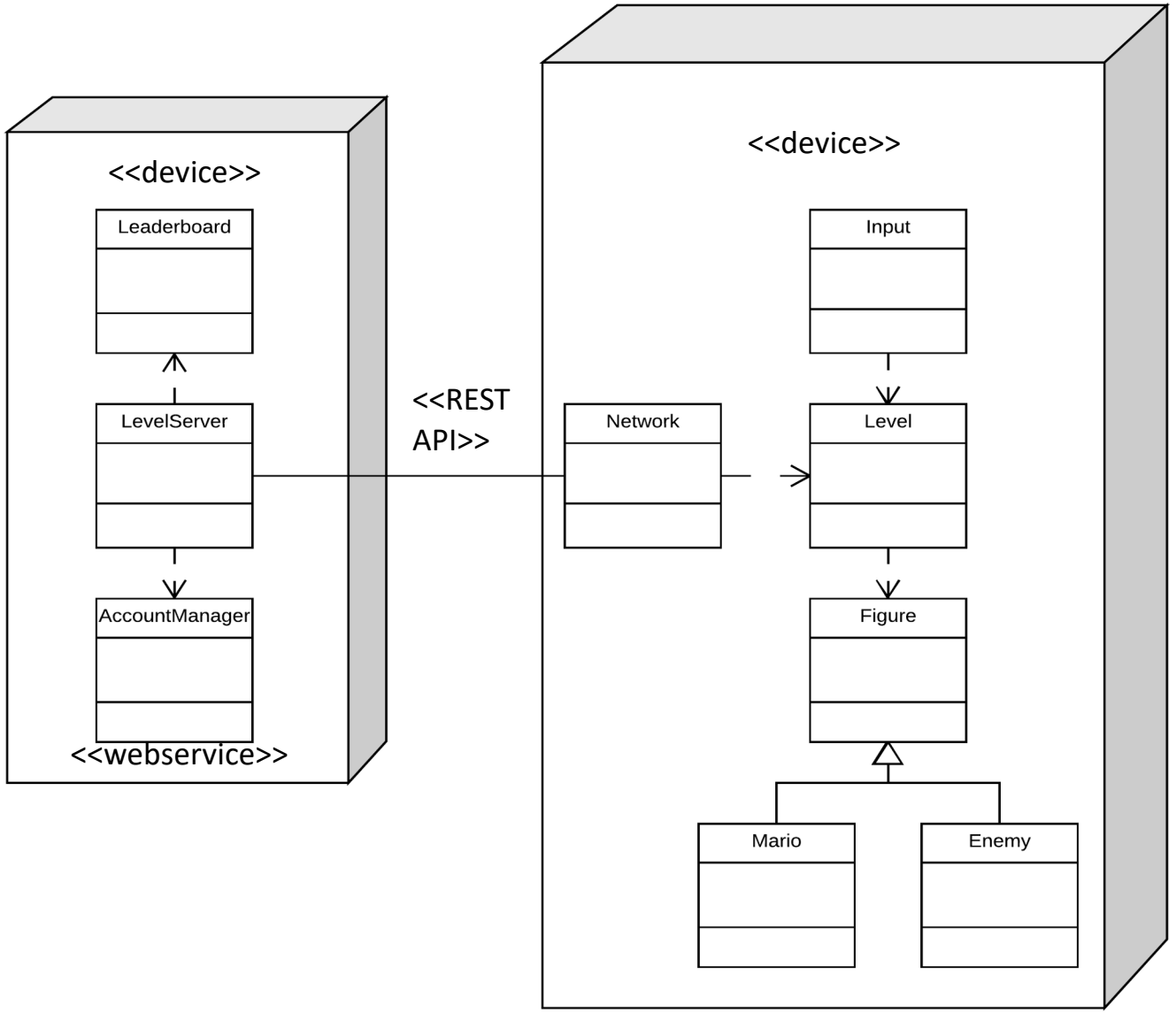


Components can be composed of other components or classes

Another view – UML Deployment Diagram

- Focus on identifying physical devices and what software components they hold (where system parts will reside/execute)
- Contain core software components
- Some of the symbols to use (there are also others)
  - Double rectangles for devices & components, <<device>>, <<webservice>>, <<REST API>>

Deployment of Mario - example



Software Design Exercise II

Learning Objectives

Be able to:

- Read and write basic UML class, component and deployment diagrams

Many thanks to Gail C. Murphy

Software Design Exercise II (time dependent) [https://bit.ly/49NsViF] (15mins)

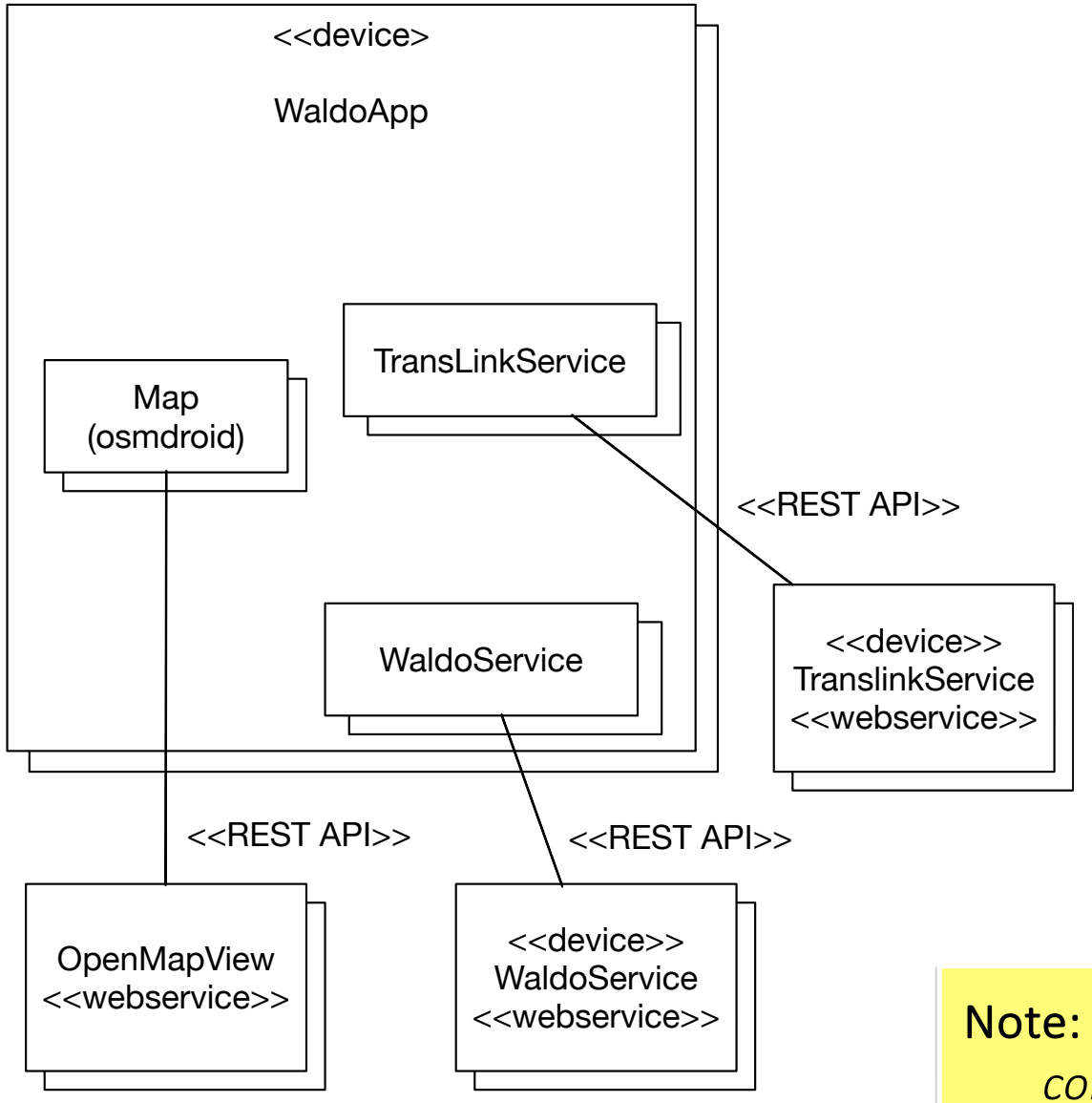
Go back to the system we provided you (Waldo) and your earlier sketches

For the system, sketch (and *copy & paste it into google doc*)

- a UML Deployment Diagram
- a UML Component Diagram
- (a UML Class Diagram)

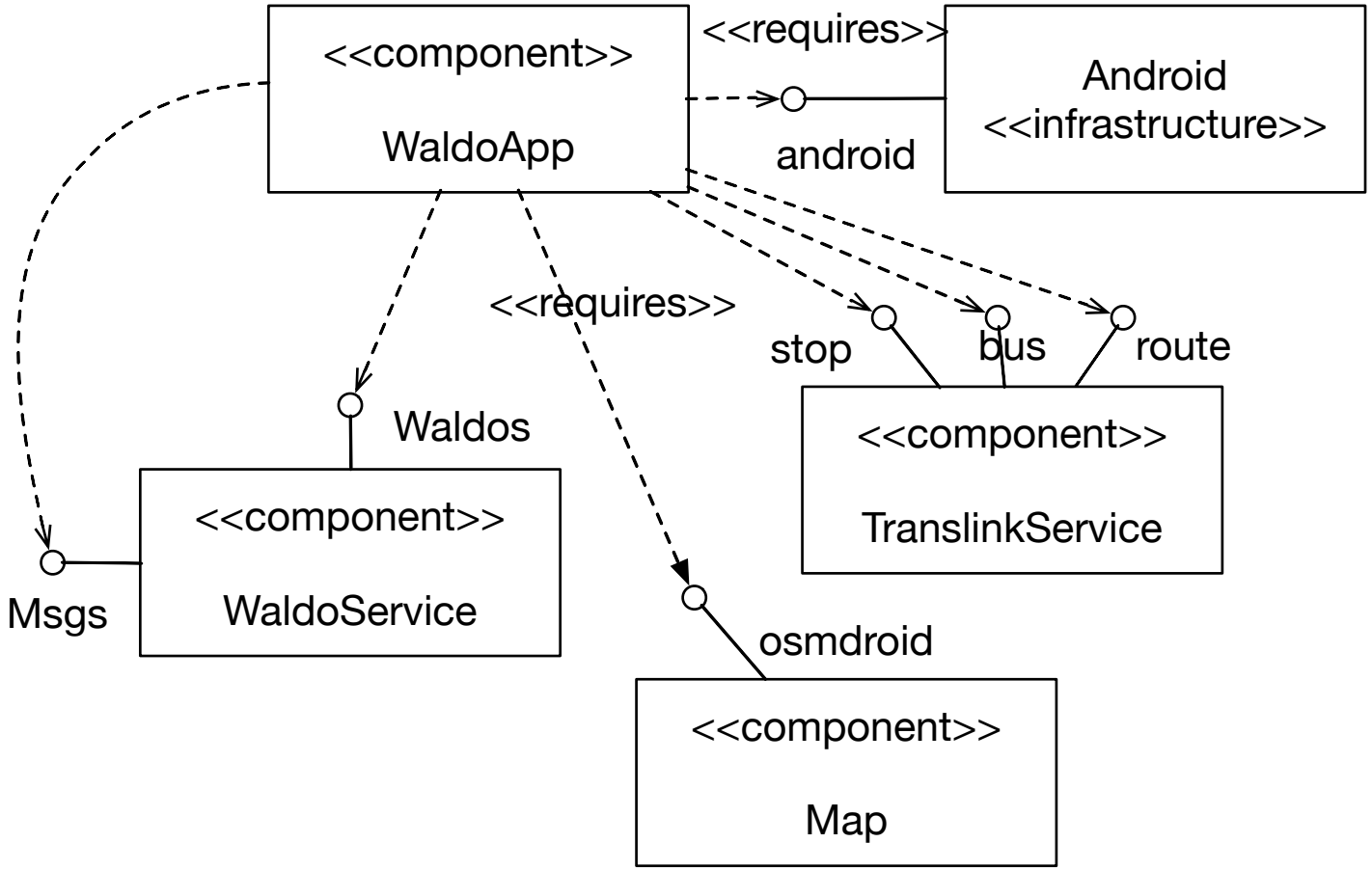
- There is not one “right” answer here…

Software Architecture Exercise II – Deployment Diagram



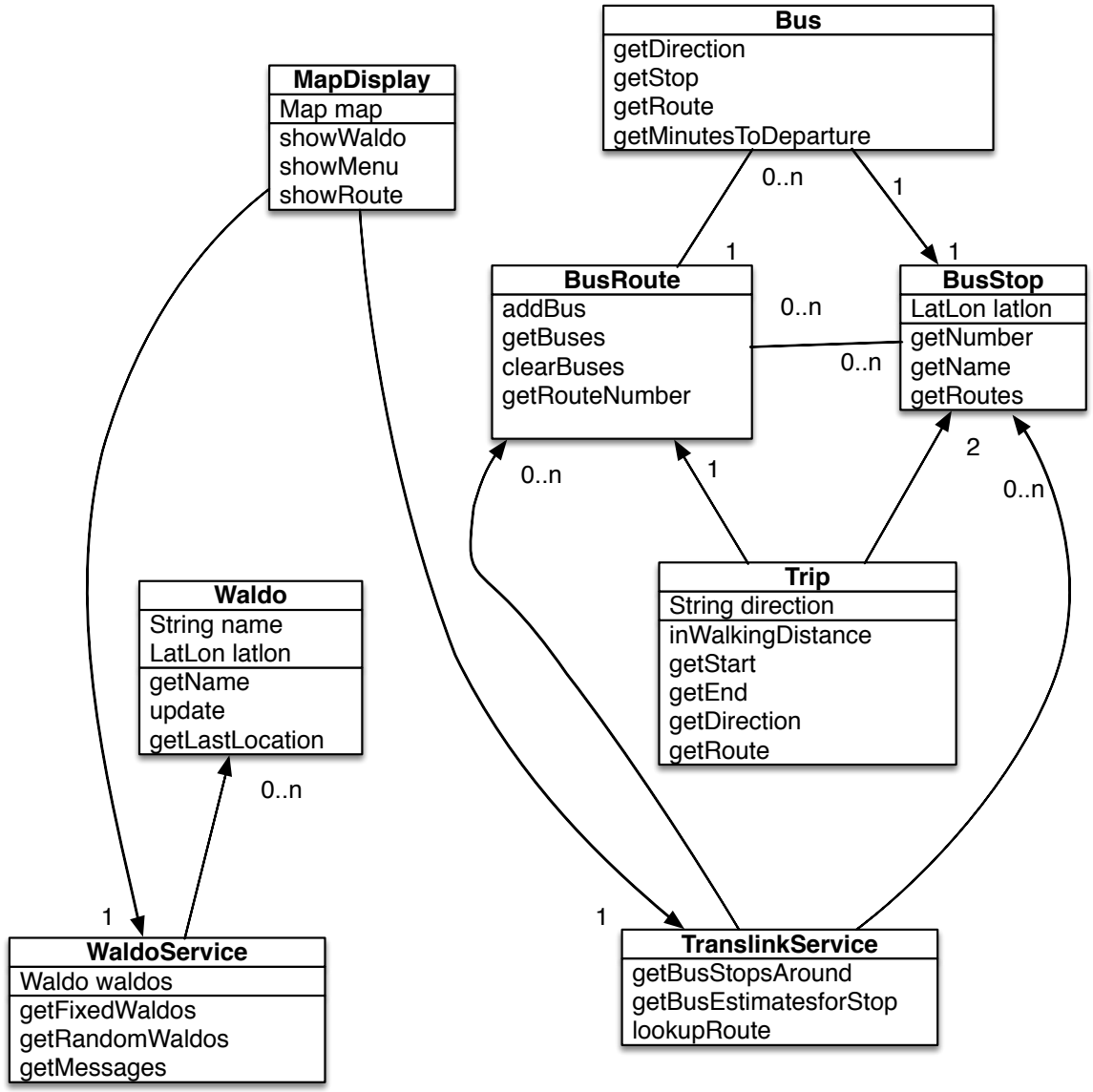
Note: *This diagram is neither complete nor perfect.*

Software Architecture Exercise II – Component Diagram



Note: *This diagram is neither complete nor perfect.*

Software Architecture Exercise II – Class Diagram



Note: *This diagram is neither complete nor perfect.*

A few comments

- Differentiate between component and deployment diagram
- For deployment: make sure to identify all physical devices and associate each component with a device
- For component: make sure to specify who requires and provides something

Kahoot, Quiz & more


Kahoot!

Quiz

1. Technical debt in software can potentially lead to...
   - Enhanced software scalability
   - Reduced productivity
   - Increased risk of defects
   - Organizational challenges
2. What is refactoring primarily used for?
   - Changing a program's output
   - Adding new features to the software
   - Changing the architecture without altering its behavior
   - Improving the software's graphical user interface

Quiz

3. What are the results of information hiding in software development?
   a) It makes software more expensive to maintain
   b) It allows us to not have to know the details of the implementation

Quiz

4. You are working on a system for the university where you want to model the university schedule. The goal is to find out if the students and professors have enough time to switch rooms during the break. You start by creating 3 classes called: “Room”, “Professor” and “Student”. By creating those classes, you encapsulate their behavior and data and abstract away from the details.

- True
- False

Questions to reflect on

1. What is the difference between software design and software architecture?

2. Decomposition: is top down or bottom up generally better?
3. When is it best to use top-down and when bottom-up composition?
4. How much decomposition is too much decomposition?
5. Is it possible to combine top-down and bottom-up compositions?

6. When to use which diagram? (any rules?)
7. How big does a system have to be to benefit from a component diagram?
8. Which diagram is more commonly used: deployment or component diagram?
9. Is there enough time to develop diagrams in agile projects?
10. In which order do you draw diagrams, e.g. are component diagrams always created before class diagrams or not?

Additional resources

David Garlan and Mary Shaw “An Introduction to Software Architecture”

Richard Taylor et al. “Software Architecture: Foundations, Theory and Practice” (Wiley)

Next week:
Lower-level design & decomposing user stories

- Listen to specified videos and readings
- Revisit class and sequence diagrams, we won't spend much time going through specifics
- Fill in Quiz (next Wednesday, 10:00 – 10:20am)