CSE403 ● Software engineering ● sp12

Week 3-4				
Monday	Tuesday	Wednesday	Thursday	Friday
• Design • Reading II due	• Group meetings • SRS due	• Design	• UML	• Design • Progress report due
• Lecture TBA • Reading III due	• Group meetings	• Phil Kimmey on using git @ rover.com	• SDS++ due	• Midterm review – content, format • Progress report due

Today

- Cohesion and coupling – why software entities are organized together and how they interact
- Conceptual integrity – Brooks and others

Cohesion

- Cohesion is the reason that elements are grouped together in a module
- An alternative view is that cohesion is the degree to which elements in a module are related to each other

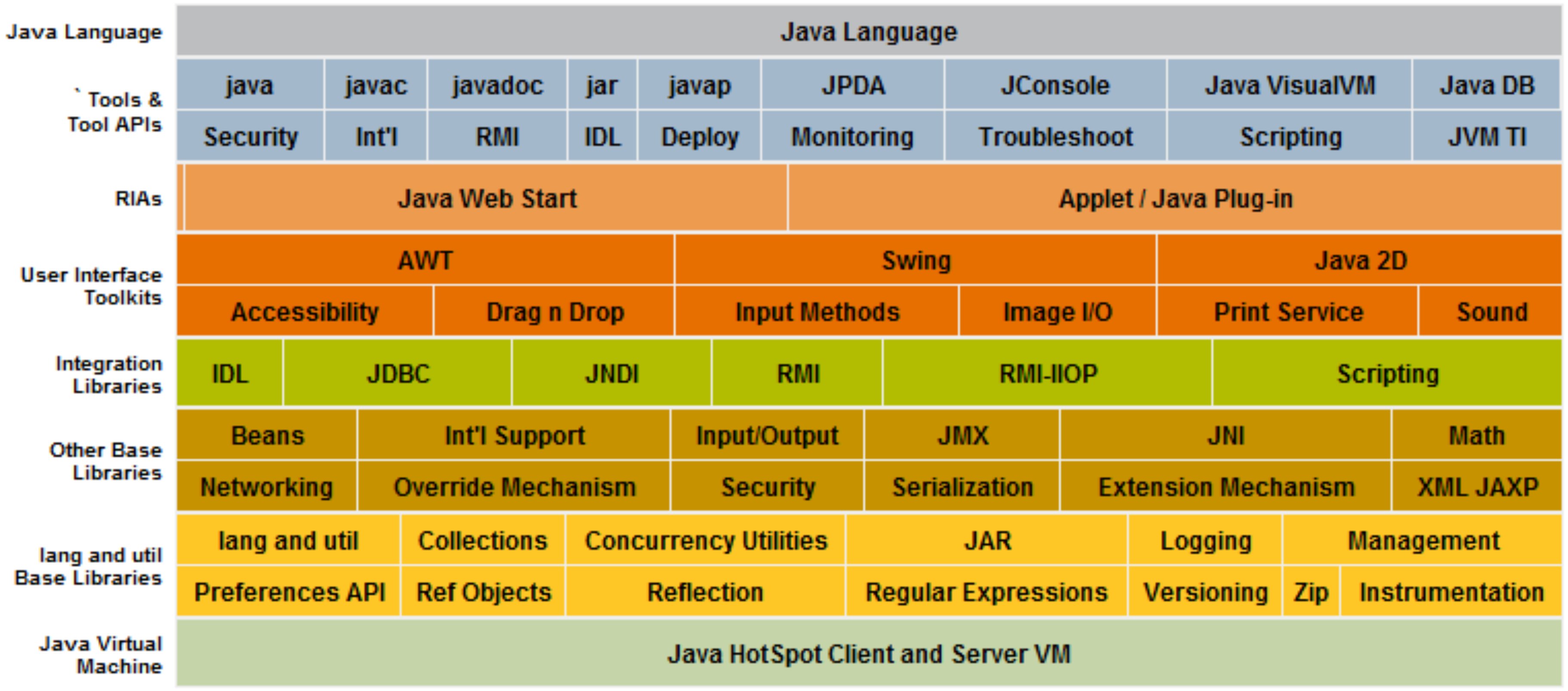
- In general, “better” cohesion has a better reason to combine entities together and makes it easier to
  - understand modules,
  - maintain the software, and
  - reuse modules.

Name	Description
<assert.h>	Contains the assert macro, used to assist with detecting logical errors …
<ctype.h>	Defines set of functions used to classify characters by their types or to …
<errno.h>	For testing error codes reported by library functions.
<locale.h>	Defines localization functions.
<math.h>	Defines common mathematical functions.
<signal.h>	Defines signal handling functions.
<stdalign.h>	For querying and specifying the alignment of objects.
<stdarg.h>	For accessing a varying number of arguments passed to functions.
<stdatomic.h>	For atomic operations on data shared between threads.
<stdbool.h>	Defines a boolean data type.
<stddef.h>	Defines several useful types and macros.
<stdio.h>	Defines core input and output functions
<stdlib.h>	Defines numeric conversion functions, pseudo-random numbers generation functions, memory allocation, process control functions
<string.h>	Defines string handling functions.
<threads.h>	Defines functions for managing multiple Threads as well as mutexes and …
<time.h>	Defines date and time handling functions

stdlib.c

Some removed

Java 7 JDK



On what basis/bases were the stdlib.c and JDK cohesion decisions made?

Extra credit

(Some) Kinds of cohesion

- Coincidental cohesion: grouped despite lack of relationship – e.g., “Utilities”
- Temporal cohesion: grouped together because of when they are processed – e.g., initialization code
- Communicational cohesion: grouped together because they use the same data (whether encapsulated or not)
- Functional cohesion: grouped because they share data and contribute to a well-defined task

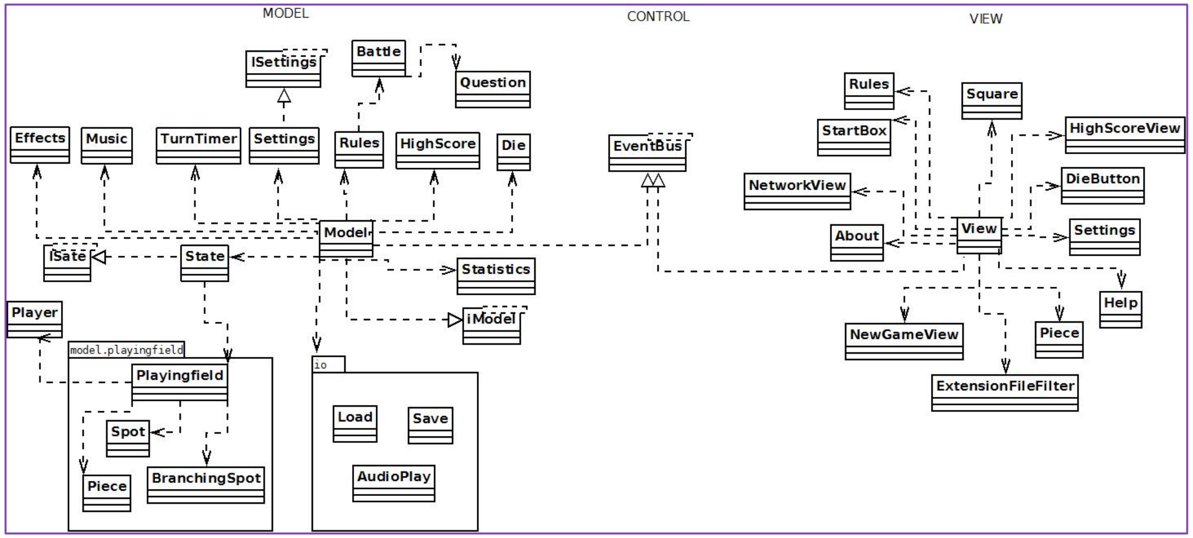
Metrics

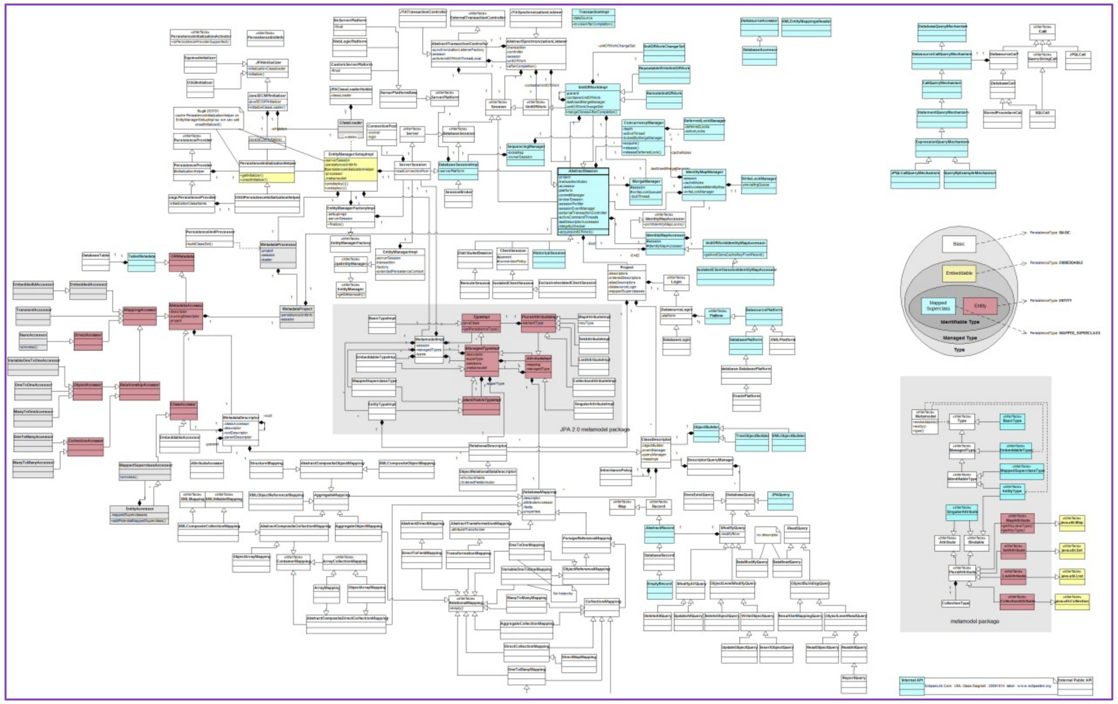
- Categorization is not precise nor even totally ordered
- There are a number of attempts to quantify cohesion
- An example of an oft-cited one is LCOM (Lack of Cohesion Of Methods) based on how many reasons a class has to change
  - `LCOM = 1 - (sum(mf)/M*F)`
    `M` ≡ #methods in the class
    `F` ≡ #instance fields in the class
    `mf` ≡ #methods of the class accessing field `f`
  - All methods in a highly cohesive class use all the instance fields in the class – that is, `sum(mf) = M*F`
- The objective of defining quantitative metrics for cohesion has, to me, been largely unsuccessful because (a) the connection of the metrics to our conceptual understanding is often weak and (b) the metrics rarely provide any "actionable" information

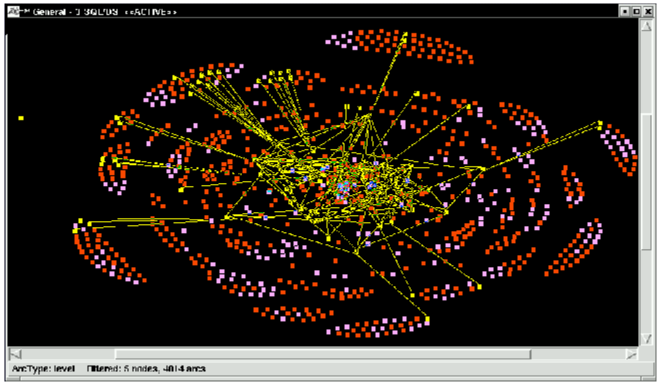
Coupling

- Given a set of modules, coupling characterizes how their entities interact across module boundaries
- An alternative view is that coupling is the degree to which each module relies on the other modules

- In general, “better” (more “loosely”) coupling
  - reduces the ripple effect, where making a change in one module forces changes in other modules
  - simplifies building, testing and reuse
- It is usually the case that better cohesion and better coupling go hand-in-hand


MODEL
CONTROL
VIEW
ISettings
Battle
Question
Effects
Music
TurnTimer
Settings
Rules
HighScore
Die
EventBus
Model
ISate
State
Statistics
iModel
Player
model.playingfield
Playingfield
Spot
Piece
BranchingSpot
io
Load
Save
AudioPlay
Rules
Square
StartBox
HighScoreView
NetworkView
DieButton
View
About
Settings
Help
NewGameView
Piece
ExtensionFileFilter




General - 3 SQL/DB <<ACTIVE>>
ArcType: level
Filtered: 5 nodes, 4014 arcs

(Some) Kinds of coupling

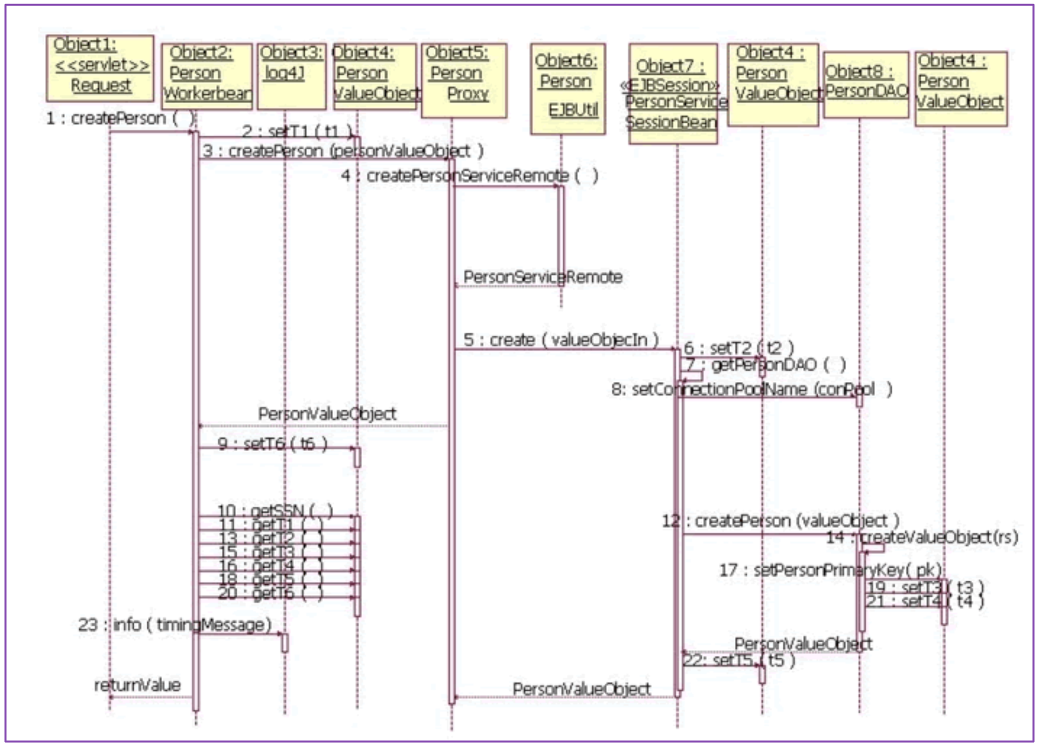
- Content/pathological coupling is when modules are so interconnected that they should essentially be a single module
- Common coupling is when modules share the same global data
- Control coupling is when a module controls the flow of another module, for example by passing a parameter that drives the called module
- Data coupling is when modules share data through parameters
- Message coupling is where all interactions are through parameters and message passing
- No coupling is where modules do not communicate at all with one another

Essentially: what dependences?

- There are a large number of kinds of dependences
  - Control dependence – which module M_1 invokes which M_2?
  - Name dependence – which M_1 knows the name of which M_2?
  - Data-flow dependence – which data D_1 is needed to compute which data D_2?
  - Source file dependence – which file F_1 must be compiled before which F_2?
  - Global data dependence – which D_1 is referenced by which M_2?
  - …
- Of course, the devil is in the details – for example, control dependence
  - static (methods in M_1 can invoke methods in M_2) or dynamic (which methods in M_1 do invoke methods in M_2 and over what test suite)?
  - infrequency (if only one method in M_1 can invoke a method in M_2 at most once) or frequency (methods in M_1 invoke most of the methods in M_2 a lot of times) … and is that static or dynamic

So many dependences

- Every arrow in every UML diagram is a dependence
- Many dependences are implicit in the code and design – they can be determined by human inspection or, more commonly, automated analysis
- Which dependences matter the most?


Object1: <<servlet>> Request
Object2: Person Workerbean
Object3: log4J
Object4: Person ValueObject
Object5: Person Proxy
Object6: Person EJBUtil
Object7 : «EJBSession» PersonService SessionBean
Object4 : Person ValueObject
Object8 : PersonDAO
Object4 : Person ValueObject
1 : createPerson ( )
2 : setT1 (t1)
3 : createPerson (personValueObject)
4 : createPersonServiceRemote ( )
PersonServiceRemote
5 : create (valueObjecIn)
6 : setT2 (t2)
7 : getPersonDAO ( )
8: setConnectionPoolName (conPool)
PersonValueObject
9 : setT6 (t6)
10 : getSSN ( )
11 : getT1 ( )
13 : getT2 ( )
15 : getT3 ( )
16 : getT4 ( )
18 : getT5 ( )
20 : getT6 ( )
12 : createPerson (valueObject)
14 : createValueObject(rs)
17 : setPersonPrimaryKey(pk)
19 : setT3 (t3)
21 : setT4 (t4)
23 : info (timingMessage)
PersonValueObject
22: setT5 (t5)
returnValue
PersonValueObject

Metrics

- Like cohesion, the earlier categorization is not precise nor even totally ordered
- There are a large number of attempts to quantify cohesion
- Here's one very simple one

```
C = 1 / (number of input parameters +
         number of output parameters +
         number of global variables used +
         number of modules called +
         number of modules calling)
```

- The objective of defining quantitative metrics for coupling has, in some sense, been worse than for cohesion because there are so many kinds of dependences, the metrics are often imprecise about dependences, and the tools to extract dependences have a set of complications as well

Tool complications: theoretical

- Example: Call graph (control dependence)
  - *Undecidable* if it must include only pairs **`<a,b>`** where **`a`** calls **`b`** in some possible execution of a program
  - *Conservative* if it never excludes a pair **`<a,b>`** if **`a`** could call **`b`** in some possible execution
  - *Useless* (although conservative and fast to compute) if it includes all pairs **`<a,b>`**
  - *Dynamic* if it only includes a pair **`<a,b>`** if **`a`** calls **`b`** in some actual execution

Tool complications: practical

- Reflection (like in Java)
- Events
- Hidden interactions (such as through the file system)
- C/C++
  - What defines a project? A directory, a make file, etc.?
  - Before the preprocessor is run, or after? If before, over all possible definitions?

Perfect coupling and cohesion

But fails on complexity

Conceptual integrity

- Fred Brooks 1975: *Conceptual integrity is* the *most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas.*
- Brooks 1995: *I am more convinced than ever. Conceptual integrity is central to product quality. Having a system architect is the most important single step toward conceptual integrity.*

Conceptual integrity

- McConnell: “Good software architecture makes the rest of the project easy.”
- Hoare 1985: ““There are two ways of constructing a software design:
  - one way is to make it so simple that there are obviously no deficiencies;
  - the other is to make it so complicated that there are no obvious deficiencies.”
- Rechtin/Maier: “In architecting a new program, all the serious mistakes are made in the first day”

CSE403 ● Software engineering ● sp12

Week 3-4				
Monday	Tuesday	Wednesday	Thursday	Friday
• Design • Reading II due	• Group meetings • SRS due	• Design	• UML	• Design • Progress report due
• Lecture TBA • Reading III due	• Group meetings	• Phil Kimmey on using git @ rover.com	• SDS++ due	• Midterm review – content, format • Progress report due