

Goals and Relevance of Design

- The structure of something is the set of relations between its parts
- Something not built from (recognizable) parts is called unstructured

Design

- structures a system into **recognizable** units (yields software architecture);
- determines the approach for realising the required software;
- provides hierarchical structuring into a **recognizable** number of units at architectural level

Overimplified process model "Design"

10/20

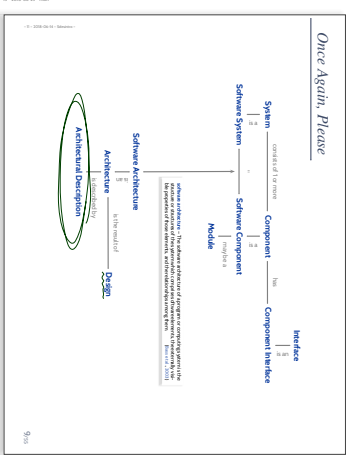
Topic Area Architecture & Design: Content

- VL11
 - Introduction and Vocabulary
 - Software Modelling
 - model, view, viewpoints, 4+1 view
 - Modelling structure
 - UML
- VL12
 - Principles of Design
 - modularity, separation of concerns
 - information hiding and data encapsulation
 - abstract data types, object orientation
 - abstract data types, object orientation
- VL13
 - Design Patterns
 - Modelling behaviour
 - communicating finite automata (CFA)
 - Uppdal query language
 - CRYS Software
 - Unified Modelling Language (UML)
 - an outlook on hierarchical state-machines
- VL14
 - Model-driven Based Software Engineering
- VL15
 - Model-driven Based Software Engineering

Content

- Principles of (Good) Design
 - modularity, separation of concerns
 - information hiding and data encapsulation
 - abstract data types, object orientation
 - ... by example
- Architecture Patterns
 - Layered Architecture, Page-Filer, Model-View-Controller
- Design Patterns
 - Strategy, Examples
- Libraries and Frameworks

Once Again, Please



Principles of (Architectural) Design

- 1) **Modularisation**
 - split software into units / components of **manageable size**
 - provide well-defined interfaces
- 2) **Separation of Concerns**
 - each component should be **responsible for a particular area of tasks**
 - group data and operation on that data functional aspects (functional vs technical, functionality and interaction)
- 3) **Information Hiding**
 - the "need to know principle" / information hiding
 - users (e.g. other developers) need not necessarily know the algorithm and helper data which enable the components' interface
- 4) **Data Encapsulation**
 - offer operators to access component data
 - instead of accessing data variables, lists, etc.) directly
 - many programming languages and systems offer means to **enforce** some of these principles technically, i.e. use these means

1.) Modularisation

modular decomposition – The process of breaking a system into components to facilitate design and development is an element of modular programming (IEEE Std 1191)

modularity – The degree to which a system or computer program is composed of discrete components that can change to one component's requirements (IEEE Std 1191)

- **So, modularity is a property of an architecture.**
- **Goals of modular decomposition:**
 - The structure of each module should be simple and easily comprehensible
 - Information on the implementation of other modules should not be necessary
 - The other modules should not be affected by implementation changes
 - do not avoid design and code reuse, **avoid code changes**
 - **do not avoid design and code reuse, avoid code changes**
- **Bigger changes should be the result of a set of minor changes.**
- **Along as the interface does not change,** it should be possible to try out new versions of modules together.

2.) Separation of Concerns

- **Separation of concerns** is a fundamental principle in software engineering
- each component should be **responsible for a particular area of tasks**
- component which try to cover different task areas tend to be unnecessarily complex, thus hard to understand and maintain
- **Criteria for separation/grouping:**
 - in **object oriented design**, data and operations on that data are grouped into classes
 - sometimes, functional aspects (features) are grouped and related as separate components
 - separate **functional and technical** components
 - Example: logical flow of logical messages in a communication protocol (function) vs. communication messages using a certain technology (technical)
 - assign flexible or variable functionality to own components
 - Example: different networking technology (wires, etc.)
 - assign functionally which is expected to need extensions or changes later to own components
 - separate system **functionality and interaction**
 - Example: most prominently graphical user interface (GUI) and the input/output

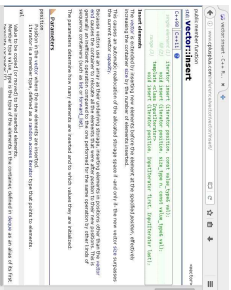
3.) Information Hiding

- By now, we only discussed the grouping of data and operations
 - One should also consider **accessibility**
 - The "need to know principle" is called **information hiding** in SW engineering (Parnas, 1972)
- Information Hiding** – A software development technique in which each module interfaces reveal information about the module that is not in the module's interface specification (IEEE 610.11990)
- **Note:** what is hidden is information which other components need not know (e.g. how data is stored and accessed, how operations are implemented)
 - In other words, **information hiding is about making explicit** for one component which data or operations other component may use of this component
 - **Advantages / goals:**
 - Hide data and operations which are **shared** with other components, avoiding
 - as long as the relationships between them are the same (e.g. the employed sorting algorithm)
 - **HOW** other components cannot (**unintentionally**) depend on details they are not supposed to
 - Components can be verified / validated in isolation

Information Hiding – A software development technique in which each module interfaces reveal information about the module that is not in the module's interface specification (IEEE 610.11990)

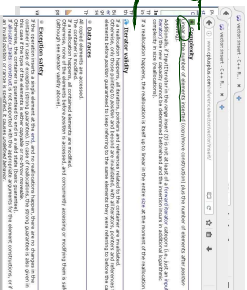
4.) Data Encapsulation

- **Similar direction: data encapsulation** (example later)
- Do not access data variables, files, etc. directly where needed, but encapsulate the data in a component which offers operations to access (read, write, etc.) the data
- **Real world example:** Users do not write to bank accounts directly, only bank clerks do.



4.) Data Encapsulation

- **Similar direction: data encapsulation** (example later)
- Do not access data variables, files, etc. directly where needed, but encapsulate the data in a component which offers operations to access (read, write, etc.) the data
- **Real world example:** Users do not write to bank accounts directly, only bank clerks do.



4.) Data Encapsulation

- Similar direction **data encapsulation** (example later)
- Do not access data (variables, files, etc.) directly when needed, but encapsulate the data in a component which offers operations to access it (read, write, etc.) the data.
- Real-World Example: Users don't write to bank accounts directly, only bank clerks do.

Information Hiding and data encapsulation – when enforced technically (examples later) – usually **comes at the price of worse efficiency.**

- It is more efficient to read a component's data directly
- Knowledge of the internal structure of the component may enable more efficient operations
- Knowledge of the internal structure of the component may enable more efficient operations
- Example: If a sequence of data items is stored as a singly-linked list, accessing the data items in list-order may be more efficient than accessing them in reverse order by position.
- **Good modules** give advantages in their documentation (e.g. C++ standard library)
- **Example:** If an implementation stores intermediate results at a certain place, it may be tempting to optimize for performance by changing the location of the results.
- → **maintainance nightmare** – if the result is needed in another context
- add its corresponding operation equality to the interface

It's worth today's hardware and programming languages, this is hardly an issue any more. It is the issue of **efficiency**, if you're really picky.

11/09

A Classification of Modules (Naeff, 1989)

- **functional modules**
 - group computations which belong together logically
 - do not have memory or state that is behaviour of offered functionality, does not depend on prior program
 - Example: mathematical function, transformations
- **data object modules**
 - abstract representation of data
 - implement a user-defined data type in form of an abstract data type (ADT)
 - Example: modules encapsulating global configuration data, database
- **data type modules**
 - implement a user-defined data type in form of an abstract data type (ADT)
 - Example: game object
- **data object-oriented design**
 - classes are data type modules
 - data object modules correspond to classes offering only data methods or interfaces (← linked)
 - **functional modules** access seldom, one example is game database.

12/09

Example: Module 'List of Names'

- Task: store a list of names in N of type 'list of strings'.
- Operations (in interface of the module)
 - insert(s: string):
 - $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, \forall 0 \leq i < m \wedge n_i < n_{i+1}$
 - post-condition: $\bigcirc_{0 \leq i < m} n_{i+1} < n_i$, if $n_i < m$, $n_{i+1} < n_i \wedge n_{i+1} < n_i \wedge \text{add}(N)$ otherwise
 - $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, \forall 0 \leq i < m \wedge n_i < n_{i+1}$
- remove(last):
 - pre-condition: $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, 0 \leq i < m$
 - post-condition: $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-2}, m \in \mathbb{N}_0, 0 \leq i < m$
- get(last): string:
 - pre-condition: $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, 0 \leq i < m$
 - post-condition: $N = \text{add}(N, \text{get}(last))$
- dump():
 - pre-condition: $N = n_0, \dots, n_{m-1}, m \in \mathbb{N}_0$
 - post-condition: $N = \text{add}(N)$
 - side-effect: n_0, \dots, n_{m-1} printed to standard output in this order

14/09

A Possible Implementation: Plain List, no Duplicates

```

1  private:
2  private:
3  private:
4  private:
5  private:
6  private:
7  private:
8  private:
9  private:
10 private:
11 private:
12 private:
13 private:
14 private:
15 private:
16 private:
17 private:
18 private:
19 private:
20 private:
21 private:
22 private:
23 private:
24 private:
25 private:
26 private:
27 private:
28 private:
29 private:
30 private:
31 private:
32 private:
33 private:
34 private:
35 private:
36 private:
37 private:
38 private:
39 private:
40 private:
41 private:
42 private:
43 private:
44 private:
45 private:
46 private:
47 private:
48 private:
49 private:
50 private:
51 private:
52 private:
53 private:
54 private:
55 private:
56 private:
57 private:
58 private:
59 private:
60 private:
61 private:
62 private:
63 private:
64 private:
65 private:
66 private:
67 private:
68 private:
69 private:
70 private:
71 private:
72 private:
73 private:
74 private:
75 private:
76 private:
77 private:
78 private:
79 private:
80 private:
81 private:
82 private:
83 private:
84 private:
85 private:
86 private:
87 private:
88 private:
89 private:
90 private:
91 private:
92 private:
93 private:
94 private:
95 private:
96 private:
97 private:
98 private:
99 private:
100 private:
101 private:
102 private:
103 private:
104 private:
105 private:
106 private:
107 private:
108 private:
109 private:
110 private:
111 private:
112 private:
113 private:
114 private:
115 private:
116 private:
117 private:
118 private:
119 private:
120 private:
121 private:
122 private:
123 private:
124 private:
125 private:
126 private:
127 private:
128 private:
129 private:
130 private:
131 private:
132 private:
133 private:
134 private:
135 private:
136 private:
137 private:
138 private:
139 private:
140 private:
141 private:
142 private:
143 private:
144 private:
145 private:
146 private:
147 private:
148 private:
149 private:
150 private:
151 private:
152 private:
153 private:
154 private:
155 private:
156 private:
157 private:
158 private:
159 private:
160 private:
161 private:
162 private:
163 private:
164 private:
165 private:
166 private:
167 private:
168 private:
169 private:
170 private:
171 private:
172 private:
173 private:
174 private:
175 private:
176 private:
177 private:
178 private:
179 private:
180 private:
181 private:
182 private:
183 private:
184 private:
185 private:
186 private:
187 private:
188 private:
189 private:
190 private:
191 private:
192 private:
193 private:
194 private:
195 private:
196 private:
197 private:
198 private:
199 private:
200 private:
201 private:
202 private:
203 private:
204 private:
205 private:
206 private:
207 private:
208 private:
209 private:
210 private:
211 private:
212 private:
213 private:
214 private:
215 private:
216 private:
217 private:
218 private:
219 private:
220 private:
221 private:
222 private:
223 private:
224 private:
225 private:
226 private:
227 private:
228 private:
229 private:
230 private:
231 private:
232 private:
233 private:
234 private:
235 private:
236 private:
237 private:
238 private:
239 private:
240 private:
241 private:
242 private:
243 private:
244 private:
245 private:
246 private:
247 private:
248 private:
249 private:
250 private:
251 private:
252 private:
253 private:
254 private:
255 private:
256 private:
257 private:
258 private:
259 private:
260 private:
261 private:
262 private:
263 private:
264 private:
265 private:
266 private:
267 private:
268 private:
269 private:
270 private:
271 private:
272 private:
273 private:
274 private:
275 private:
276 private:
277 private:
278 private:
279 private:
280 private:
281 private:
282 private:
283 private:
284 private:
285 private:
286 private:
287 private:
288 private:
289 private:
290 private:
291 private:
292 private:
293 private:
294 private:
295 private:
296 private:
297 private:
298 private:
299 private:
300 private:
301 private:
302 private:
303 private:
304 private:
305 private:
306 private:
307 private:
308 private:
309 private:
310 private:
311 private:
312 private:
313 private:
314 private:
315 private:
316 private:
317 private:
318 private:
319 private:
320 private:
321 private:
322 private:
323 private:
324 private:
325 private:
326 private:
327 private:
328 private:
329 private:
330 private:
331 private:
332 private:
333 private:
334 private:
335 private:
336 private:
337 private:
338 private:
339 private:
340 private:
341 private:
342 private:
343 private:
344 private:
345 private:
346 private:
347 private:
348 private:
349 private:
350 private:
351 private:
352 private:
353 private:
354 private:
355 private:
356 private:
357 private:
358 private:
359 private:
360 private:
361 private:
362 private:
363 private:
364 private:
365 private:
366 private:
367 private:
368 private:
369 private:
370 private:
371 private:
372 private:
373 private:
374 private:
375 private:
376 private:
377 private:
378 private:
379 private:
380 private:
381 private:
382 private:
383 private:
384 private:
385 private:
386 private:
387 private:
388 private:
389 private:
390 private:
391 private:
392 private:
393 private:
394 private:
395 private:
396 private:
397 private:
398 private:
399 private:
400 private:
401 private:
402 private:
403 private:
404 private:
405 private:
406 private:
407 private:
408 private:
409 private:
410 private:
411 private:
412 private:
413 private:
414 private:
415 private:
416 private:
417 private:
418 private:
419 private:
420 private:
421 private:
422 private:
423 private:
424 private:
425 private:
426 private:
427 private:
428 private:
429 private:
430 private:
431 private:
432 private:
433 private:
434 private:
435 private:
436 private:
437 private:
438 private:
439 private:
440 private:
441 private:
442 private:
443 private:
444 private:
445 private:
446 private:
447 private:
448 private:
449 private:
450 private:
451 private:
452 private:
453 private:
454 private:
455 private:
456 private:
457 private:
458 private:
459 private:
460 private:
461 private:
462 private:
463 private:
464 private:
465 private:
466 private:
467 private:
468 private:
469 private:
470 private:
471 private:
472 private:
473 private:
474 private:
475 private:
476 private:
477 private:
478 private:
479 private:
480 private:
481 private:
482 private:
483 private:
484 private:
485 private:
486 private:
487 private:
488 private:
489 private:
490 private:
491 private:
492 private:
493 private:
494 private:
495 private:
496 private:
497 private:
498 private:
499 private:
500 private:
501 private:
502 private:
503 private:
504 private:
505 private:
506 private:
507 private:
508 private:
509 private:
510 private:
511 private:
512 private:
513 private:
514 private:
515 private:
516 private:
517 private:
518 private:
519 private:
520 private:
521 private:
522 private:
523 private:
524 private:
525 private:
526 private:
527 private:
528 private:
529 private:
530 private:
531 private:
532 private:
533 private:
534 private:
535 private:
536 private:
537 private:
538 private:
539 private:
540 private:
541 private:
542 private:
543 private:
544 private:
545 private:
546 private:
547 private:
548 private:
549 private:
550 private:
551 private:
552 private:
553 private:
554 private:
555 private:
556 private:
557 private:
558 private:
559 private:
560 private:
561 private:
562 private:
563 private:
564 private:
565 private:
566 private:
567 private:
568 private:
569 private:
570 private:
571 private:
572 private:
573 private:
574 private:
575 private:
576 private:
577 private:
578 private:
579 private:
580 private:
581 private:
582 private:
583 private:
584 private:
585 private:
586 private:
587 private:
588 private:
589 private:
590 private:
591 private:
592 private:
593 private:
594 private:
595 private:
596 private:
597 private:
598 private:
599 private:
600 private:
601 private:
602 private:
603 private:
604 private:
605 private:
606 private:
607 private:
608 private:
609 private:
610 private:
611 private:
612 private:
613 private:
614 private:
615 private:
616 private:
617 private:
618 private:
619 private:
620 private:
621 private:
622 private:
623 private:
624 private:
625 private:
626 private:
627 private:
628 private:
629 private:
630 private:
631 private:
632 private:
633 private:
634 private:
635 private:
636 private:
637 private:
638 private:
639 private:
640 private:
641 private:
642 private:
643 private:
644 private:
645 private:
646 private:
647 private:
648 private:
649 private:
650 private:
651 private:
652 private:
653 private:
654 private:
655 private:
656 private:
657 private:
658 private:
659 private:
660 private:
661 private:
662 private:
663 private:
664 private:
665 private:
666 private:
667 private:
668 private:
669 private:
670 private:
671 private:
672 private:
673 private:
674 private:
675 private:
676 private:
677 private:
678 private:
679 private:
680 private:
681 private:
682 private:
683 private:
684 private:
685 private:
686 private:
687 private:
688 private:
689 private:
690 private:
691 private:
692 private:
693 private:
694 private:
695 private:
696 private:
697 private:
698 private:
699 private:
700 private:
701 private:
702 private:
703 private:
704 private:
705 private:
706 private:
707 private:
708 private:
709 private:
710 private:
711 private:
712 private:
713 private:
714 private:
715 private:
716 private:
717 private:
718 private:
719 private:
720 private:
721 private:
722 private:
723 private:
724 private:
725 private:
726 private:
727 private:
728 private:
729 private:
730 private:
731 private:
732 private:
733 private:
734 private:
735 private:
736 private:
737 private:
738 private:
739 private:
740 private:
741 private:
742 private:
743 private:
744 private:
745 private:
746 private:
747 private:
748 private:
749 private:
750 private:
751 private:
752 private:
753 private:
754 private:
755 private:
756 private:
757 private:
758 private:
759 private:
760 private:
761 private:
762 private:
763 private:
764 private:
765 private:
766 private:
767 private:
768 private:
769 private:
770 private:
771 private:
772 private:
773 private:
774 private:
775 private:
776 private:
777 private:
778 private:
779 private:
780 private:
781 private:
782 private:
783 private:
784 private:
785 private:
786 private:
787 private:
788 private:
789 private:
790 private:
791 private:
792 private:
793 private:
794 private:
795 private:
796 private:
797 private:
798 private:
799 private:
800 private:
801 private:
802 private:
803 private:
804 private:
805 private:
806 private:
807 private:
808 private:
809 private:
810 private:
811 private:
812 private:
813 private:
814 private:
815 private:
816 private:
817 private:
818 private:
819 private:
820 private:
821 private:
822 private:
823 private:
824 private:
825 private:
826 private:
827 private:
828 private:
829 private:
830 private:
831 private:
832 private:
833 private:
834 private:
835 private:
836 private:
837 private:
838 private:
839 private:
840 private:
841 private:
842 private:
843 private:
844 private:
845 private:
846 private:
847 private:
848 private:
849 private:
850 private:
851 private:
852 private:
853 private:
854 private:
855 private:
856 private:
857 private:
858 private:
859 private:
860 private:
861 private:
862 private:
863 private:
864 private:
865 private:
866 private:
867 private:
868 private:
869 private:
870 private:
871 private:
872 private:
873 private:
874 private:
875 private:
876 private:
877 private:
878 private:
879 private:
880 private:
881 private:
882 private:
883 private:
884 private:
885 private:
886 private:
887 private:
888 private:
889 private:
890 private:
891 private:
892 private:
893 private:
894 private:
895 private:
896 private:
897 private:
898 private:
899 private:
900 private:
901 private:
902 private:
903 private:
904 private:
905 private:
906 private:
907 private:
908 private:
909 private:
910 private:
911 private:
912 private:
913 private:
914 private:
915 private:
916 private:
917 private:
918 private:
919 private:
920 private:
921 private:
922 private:
923 private:
924 private:
925 private:
926 private:
927 private:
928 private:
929 private:
930 private:
931 private:
932 private:
933 private:
934 private:
935 private:
936 private:
937 private:
938 private:
939 private:
940 private:
941 private:
942 private:
943 private:
944 private:
945 private:
946 private:
947 private:
948 private:
949 private:
950 private:
951 private:
952 private:
953 private:
954 private:
955 private:
956 private:
957 private:
958 private:
959 private:
960 private:
961 private:
962 private:
963 private:
964 private:
965 private:
966 private:
967 private:
968 private:
969 private:
970 private:
971 private:
972 private:
973 private:
974 private:
975 private:
976 private:
977 private:
978 private:
979 private:
980 private:
981 private:
982 private:
983 private:
984 private:
985 private:
986 private:
987 private:
988 private:
989 private:
990 private:
991 private:
992 private:
993 private:
994 private:
995 private:
996 private:
997 private:
998 private:
999 private:
1000 private:

```

15/09

Example

- (i) information hiding and data encapsulation, **not enforced**
- (ii) → negative effects when requirements change.
- (iii) enforcing information hiding and data encapsulation by modules
- (iv) abstract data types.
- (v) **object oriented without** information hiding and data encapsulation.
- (vi) **object oriented with** information hiding and data encapsulation.

13/09

Change Interface: Support Duplicate Names

- Task: in addition, **count**() should tell how many vs we have
- Operations (in interface of the module)
 - insert(s: string):
 - $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, \forall 0 \leq i < m \wedge n_i < n_{i+1}$
 - post-condition: $\bigcirc_{0 \leq i < m} n_{i+1} < n_i$, if $n_i < m$, $n_{i+1} < n_i \wedge n_{i+1} < n_i \wedge \text{count}(n) = 1$
 - $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, \forall 0 \leq i < m$
 - if $n_i < m$, $n_{i+1} < n_i \wedge n_{i+1} < n_i \wedge \text{count}(n) = 1$
 - if $n_i < m$, for some $0 \leq i < m$, $N = \text{add}(N, \text{count}(n)) = \text{add}(\text{count}(n)) + 1$
- remove(last):
 - pre-condition: $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, 0 \leq i < m$
 - post-condition: $N = \text{add}(N, \text{count}(n)) = \text{add}(\text{count}(n)) - 1$
 - $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, 0 \leq i < m$
 - if $n_i < m$, $n_{i+1} < n_i \wedge n_{i+1} < n_i \wedge \text{count}(n) = 1$
 - if $n_i < m$, for some $0 \leq i < m$, $N = \text{add}(N, \text{count}(n)) = \text{add}(\text{count}(n)) - 1$
- get(last): string, and dump():
 - unchanged contract

16/09

Object Oriented

```
public class Student {
    private String
    private int age;
    private String name;

    void register(int age, String name) {
        this.age = age;
        this.name = name;
    }

    void display() {
        System.out.println("Age: " + age);
        System.out.println("Name: " + name);
    }

    void setName(String name) {
        this.name = name;
    }

    void setAge(int age) {
        this.age = age;
    }

    public static void main(String[] args) {
        Student s = new Student();
        s.register(18, "John");
        s.display();
        s.setName("Jane");
        s.setAge(20);
        s.display();
    }
}
```

Source

```
public class Student {
    private String name;
    private int age;

    void register(int age, String name) {
        this.age = age;
        this.name = name;
    }

    void display() {
        System.out.println("Age: " + age);
        System.out.println("Name: " + name);
    }

    void setName(String name) {
        this.name = name;
    }

    void setAge(int age) {
        this.age = age;
    }

    public static void main(String[] args) {
        Student s = new Student();
        s.register(18, "John");
        s.display();
        s.setName("Jane");
        s.setAge(20);
        s.display();
    }
}
```

Output

```
Age: 18
Name: John
Age: 20
Name: Jane
```

20.00

Object Oriented + Data Encapsulation / Information Hiding

```
public class Student {
    private String
    private int age;
    private String name;

    void register(int age, String name) {
        this.age = age;
        this.name = name;
    }

    void display() {
        System.out.println("Age: " + age);
        System.out.println("Name: " + name);
    }

    void setName(String name) {
        this.name = name;
    }

    void setAge(int age) {
        this.age = age;
    }

    public static void main(String[] args) {
        Student s = new Student();
        s.register(18, "John");
        s.display();
        s.setName("Jane");
        s.setAge(20);
        s.display();
    }
}
```

Source

```
public class Student {
    private String name;
    private int age;

    void register(int age, String name) {
        this.age = age;
        this.name = name;
    }

    void display() {
        System.out.println("Age: " + age);
        System.out.println("Name: " + name);
    }

    void setName(String name) {
        this.name = name;
    }

    void setAge(int age) {
        this.age = age;
    }

    public static void main(String[] args) {
        Student s = new Student();
        s.register(18, "John");
        s.display();
        s.setName("Jane");
        s.setAge(20);
        s.display();
    }
}
```

Output

```
Age: 18
Name: John
Age: 20
Name: Jane
```

21.00

Object Oriented + Data Encapsulation / Information Hiding

```
public class Student {
    private String
    private int age;
    private String name;

    void register(int age, String name) {
        this.age = age;
        this.name = name;
    }

    void display() {
        System.out.println("Age: " + age);
        System.out.println("Name: " + name);
    }

    void setName(String name) {
        this.name = name;
    }

    void setAge(int age) {
        this.age = age;
    }

    public static void main(String[] args) {
        Student s = new Student();
        s.register(18, "John");
        s.display();
        s.setName("Jane");
        s.setAge(20);
        s.display();
    }
}
```

Source

```
public class Student {
    private String name;
    private int age;

    void register(int age, String name) {
        this.age = age;
        this.name = name;
    }

    void display() {
        System.out.println("Age: " + age);
        System.out.println("Name: " + name);
    }

    void setName(String name) {
        this.name = name;
    }

    void setAge(int age) {
        this.age = age;
    }

    public static void main(String[] args) {
        Student s = new Student();
        s.register(18, "John");
        s.display();
        s.setName("Jane");
        s.setAge(20);
        s.display();
    }
}
```

Output

```
Age: 18
Name: John
Age: 20
Name: Jane
```

22.00

"Tell Them What You've Told Them"

- (i) Information hiding and data encapsulation **not enforced**.
- (ii) → negative effects when requirements change.
- (iii) enforcing information hiding and data encapsulation by modules.
- (iv) abstract data types.
- (v) object oriented **without** information hiding and data encapsulation.
- (vi) object oriented **with** information hiding and data encapsulation.

22.00

Content

- Principles of (Good) Design
 - modularity, separation of concerns
 - information hiding and data encapsulation
 - abstract data types, object orientation
 - ... by example
- Architecture Patterns
 - Layered Architecture, Page-Fixer, Model-View-Controller
- Design Patterns
 - Strategy, Examples
- Libraries and Frameworks

23.00

Architecture Patterns

24.00

Introduction

- Over decades of software engineering, many clever, proved and tested designs of solutions for particular problems emerged
- Question: can we generalize, document and re-use these designs?
- Goals:
 - "don't reinvent the wheel"
 - benefit from "clever" from "proven and tested" and from "solution"

architectural pattern – An architectural pattern expresses a fundamental structural or generative schema for software systems. It specifies their responsibilities and includes rules and guidelines for organizing the relationships between them.
Buchanan et al. (1994)

25/16

Introduction Cont'd

architectural pattern: An architectural pattern expresses a fundamental structural or generative schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.
Buchanan et al. (1994)

- Using an architectural pattern
 - impose certain characteristics or properties of the software system (e.g., performance, security, maintainability, etc.),
 - determine structure on a high level of the architecture, thus is typically a central and fundamental design decision
- The information that "where, how, ..." a well-known architecture / design pattern **helps** in a given software can
 - make comparison and maintenance significantly easier
 - avoid errors

26/16

Layered Architectures

Example: Layered Architectures

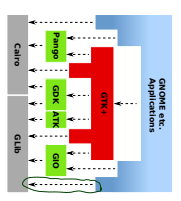
- (Calligaris, 2003)
- A layer whose components only interact with components of their three **neighboring** layers is called **protocol based layer** or **protocol layer** (which is only used by the layers directly above and below it)
- Example: The ISO/OSI reference model.



28/16

Example: Layered Architectures Cont'd

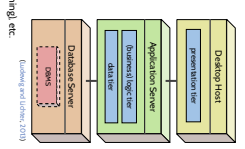
- Object-oriented layer**: interacts with layers directly (and possibly further) above and below
- Rules**: the components of a layer may use
 - only** components of the protocol-based layer directly beneath, or
 - all** components of layers further beneath.



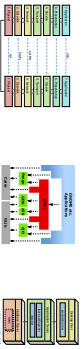
29/16

Example: Three-Tier Architecture

- presentation layer (or tier)**: user interface; presents information obtained from the back-end; the user controls interaction with the back-end; the user is responsible for displaying "user inputs"
- logic layer**: core system functionality; layer is designed without information about the presentation layer; may only read/write data by tier
- data layer**: persistent data storage; hides information about how data is stored; the presentation layer may only read/write information in a form useful for the logic layer



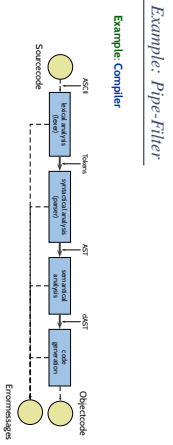
30/16



- **Advantages:**
 - **protocol based:** only neighboring layers are coupled, i.e. components of these layers interact.
 - coupling is low: data usually encapsulated.
 - changes have local effect: only neighboring layers affected.
 - **product based:** distributed implementation often easy.
- **Disadvantages:**
 - performance (as usual) – nowadays often not a problem.

31/16

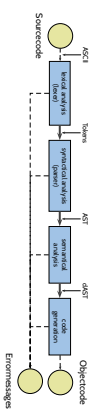
Pipe-Filter



- **Advantages:**
 - if the filters use a common data exchange format, all filters may need changes
 - filters don't use global data, in particular not for variable error conditions
- **Disadvantages:**
 - if the filters use a common data exchange format, all filters may need changes
 - filters don't use global data, in particular not for variable error conditions

32/16

Example Compiler

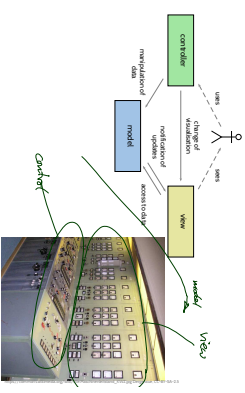


Example UNIX Pipes
`ls -l | grep Search.txt | awk '{ print $5 }'`

- **Advantages:**
 - if the filters use a common data exchange format, all filters may need changes
 - filters don't use global data, in particular not for variable error conditions
- **Disadvantages:**
 - if the filters use a common data exchange format, all filters may need changes
 - filters don't use global data, in particular not for variable error conditions

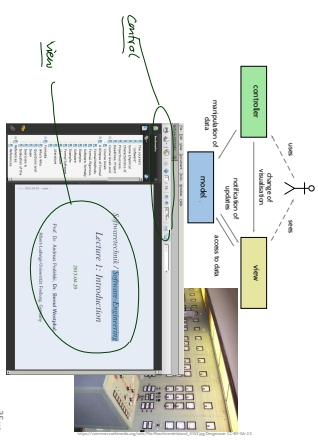
33/16

Example: Model-View-Controller



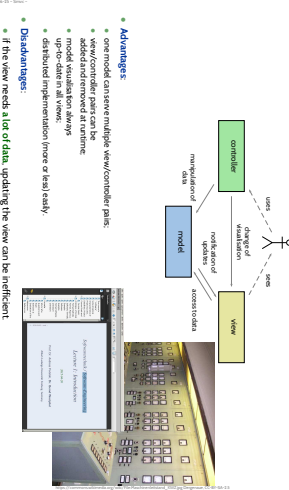
34/16

Example: Model-View-Controller



35/16

Example: Model-View-Controller



35/44

- Advantages:
 - one model can serve multiple view/controller pairs
 - view/controller pairs can be added and removed as runtime
 - control and data are separated
 - update address all views
 - distributed implementation (more or less) easily
- Disadvantages:
 - if the view needs a lot of data, updating the view can be inefficient

Design Patterns

36/44

Design Patterns

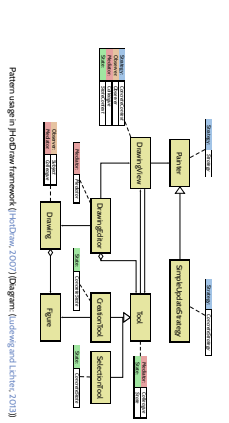
- It is as simple as the name suggests, but on a lower scale
- Often traced back to (Alexander et al., 1977; Alexander, 1979).



Design patterns — are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context. A design pattern names, abstracts, and defines the key aspects of a **reusable object-oriented design**. (Gamma et al., 1993)

37/44

Example: Pattern Usage and Documentation



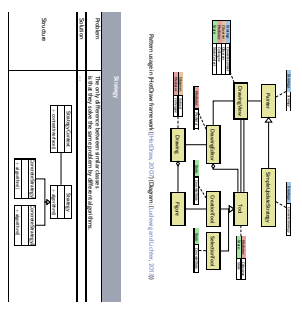
38/44

Example: Strategy

Strategy
<p>Problem</p> <p>The role of different behavior in the class varies, but that they solve the same problem by different algorithms.</p>
<p>Solution</p> <ul style="list-style-type: none"> Have one class Strategy interface with all common operations Use the class Strategy interface to define the family of all operations to be implemented differently Implement each operation in a separate class Use the Strategy interface to define the family of all operations to be implemented differently Use the Strategy interface to define the family of all operations to be implemented differently <p>Structure</p>

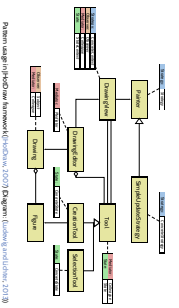
39/44

Example: Pattern Usage and Documentation



40/44

Example: Pattern Usage and Documentation

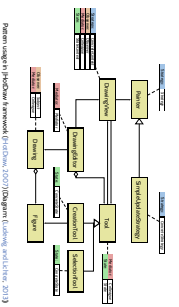


Pattern usage in theOwl framework (Grichkin, 2007) [Diagram, Ludwig and Luban, 2009]

Overview
Problem
Multiple objects need to adjust their state
4. Get object reference from system and do change
Example
If files are added or removed

41.00

Example: Pattern Usage and Documentation

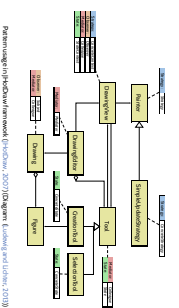


Pattern usage in theOwl framework (Grichkin, 2007) [Diagram, Ludwig and Luban, 2009]

Pattern
Problem
The behavior of an object depends on its internal state
Example
The object of program that open window in turn depends from system, but when the window is not open

41.00

Example: Pattern Usage and Documentation



Pattern usage in theOwl framework (Grichkin, 2007) [Diagram, Ludwig and Luban, 2009]

Pattern
Problem
When the functionality of the system may be broken, complex appearance and design of the next part of the system (new, better, input) and the existing code is not suitable
Example
When the application uses the same data, it should be possible to change the data

41.00

Other Patterns: Singleton and Memoize

Singleton
Problem
Only one data, each by one instance should exist in the system
Example
Print books
Pattern
The state of an object must be retained in a way that allows to re-encapsulation
Undo mechanism
Example
Undo mechanism

42.00

Design Patterns: Discussion

"The development of design patterns is considered to be one of the most important innovations of software engineering in recent years" (Ludwig and Luban, 2009)

(Ludwig and Luban, 2009)

- Advantages:**
 - Reuse the experience of others and employ well-proven solutions
 - Can improve quality, create the changeability or reuse
 - They facilitate documentation of architectures and discussions about architecture.
 - Can be combined in a flexible way.
 - Some codes in particular architecture can correspond to code of multiple patterns
 - Helps teaching software design.
 - Disadvantages:**
 - Using a pattern is not a value in itself.
 - Having too much global data is cannot be justified by "but it's the pattern Singleton".
 - Again, reading is easy, writing need not be.
- Note: Understanding abstract descriptions of design patterns or their user's reading software may be easy - using design patterns to improve software design <https://www.researchgate.net/publication/317113333>

43.00

Libraries and Frameworks

44.00

Libraries and Frameworks

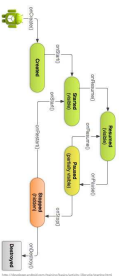
• **Class Library:**
a collection of operations or classes offering generally usable functionality in a re-usable way.

Example:

- **libc** - standard C library (is in particular abstraction layer for operating system functional)
- **glibc** - GNU multi-platform library of Lecture 6
- **libz** - compressed data
- **libxml** - extensible Markup Language (XML) file, provide DOM tree

• **Framework:** class hierarchies which determine a generic solution for similar problems in a particular context

• Example: Android Application Framework



45/00

Libraries and Frameworks

• **Class Library:**
a collection of operations or classes offering generally usable functionality in a re-usable way.

Example:

- **libc** - standard C library (is in particular abstraction layer for operating system functional)
- **glibc** - GNU multi-platform library of Lecture 6
- **libz** - compressed data
- **libxml** - extensible Markup Language (XML) file, provide DOM tree

• **Framework:** class hierarchies which determine a generic solution for similar problems in a particular context

• Example: Android Application Framework

• The difference lies in **low-of-control**:
library modules are called from user code. frameworks call user code.

• **Product line:** parameterised design/code

[All turn indicators are equal, turn indicators in premium cars are more equal.]

45/00

Quality Criteria on Architectures

• **stability**

- architecture design should keep in mind (or formal verification) in mind
- High level of design abstraction/make using significantly easier/provide better, e.g. particular nesting interfaces may improve stability
- e.g. allow injection of user input not only via GUI, or provide particular output for test).

• **changeability / maintainability**

- most systems that are used need to be changed or maintained.
- in particular when requirements change.

• **risk assessment:** part of the system with high probability for changes should be designed and tested using test procedures which support the specific requirements.

• **portability**

- porting adaptations to different platform (OS, hardware, infrastructure)
- systems with long lifetime may need to be adapted to different platforms
- adaptability: how adaptable may design => independent design steps

• **Note:**

- a good design/model is tested all supposed to support the solution.
- it need **not** be a good domain model.

47/00

References

11-10-2024 11:31

48/00

Quality Criteria on Architectures

46/00

References

- Abusurrah, C. (2020). *The Foundations of Software Quality Assurance*. New York: Springer.
- Abusurrah, C., Chikawa, S., and Saitouma, A. (Eds.). *Quality Assurance in Software Development*. London: Springer, 2019.
- Abusurrah, C., Chikawa, S., and Saitouma, A. (Eds.). *Quality Assurance in Software Development*. London: Springer, 2019.
- Abusurrah, C., Chikawa, S., and Saitouma, A. (Eds.). *Quality Assurance in Software Development*. London: Springer, 2019.
- Abusurrah, C., Chikawa, S., and Saitouma, A. (Eds.). *Quality Assurance in Software Development*. London: Springer, 2019.
- Abusurrah, C., Chikawa, S., and Saitouma, A. (Eds.). *Quality Assurance in Software Development*. London: Springer, 2019.
- Abusurrah, C., Chikawa, S., and Saitouma, A. (Eds.). *Quality Assurance in Software Development*. London: Springer, 2019.
- Abusurrah, C., Chikawa, S., and Saitouma, A. (Eds.). *Quality Assurance in Software Development*. London: Springer, 2019.
- Abusurrah, C., Chikawa, S., and Saitouma, A. (Eds.). *Quality Assurance in Software Development*. London: Springer, 2019.
- Abusurrah, C., Chikawa, S., and Saitouma, A. (Eds.). *Quality Assurance in Software Development*. London: Springer, 2019.

49/00