

Vocabulary

system – A collection of components organized to accomplish a specific function or set of functions.

IEEE 1471 (2000)

software system: A set of software units and their relations, if they together serve common purpose.
 This purpose is in general complex, it usually includes, next to providing one (or more) executable program(s), also the organisation, usage, maintenance, and further development.
 (Ludwig and Uicker, 2013)

component—One of the parts that make up a system. A component may be hardware or software and may be subdivided into other components. IEEE 610.12 (1990)

software component—An architectural entity that

- (1) encapsulates a subset of the system's functionality and/or data,
- (2) requests access to that subset via an explicitly defined interface, and
- (3) has explicitly defined dependencies on its required execution context.

[Taylor et al., 2010]

7/48

Vocabulary Cont'd

module—(1) A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from an assembler, compiler, linkage editor, or executive routine.
(2) A logically separable part of a program

IEEE 610.12(1990)

module—A set of operations and data visible from the outside only in so far as explicitly permitted by the programmers. ~ *SW component* (Ludewig and Lichter, 2013)

interface—A boundary across which two independent entities meet and interact or communicate with each other.
(Bachmann et al., 2002)

interface (of component) – The boundary between two communicating components. The interface of a component provides the services of the component to the component's environment and/or requires services needed by the component from the environment.

(Ludewig and Lichey, 2013)

8/4

Even More Vocabulary

design-

- (1) The process of defining the architecture, components, interfaces, and other characteristics of a system or component
- (2) The result of the process in (1).

IEEE 610.12 (1990)

architecture—The fundamental organization of a system embodied in its components and the relationships to each other and to the environment, and the principles guiding its design and evolution.

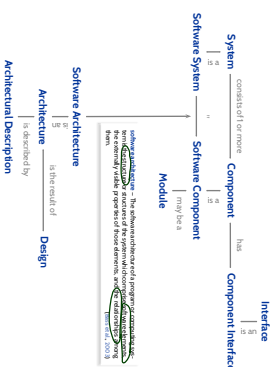
software architecture—The software architecture of a program or computing system is the structure or structures of the system which comprise software elements, the external visible properties of those elements, and the relationships among them.

architectural description – A model – document, product or other artifact – to communicate and record a system's architecture. An architectural description conveys a set of views each of which depicts the system by describing domain concerns.

[Ellis et al., 1996]

9/4

Once Again, Please



10/48

Goals and Relevance of Design

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

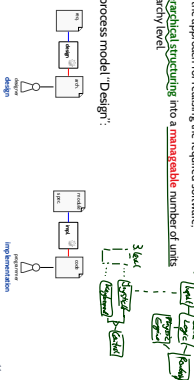
Design...

- (i) structures a system into manageable units (yields software architecture).
 - (ii) determines the approach for realising the required software. 2nd
 - (iii) provides hierarchical structuring into a manageable number of units at each hierarchy level
-
- ```

graph TD
 subgraph System
 direction TB
 CL[Control Logic]
 IO[Input/Output]
 end
 System --- Ellipsis[...]

```

### Oversimplified process model "Design"



114

## Content

- **Vocabulary**
  - bedrock system, component
  - module, interface
  - design, architecture
- **Principles of Good Design**
  - modularity
  - separation of concerns
  - information hiding and data encapsulation
  - abstract data types, object orientation
- **Information hiding / data encapsulation / etc by example**
- **Software Modelling**
  - model
  - views & viewpoints
  - the 4+1 view
  - model-driven software engineering
- **An outlook on UML**

12/4

## Principles of (Architectural) Design

### 3.) Information Hiding

- By now we only discussed the **grouping** of data and operators.  
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's interfaces reveal as little as possible about the module's inner workings, and other modules are prevented from using information about the module that is not in the module's interface specification.  
**IEE 61012 (1990)**
- **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 components may use of this component.
- **Advantages / Goals**:
  - Hidden solutions may be **changed** without other components noticing, as long as the visible behaviour stays the same (e.g. the employed sorting algorithm).
  - IDW: other components cannot (**unintentionally**) depend on details they are not supposed to.
  - Components can be verified / validated in isolation.

### 1.) Modularisation

- **modular decomposition** – The process of breaking a system into components to facilitate design and development; an element of modular programming.  
**IEE 61012 (1990)**
- **modularity** – The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.  
**IEE 61012 (1990)**
- So, **modularity** is a **property** of an architecture.
- Goals of modular decomposition:
  - The **structure** of each module should be **simple and easily comprehensible**.
  - The **implementation** of a module should be **exchangeable**. Information on the implementation of other modules should not be necessary. The other modules should not be affected by implementation changes.
  - Modules should be designed such that **expected changes**.
  - No requirements should be placed on the **visible interface**.
  - Changes should be the result of an act of minor changes.
  - As long as the interface does not change.it should be possible to test old and new versions of a module together.

### 4.) Data Encapsulation

- Similar direction: **data encapsulation** (examples 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.

### 2.) Separation of Concerns

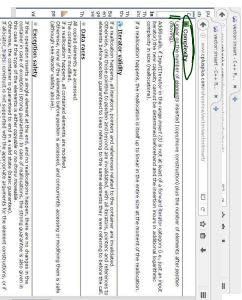
- **Separation of concerns** is a fundamental principle in software engineering
- each component should be **responsible for a particular area of task**.
- components which try to cover different task areas tend to be unnecessarily complex, thus hard to understand and maintain.
- **Criteria for separation/grouping**: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

### 4.) Data Encapsulation

- Similar direction: **data encapsulation** (examples 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** (examples 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.



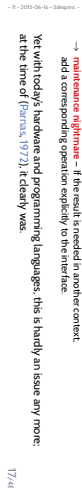
17/48

#### 4.) Data Encapsulation

- **Similar direction: data encapsulation** (examples: later)
  - Do not access data (variables, files, etc.) directly where needed, but encapsulate the data 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.
- **Information hiding and data encapsulation** – When enforced technically (examples: later) – usually **come at the price of some efficiency**.

- **Example 1:** `if (x < 0) { ... }`
  - then calling a operation to provide the value that is an overhead of one operation call.
- **Example 2:** `if (x < 0) { ... }`
  - Knowing how a component works internally, you enable more efficient operation.
- **Example 3:** A sequence of data items is stored as a singly-linked list, accessing the data items in order may be more efficient than decoupling them in reverse order to position `Good middle` page design (like in their documentation) (e.g., C++ standard library).
- **Example 4:** If an implementation stores intermediate results in a certain place, it may be tempting to quickly read that place when the intermediate results are needed in a different context.
  - **maintainance nightmare** – If the result is needed in another context, add a corresponding operation explicitly to the interface.

Yet with today's hardware and programming languages, this is hardly an issue any more: at the time of (Parnas, 1972), it clearly was.



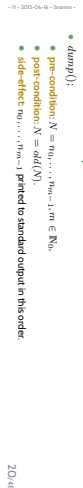
17/48

### Example

- (i) **information hiding and data encapsulation not enforced.**
- (ii)  $\rightarrow$  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.

*Example: Module 'List of Names'*

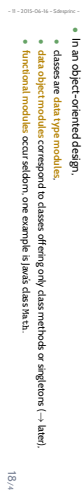
- **Task:** store a list of names in  $N$  of type `list of string`.
- **Operations** (in interface of the module)
  - `insert: string → t`
  - `pre-condition:`  $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, 0 \leq i < m+1, \text{ <len } n_{i+1}$
  - `post-condition:`  $N = n_0, \dots, n_{i-1}, n_i, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, 0 \leq i < m+1, \text{ <len } n_i$  or  $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, i < m, m \in \mathbb{N}_0, 0 \leq i < m, \text{ <len } n_{i+1}$  or  $N = \text{add}(N)$  or  $N = \text{remove}(i, \text{int})$
  - `pre-condition:`  $N = n_0, \dots, n_{i-1}, n_i, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, 0 \leq i < m$
  - `post-condition:`  $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}, i$
  - `get(i, int) → string`
  - `post-condition:`  $N = n_0, \dots, n_{i-1}, n_i, n_{i+1}, \dots, n_{m-1}, m \in \mathbb{N}_0, 0 \leq i < m$
  - `post-condition:`  $N = \text{add}(N, \text{read}(\text{int}))$
  - `dump()`
  - `pre-condition:`  $N = n_0, \dots, n_{m-1}, m \in \mathbb{N}_0$
  - `side-effect:`  $N = n_0, \dots, n_{m-1}$  printed to standard output in this order



20/45

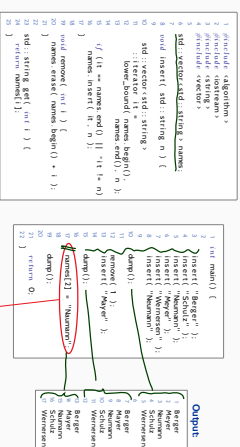
### *A Classification of Modules (Nagl, 1990)*

- **functional modules**
  - group computations which belong together logically
  - don't have "memory" or state that behaviour of *defined* functionally does not depend on previous program evolution
- **Examples** mathematical functions, transformations
- **data object modules**
  - realise encapsulation of data
  - a data module hides kind and structure of data, interface offers operations to manipulate
- **Examples** modules encapsulating global configuration data, databases
- **data type modules**
  - implement a user-defined data type in form of an abstract data type (ADT)
  - allows to create and use as many exemplars of the data type
- **Example** game object
- In an object-oriented design,
  - classes are **data type modules**
  - **data object modules** correspond to classes offering only class methods or *singulars* (– later)
  - **functional modules** occur outside, one example is *laws* class in *hah*.

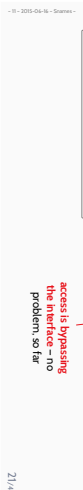


18/4

### *A Possible Implementation: Plain List, no Duplicates*



access is bypassing the interface – no problem, so far



21/4

## Change Interface: Support Duplicate Names

- **Take in addition:** *count(n)* should tell how many *n*'s we have.
- **Operations** (in interface of the module)
  - ***insert***: *insert(x)*
    - **pre-condition**:  $N = n_0, \dots, n_{i-1}, x, \dots, n_{m-1} \in \mathbb{N}_0, \forall 0 \leq j < m: y_{j+1} \leq y_j + 1$
    - **post-condition**:  $N = n_0, \dots, n_{i-1}, x, n_i, \dots, n_{m-1}, \text{count}(n_i) = 1$
  - ***remove***: *remove(i)*
    - **pre-condition**:  $N = n_0, \dots, n_{i-1}, n_i, n_{i+1}, \dots, n_{m-1} \in \mathbb{N}_0, 0 \leq i < m$
    - **post-condition**:  $N = n_0, \dots, n_{i-1}, n_{i+1}, \dots, n_{m-1}$
    - **if** *count(n<sub>i</sub>)* > 1, *N* = *old(N, count(n<sub>i</sub>) - 1)*
- ***get***: *get(i)* : string; and *dump*() :
  - unchanged content

## Data Encapsulation + Information Hiding

[illegible]

### Changed Implementation: Support Duplicates

[illegible]

## Abstract Data Type

## Data Encapsulation + Information Hiding

```

1 #include <string>
2 using namespace std;
3
4 void display()
5 {
6 cout << "Enter a string : ";
7 string str;
8 while (str.empty())
9 {
10 str = "Invalid string!";
11 }
12 }
13
14 int main()
15 {
16 display();
17 string str;
18 while (str.empty())
19 {
20 str = "Invalid string!";
21 }
22 cout << "The string is : " << str << endl;
23 }

```

**Source**

```

1 #include <string>
2 using namespace std;
3
4 void display()
5 {
6 cout << "Enter a string : ";
7 string str;
8 while (str.empty())
9 {
10 str = "Invalid string!";
11 }
12 }
13
14 int main()
15 {
16 display();
17 string str;
18 while (str.empty())
19 {
20 str = "Invalid string!";
21 }
22 cout << "The string is : " << str << endl;
23 }

```

**Output**

```

1 Enter a string :
2
3 Invalid string!
4
5 Enter a string :
6
7 Invalid string!
8
9 Enter a string :
10
11 Invalid string!
12
13 Enter a string :
14
15 Invalid string!
16
17 Enter a string :
18
19 Invalid string!
20
21 Enter a string :
22
23 Invalid string!
24
25 Enter a string :
26
27 Invalid string!
28
29 Enter a string :
30
31 Invalid string!
32
33 Enter a string :
34
35 Invalid string!
36
37 Enter a string :
38
39 Invalid string!
40
41 Enter a string :
42
43 Invalid string!
44
45 Enter a string :
46
47 Invalid string!
48
49 Enter a string :
50
51 Invalid string!
52
53 Enter a string :
54
55 Invalid string!
56
57 Enter a string :
58
59 Invalid string!
60
61 Enter a string :
62
63 Invalid string!
64
65 Enter a string :
66
67 Invalid string!
68
69 Enter a string :
70
71 Invalid string!
72
73 Enter a string :
74
75 Invalid string!
76
77 Enter a string :
78
79 Invalid string!
80
81 Enter a string :
82
83 Invalid string!
84
85 Enter a string :
86
87 Invalid string!
88
89 Enter a string :
90
91 Invalid string!
92
93 Enter a string :
94
95 Invalid string!
96
97 Enter a string :
98
99 Invalid string!
100
101 Enter a string :
102
103 Invalid string!
104
105 Enter a string :
106
107 Invalid string!
108
109 Enter a string :
110
111 Invalid string!
112
113 Enter a string :
114
115 Invalid string!
116
117 Enter a string :
118
119 Invalid string!
120
121 Enter a string :
122
123 Invalid string!
124
125 Enter a string :
126
127 Invalid string!
128
129 Enter a string :
130
131 Invalid string!
132
133 Enter a string :
134
135 Invalid string!
136
137 Enter a string :
138
139 Invalid string!
140
141 Enter a string :
142
143 Invalid string!
144
145 Enter a string :
146
147 Invalid string!
148
149 Enter a string :
150
151 Invalid string!
152
153 Enter a string :
154
155 Invalid string!
156
157 Enter a string :
158
159 Invalid string!
160
161 Enter a string :
162
163 Invalid string!
164
165 Enter a string :
166
167 Invalid string!
168
169 Enter a string :
170
171 Invalid string!
172
173 Enter a string :
174
175 Invalid string!
176
177 Enter a string :
178
179 Invalid string!
180
181 Enter a string :
182
183 Invalid string!
184
185 Enter a string :
186
187 Invalid string!
188
189 Enter a string :
190
191 Invalid string!
192
193 Enter a string :
194
195 Invalid string!
196
197 Enter a string :
198
199 Invalid string!
200
201 Enter a string :
202
203 Invalid string!
204
205 Enter a string :
206
207 Invalid string!
208
209 Enter a string :
210
211 Invalid string!
212
213 Enter a string :
214
215 Invalid string!
216
217 Enter a string :
218
219 Invalid string!
220
221 Enter a string :
222
223 Invalid string!
224
225 Enter a string :
226
227 Invalid string!
228
229 Enter a string :
230
231 Invalid string!
232
233 Enter a string :
234
235 Invalid string!
236
237 Enter a string :
238
239 Invalid string!
240
241 Enter a string :
242
243 Invalid string!
244
245 Enter a string :
246
247 Invalid string!
248
249 Enter a string :
250
251 Invalid string!
252
253 Enter a string :
254
255 Invalid string!
256
257 Enter a string :
258
259 Invalid string!
260
261 Enter a string :
262
263 Invalid string!
264
265 Enter a string :
266
267 Invalid string!
268
269 Enter a string :
270
271 Invalid string!
272
273 Enter a string :
274
275 Invalid string!
276
277 Enter a string :
278
279 Invalid string!
280
281 Enter a string :
282
283 Invalid string!
284
285 Enter a string :
286
287 Invalid string!
288
289 Enter a string :
290
291 Invalid string!
292
293 Enter a string :
294
295 Invalid string!
296
297 Enter a string :
298
299 Invalid string!
300
301 Enter a string :
302
303 Invalid string!
304
305 Enter a string :
306
307 Invalid string!
308
309 Enter a string :
310
311 Invalid string!
312
313 Enter a string :
314
315 Invalid string!
316
317 Enter a string :
318
319 Invalid string!
320
321 Enter a string :
322
323 Invalid string!
324
325 Enter a string :
326
327 Invalid string!
328
329 Enter a string :
330
331 Invalid string!
332
333 Enter a string :
334
335 Invalid string!
336
337 Enter a string :
338
339 Invalid string!
340
341 Enter a string :
342
343 Invalid string!
344
345 Enter a string :
346
347 Invalid string!
348
349 Enter a string :
350
351 Invalid string!
352
353 Enter a string :
354
355 Invalid string!
356
357 Enter a string :
358
359 Invalid string!
360
361 Enter a string :
362
363 Invalid string!
364
365 Enter a string :
366
367 Invalid string!
368
369 Enter a string :
370
371 Invalid string!
372
373 Enter a string :
374
375 Invalid string!
376
377 Enter a string :
378
379 Invalid string!
380
381 Enter a string :
382
383 Invalid string!
384
385 Enter a string :
386
387 Invalid string!
388
389 Enter a string :
390
391 Invalid string!
392
393 Enter a string :
394
395 Invalid string!
396
397 Enter a string :
398
399 Invalid string!
400
401 Enter a string :
402
403 Invalid string!
404
405 Enter a string :
406
407 Invalid string!
408
409 Enter a string :
410
411 Invalid string!
412
413 Enter a string :
414
415 Invalid string!
416
417 Enter a string :
418
419 Invalid string!
420
421 Enter a string :
422
423 Invalid string!
424
425 Enter a string :
426
427 Invalid string!
428
429 Enter a string :
430
431 Invalid string!
432
433 Enter a string :
434
435 Invalid string!
436
437 Enter a string :
438
439 Invalid string!
440
441 Enter a string :
442
443 Invalid string!
444
445 Enter a string :
446
447 Invalid string!
448
449 Enter a string :
450
451 Invalid string!
452
453 Enter a string :
454
455 Invalid string!
456
457 Enter a string :
458
459 Invalid string!
460
461 Enter a string :
462
463 Invalid string!
464
465 Enter a string :
466
467 Invalid string!
468
469 Enter a string :
470
471 Invalid string!
472
473 Enter a string :
474
475 Invalid string!
476
477 Enter a string :
478
479 Invalid string!
480
481 Enter a string :
482
483 Invalid string!
484
485 Enter a string :
486
487 Invalid string!
488
489 Enter a string :
490
491 Invalid string!
492
493 Enter a string :
494
495 Invalid string!
496
497 Enter a string :
498
499 Invalid string!
500
501 Enter a string :
502
503 Invalid string!
504
505 Enter a string :
506
507 Invalid string!
508
509 Enter a string :
510
511 Invalid string!
512
513 Enter a string :
514
515 Invalid string!
516
517 Enter a string :
518
519 Invalid string!
520
521 Enter a string :
522
523 Invalid string!
524
525 Enter a string :
526
527 Invalid string!
528
529 Enter a string :
530
531 Invalid string!
532
533 Enter a string :
534
535 Invalid string!
536
537 Enter a string :
538
539 Invalid string!
540
541 Enter a string :
542
543 Invalid string!
544
545 Enter a string :
546
547 Invalid string!
548
549 Enter a string :
550
551 Invalid string!
552
553 Enter a string :
554
555 Invalid string!
556
557 Enter a string :
558
559 Invalid string!
560
561 Enter a string :
562
563 Invalid string!
564
565 Enter a string :
566
567 Invalid string!
568
569 Enter a string :
570
571 Invalid string!
572
573 Enter a string :
574
575 Invalid string!
576
577 Enter a string :
578
579 Invalid string!
580
581 Enter a string :
582
583 Invalid string!
584
585 Enter a string :
586
587 Invalid string!
588
589 Enter a string :
590
591 Invalid string!
592
593 Enter a string :
594
595 Invalid string!
596
597 Enter a string :
598
599 Invalid string!
600
601 Enter a string :
602
603 Invalid string!
604
605 Enter a string :
606
607 Invalid string!
608
609 Enter a string :
610
611 Invalid string!
612
613 Enter a string :
614
615 Invalid string!
616
617 Enter a string :
618
619 Invalid string!
620
621 Enter a string :
622
623 Invalid string!
624
625 Enter a string :
626
627 Invalid string!
628
629 Enter a string :
630
631 Invalid string!
632
633 Enter a string :
634
635 Invalid string!
636
637 Enter a string :
638
639 Invalid string!
640
641 Enter a string :
642
643 Invalid string!
644
645 Enter a string :
646
647 Invalid string!
648
649 Enter a string :
650
651 Invalid string!
652
653 Enter a string :
654
655 Invalid string!
656
657 Enter a string :
658
659 Invalid string!
660
661 Enter a string :
```

## Abstract Data Type

```

1 // pointer to struct "Name"
2 typedef struct "Name"
3 {
4 Name *next;
5 }
6
7 void dump(Name *name)
8 {
9 if (name == NULL)
10 return;
11 printf("Name: %s\n", name->name);
12 if (name->next != NULL)
13 dump(name->next);
14 }
15
16 void insert(Name *name, int i, string n)
17 {
18 if (i == 0)
19 return;
20 if (i < 0)
21 return;
22 if (i > 10)
23 return;
24 if (i < 0 || i > 10)
25 return;
26 if (i < 0 || i > 10)
27 return;
28 if (i < 0 || i > 10)
29 return;
30 if (i < 0 || i > 10)
31 return;
32 if (i < 0 || i > 10)
33 return;
34 if (i < 0 || i > 10)
35 return;
36 if (i < 0 || i > 10)
37 return;
38 if (i < 0 || i > 10)
39 return;
40 if (i < 0 || i > 10)
41 return;
42 if (i < 0 || i > 10)
43 return;
44 if (i < 0 || i > 10)
45 return;
46 if (i < 0 || i > 10)
47 return;
48 if (i < 0 || i > 10)
49 return;
50 if (i < 0 || i > 10)
51 return;
52 if (i < 0 || i > 10)
53 return;
54 if (i < 0 || i > 10)
55 return;
56 if (i < 0 || i > 10)
57 return;
58 if (i < 0 || i > 10)
59 return;
60 if (i < 0 || i > 10)
61 return;
62 if (i < 0 || i > 10)
63 return;
64 if (i < 0 || i > 10)
65 return;
66 if (i < 0 || i > 10)
67 return;
68 if (i < 0 || i > 10)
69 return;
70 if (i < 0 || i > 10)
71 return;
72 if (i < 0 || i > 10)
73 return;
74 if (i < 0 || i > 10)
75 return;
76 if (i < 0 || i > 10)
77 return;
78 if (i < 0 || i > 10)
79 return;
80 if (i < 0 || i > 10)
81 return;
82 if (i < 0 || i > 10)
83 return;
84 if (i < 0 || i > 10)
85 return;
86 if (i < 0 || i > 10)
87 return;
88 if (i < 0 || i > 10)
89 return;
90 if (i < 0 || i > 10)
91 return;
92 if (i < 0 || i > 10)
93 return;
94 if (i < 0 || i > 10)
95 return;
96 if (i < 0 || i > 10)
97 return;
98 if (i < 0 || i > 10)
99 return;
100 if (i < 0 || i > 10)
101 return;
102 if (i < 0 || i > 10)
103 return;
104 if (i < 0 || i > 10)
105 return;
106 if (i < 0 || i > 10)
107 return;
108 if (i < 0 || i > 10)
109 return;
110 if (i < 0 || i > 10)
111 return;
112 if (i < 0 || i > 10)
113 return;
114 if (i < 0 || i > 10)
115 return;
116 if (i < 0 || i > 10)
117 return;
118 if (i < 0 || i > 10)
119 return;
120 if (i < 0 || i > 10)
121 return;
122 if (i < 0 || i > 10)
123 return;
124 if (i < 0 || i > 10)
125 return;
126 if (i < 0 || i > 10)
127 return;
128 if (i < 0 || i > 10)
129 return;
130 if (i < 0 || i > 10)
131 return;
132 if (i < 0 || i > 10)
133 return;
134 if (i < 0 || i > 10)
135 return;
136 if (i < 0 || i > 10)
137 return;
138 if (i < 0 || i > 10)
139 return;
140 if (i < 0 || i > 10)
141 return;
142 if (i < 0 || i > 10)
143 return;
144 if (i < 0 || i > 10)
145 return;
146 if (i < 0 || i > 10)
147 return;
148 if (i < 0 || i > 10)
149 return;
150 if (i < 0 || i > 10)
151 return;
152 if (i < 0 || i > 10)
153 return;
154 if (i < 0 || i > 10)
155 return;
156 if (i < 0 || i > 10)
157 return;
158 if (i < 0 || i > 10)
159 return;
160 if (i < 0 || i > 10)
161 return;
162 if (i < 0 || i > 10)
163 return;
164 if (i < 0 || i > 10)
165 return;
166 if (i < 0 || i > 10)
167 return;
168 if (i < 0 || i > 10)
169 return;
170 if (i < 0 || i > 10)
171 return;
172 if (i < 0 || i > 10)
173 return;
174 if (i < 0 || i > 10)
175 return;
176 if (i < 0 || i > 10)
177 return;
178 if (i < 0 || i > 10)
179 return;
180 if (i < 0 || i > 10)
181 return;
182 if (i < 0 || i > 10)
183 return;
184 if (i < 0 || i > 10)
185 return;
186 if (i < 0 || i > 10)
187 return;
188 if (i < 0 || i > 10)
189 return;
190 if (i < 0 || i > 10)
191 return;
192 if (i < 0 || i > 10)
193 return;
194 if (i < 0 || i > 10)
195 return;
196 if (i < 0 || i > 10)
197 return;
198 if (i < 0 || i > 10)
199 return;
200 if (i < 0 || i > 10)
201 return;
202 if (i < 0 || i > 10)
203 return;
204 if (i < 0 || i > 10)
205 return;
206 if (i < 0 || i > 10)
207 return;
208 if (i < 0 || i > 10)
209 return;
210 if (i < 0 || i > 10)
211 return;
212 if (i < 0 || i > 10)
213 return;
214 if (i < 0 || i > 10)
215 return;
216 if (i < 0 || i > 10)
217 return;
218 if (i < 0 || i > 10)
219 return;
220 if (i < 0 || i > 10)
221 return;
222 if (i < 0 || i > 10)
223 return;
224 if (i < 0 || i > 10)
225 return;
226 if (i < 0 || i > 10)
227 return;
228 if (i < 0 || i > 10)
229 return;
230 if (i < 0 || i > 10)
231 return;
232 if (i < 0 || i > 10)
233 return;
234 if (i < 0 || i > 10)
235 return;
236 if (i < 0 || i > 10)
237 return;
238 if (i < 0 || i > 10)
239 return;
240 if (i < 0 || i > 10)
241 return;
242 if (i < 0 || i > 10)
243 return;
244 if (i < 0 || i > 10)
245 return;
246 if (i < 0 || i > 10)
247 return;
248 if (i < 0 || i > 10)
249 return;
250 if (i < 0 || i > 10)
251 return;
252 if (i < 0 || i > 10)
253 return;
254 if (i < 0 || i > 10)
255 return;
256 if (i < 0 || i > 10)
257 return;
258 if (i < 0 || i > 10)
259 return;
260 if (i < 0 || i > 10)
261 return;
262 if (i < 0 || i > 10)
263 return;
264 if (i < 0 || i > 10)
265 return;
266 if (i < 0 || i > 10)
267 return;
268 if (i < 0 || i > 10)
269 return;
270 if (i < 0 || i > 10)
271 return;
272 if (i < 0 || i > 10)
273 return;
274 if (i < 0 || i > 10)
275 return;
276 if (i < 0 || i > 10)
277 return;
278 if (i < 0 || i > 10)
279 return;
280 if (i < 0 || i > 10)
281 return;
282 if (i < 0 || i > 10)
283 return;
284 if (i < 0 || i > 10)
285 return;
286 if (i < 0 || i > 10)
287 return;
288 if (i < 0 || i > 10)
289 return;
290 if (i < 0 || i > 10)
291 return;
292 if (i < 0 || i > 10)
293 return;
294 if (i < 0 || i > 10)
295 return;
296 if (i < 0 || i >
```

## Abstract Data Type

[illegible]

## Object Oriented

```

1 println! "vector =";
2 println! "string =";
3
4 // vector
5 let v: Vec<Vec<u8>>> =
6 vec![vec![b'a', b'b', b'c'],
7 vec![b'd', b'e', b'f'],
8 vec![b'g', b'h', b'i']];
9 println! "v: {:#?}";
10
11 // string
12 let s: String = "hello world!";
13 println! "s: {:#?}";
14
15 // map
16 let m: HashMap<Vec<u8>, Vec<u8>> =
17 HashMap::new();
18 m.insert(vec![b'a', b'b', b'c'], vec![b'd', b'e', b'f']);
19 m.insert(vec![b'g', b'h', b'i'], vec![b'j', b'k', b'l']);
20 println! "m: {:#?}";
21
22 // set
23 let s: HashSet<Vec<u8>> =
24 HashSet::new();
25 s.insert(vec![b'a', b'b', b'c']);
26 s.insert(vec![b'd', b'e', b'f']);
27 s.insert(vec![b'g', b'h', b'i']);
28 println! "s: {:#?}";
29
30 // tuple
31 let t: (Vec<u8>, Vec<u8>) =
32 (vec![b'a', b'b', b'c'], vec![b'd', b'e', b'f']);
33 println! "t: {:#?}";
34
35 // array
36 let a: [Vec<u8>; 3] =
37 [vec![b'a', b'b', b'c'],
38 vec![b'd', b'e', b'f'],
39 vec![b'g', b'h', b'i']];
40 println! "a: {:#?}";
41
42 // slice
43 let s: &[Vec<u8>] =
44 &[vec![b'a', b'b', b'c'],
45 vec![b'd', b'e', b'f'],
46 vec![b'g', b'h', b'i']];
47 println! "s: {:#?}";
48
49 // iterator
50 let v: Vec<Vec<u8>> =
51 vec![vec![b'a', b'b', b'c'],
52 vec![b'd', b'e', b'f'],
53 vec![b'g', b'h', b'i']];
54 println! "v: {:#?}";
55
56 // range
57 let r: Range<Vec<u8>> =
58 vec![b'a', b'b', b'c']..vec![b'd', b'e', b'f'];
59 println! "r: {:#?}";
60
61 // lazy
62 let l: LazyVec<Vec<u8>> =
63 LazyVec::new(|| vec![b'a', b'b', b'c'],
64 || vec![b'd', b'e', b'f'],
65 || vec![b'g', b'h', b'i']);
66 println! "l: {:#?}";
67
68 // lazy
69 let l: LazyVec<Vec<u8>> =
70 LazyVec::new(|| vec![b'a', b'b', b'c'],
71 || vec![b'd', b'e', b'f'],
72 || vec![b'g', b'h', b'i']);
73 println! "l: {:#?}";
74
75 // lazy
76 let l: LazyVec<Vec<u8>> =
77 LazyVec::new(|| vec![b'a', b'b', b'c'],
78 || vec![b'd', b'e', b'f'],
79 || vec![b'g', b'h', b'i']);
80 println! "l: {:#?}";
81
82 // lazy
83 let l: LazyVec<Vec<u8>> =
84 LazyVec::new(|| vec![b'a', b'b', b'c'],
85 || vec![b'd', b'e', b'f'],
86 || vec![b'g', b'h', b'i']);
87 println! "l: {:#?}";
88
89 // lazy
90 let l: LazyVec<Vec<u8>> =
91 LazyVec::new(|| vec![b'a', b'b', b'c'],
92 || vec![b'd', b'e', b'f'],
93 || vec![b'g', b'h', b'i']);
94 println! "l: {:#?}";
95
96 // lazy
97 let l: LazyVec<Vec<u8>> =
98 LazyVec::new(|| vec![b'a', b'b', b'c'],
99 || vec![b'd', b'e', b'f'],
100 || vec![b'g', b'h', b'i']);
101 println! "l: {:#?}";
102
103 // lazy
104 let l: LazyVec<Vec<u8>> =
105 LazyVec::new(|| vec![b'a', b'b', b'c'],
106 || vec![b'd', b'e', b'f'],
107 || vec![b'g', b'h', b'i']);
108 println! "l: {:#?}";
109
110 // lazy
111 let l: LazyVec<Vec<u8>> =
112 LazyVec::new(|| vec![b'a', b'b', b'c'],
113 || vec![b'd', b'e', b'f'],
114 || vec![b'g', b'h', b'i']);
115 println! "l: {:#?}";
116
117 // lazy
118 let l: LazyVec<Vec<u8>> =
119 LazyVec::new(|| vec![b'a', b'b', b'c'],
120 || vec![b'd', b'e', b'f'],
121 || vec![b'g', b'h', b'i']);
122 println! "l: {:#?}";
123
124 // lazy
125 let l: LazyVec<Vec<u8>> =
126 LazyVec::new(|| vec![b'a', b'b', b'c'],
127 || vec![b'd', b'e', b'f'],
128 || vec![b'g', b'h', b'i']);
129 println! "l: {:#?}";
130
131 // lazy
132 let l: LazyVec<Vec<u8>> =
133 LazyVec::new(|| vec![b'a', b'b', b'c'],
134 || vec![b'd', b'e', b'f'],
135 || vec![b'g', b'h', b'i']);
136 println! "l: {:#?}";
137
138 // lazy
139 let l: LazyVec<Vec<u8>> =
140 LazyVec::new(|| vec![b'a', b'b', b'c'],
141 || vec![b'd', b'e', b'f'],
142 || vec![b'g', b'h', b'i']);
143 println! "l: {:#?}";
144
145 // lazy
146 let l: LazyVec<Vec<u8>> =
147 LazyVec::new(|| vec![b'a', b'b', b'c'],
148 || vec![b'd', b'e', b'f'],
149 || vec![b'g', b'h', b'i']);
150 println! "l: {:#?}";
151
152 // lazy
153 let l: LazyVec<Vec<u8>> =
154 LazyVec::new(|| vec![b'a', b'b', b'c'],
155 || vec![b'd', b'e', b'f'],
156 || vec![b'g', b'h', b'i']);
157 println! "l: {:#?}";
158
159 // lazy
160 let l: LazyVec<Vec<u8>> =
161 LazyVec::new(|| vec![b'a', b'b', b'c'],
162 || vec![b'd', b'e', b'f'],
163 || vec![b'g', b'h', b'i']);
164 println! "l: {:#?}";
165
166 // lazy
167 let l: LazyVec<Vec<u8>> =
168 LazyVec::new(|| vec![b'a', b'b', b'c'],
169 || vec![b'd', b'e', b'f'],
170 || vec![b'g', b'h', b'i']);
171 println! "l: {:#?}";
172
173 // lazy
174 let l: LazyVec<Vec<u8>> =
175 LazyVec::new(|| vec![b'a', b'b', b'c'],
176 || vec![b'd', b'e', b'f'],
177 || vec![b'g', b'h', b'i']);
178 println! "l: {:#?}";
179
180 // lazy
181 let l: LazyVec<Vec<u8>> =
182 LazyVec::new(|| vec![b'a', b'b', b'c'],
183 || vec![b'd', b'e', b'f'],
184 || vec![b'g', b'h', b'i']);
185 println! "l: {:#?}";
186
187 // lazy
188 let l: LazyVec<Vec<u8>> =
189 LazyVec::new(|| vec![b'a', b'b', b'c'],
190 || vec![b'd', b'e', b'f'],
191 || vec![b'g', b'h', b'i']);
192 println! "l: {:#?}";
193
194 // lazy
195 let l: LazyVec<Vec<u8>> =
196 LazyVec::new(|| vec![b'a', b'b', b'c'],
197 || vec![b'd', b'e', b'f'],
198 || vec![b'g', b'h', b'i']);
199 println! "l: {:#?}";
200
201 // lazy
202 let l: LazyVec<Vec<u8>> =
203 LazyVec::new(|| vec![b'a', b'b', b'c'],
204 || vec![b'd', b'e', b'f'],
205 || vec![b'g', b'h', b'i']);
206 println! "l: {:#?}";
207
208 // lazy
209 let l: LazyVec<Vec<u8>> =
210 LazyVec::new(|| vec![b'a', b'b', b'c'],
211 || vec![b'd', b'e', b'f'],
212 || vec![b'g', b'h', b'i']);
213 println! "l: {:#?}";
214
215 // lazy
216 let l: LazyVec<Vec<u8>> =
217 LazyVec::new(|| vec![b'a', b'b', b'c'],
218 || vec![b'd', b'e', b'f'],
219 || vec![b'g', b'h', b'i']);
220 println! "l: {:#?}";
221
222 // lazy
223 let l: LazyVec<Vec<u8>> =
224 LazyVec::new(|| vec![b'a', b'b', b'c'],
225 || vec![b'd', b'e', b'f'],
226 || vec![b'g', b'h', b'i']);
227 println! "l: {:#?}";
228
229 // lazy
230 let l: LazyVec<Vec<u8>> =
231 LazyVec::new(|| vec![b'a', b'b', b'c'],
232 || vec![b'd', b'e', b'f'],
233 || vec![b'g', b'h', b'i']);
234 println! "l: {:#?}";
235
236 // lazy
237 let l: LazyVec<Vec<u8>> =
238 LazyVec::new(|| vec![b'a', b'b', b'c'],
239 || vec![b'd', b'e', b'f'],
240 || vec![b'g', b'h', b'i']);
241 println! "l: {:#?}";
242
243 // lazy
244 let l: LazyVec<Vec<u8>> =
245 LazyVec::new(|| vec![b'a', b'b', b'c'],
246 || vec![b'd', b'e', b'f'],
247 || vec![b'g', b'h', b'i']);
248 println! "l: {:#?}";
249
250 // lazy
251 let l: LazyVec<Vec<u8>> =
252 LazyVec::new(|| vec![b'a', b'b', b'c'],
253 || vec![b'd', b'e', b'f'],
254 || vec![b'g', b'h', b'i']);
255 println! "l: {:#?}";
256
257 // lazy
258 let l: LazyVec<Vec<u8>> =
259 LazyVec::new(|| vec![b'a', b'b', b'c'],
260 || vec![b'd', b'e', b'f'],
261 || vec![b'g', b'h', b'i']);
262 println! "l: {:#?}";
263
264 // lazy
265 let l: LazyVec<Vec<u8>> =
266 LazyVec::new(|| vec![b'a', b'b', b'c'],
267 || vec![b'd', b'e', b'f'],
268 || vec![b'g', b'h', b'i']);
269 println! "l: {:#?}";
270
271 // lazy
272 let l: LazyVec<Vec<u8>> =
273 LazyVec::new(|| vec![b'a', b'b', b'c'],
274 || vec![b'd', b'e', b'f'],
275 || vec![b'g', b'h', b'i']);
276 println! "l: {:#?}";
277
278 // lazy
279 let l: LazyVec<Vec<u8>> =
280 LazyVec::new(|| vec![b'a', b'b', b'c'],
281 || vec![b'd', b'e', b'f'],
282 || vec![b'g', b'h', b'i']);
283 println! "l: {:#?}";
284
285 // lazy
286 let l: LazyVec<Vec<u8>> =
287 LazyVec::new(|| vec![b'a', b'b
```

## Object Oriented

[illegible]

## Object Oriented + Data Encapsulation / Information Hiding

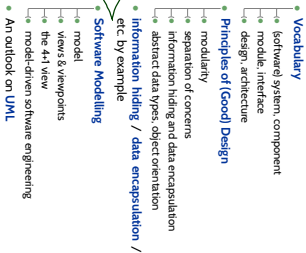
[illegible]

## Object Oriented + Data Encapsulation / Information Hiding

|     |                                       |  |
|-----|---------------------------------------|--|
| 1   | private <vector>                      |  |
| 2   | private: <string>                     |  |
| 3   | public:                               |  |
| 4   | vector() {}                           |  |
| 5   | vector( const Names & )               |  |
| 6   | private:                              |  |
| 7   | std::vector< string > content;        |  |
| 8   | std::vector< string > &string::names; |  |
| 9   | public:                               |  |
| 10  | Names() {}                            |  |
| 11  | Names( const Names & )                |  |
| 12  | void display();                       |  |
| 13  | void insert( std::string n );         |  |
| 14  | void remove( int i );                 |  |
| 15  | std::string get( int i );             |  |
| 16  | };                                    |  |
| 17  | };                                    |  |
| 18  |                                       |  |
| 19  |                                       |  |
| 20  |                                       |  |
| 21  |                                       |  |
| 22  |                                       |  |
| 23  |                                       |  |
| 24  |                                       |  |
| 25  |                                       |  |
| 26  |                                       |  |
| 27  |                                       |  |
| 28  |                                       |  |
| 29  |                                       |  |
| 30  |                                       |  |
| 31  |                                       |  |
| 32  |                                       |  |
| 33  |                                       |  |
| 34  |                                       |  |
| 35  |                                       |  |
| 36  |                                       |  |
| 37  |                                       |  |
| 38  |                                       |  |
| 39  |                                       |  |
| 40  |                                       |  |
| 41  |                                       |  |
| 42  |                                       |  |
| 43  |                                       |  |
| 44  |                                       |  |
| 45  |                                       |  |
| 46  |                                       |  |
| 47  |                                       |  |
| 48  |                                       |  |
| 49  |                                       |  |
| 50  |                                       |  |
| 51  |                                       |  |
| 52  |                                       |  |
| 53  |                                       |  |
| 54  |                                       |  |
| 55  |                                       |  |
| 56  |                                       |  |
| 57  |                                       |  |
| 58  |                                       |  |
| 59  |                                       |  |
| 60  |                                       |  |
| 61  |                                       |  |
| 62  |                                       |  |
| 63  |                                       |  |
| 64  |                                       |  |
| 65  |                                       |  |
| 66  |                                       |  |
| 67  |                                       |  |
| 68  |                                       |  |
| 69  |                                       |  |
| 70  |                                       |  |
| 71  |                                       |  |
| 72  |                                       |  |
| 73  |                                       |  |
| 74  |                                       |  |
| 75  |                                       |  |
| 76  |                                       |  |
| 77  |                                       |  |
| 78  |                                       |  |
| 79  |                                       |  |
| 80  |                                       |  |
| 81  |                                       |  |
| 82  |                                       |  |
| 83  |                                       |  |
| 84  |                                       |  |
| 85  |                                       |  |
| 86  |                                       |  |
| 87  |                                       |  |
| 88  |                                       |  |
| 89  |                                       |  |
| 90  |                                       |  |
| 91  |                                       |  |
| 92  |                                       |  |
| 93  |                                       |  |
| 94  |                                       |  |
| 95  |                                       |  |
| 96  |                                       |  |
| 97  |                                       |  |
| 98  |                                       |  |
| 99  |                                       |  |
| 100 |                                       |  |
| 101 |                                       |  |
| 102 |                                       |  |
| 103 |                                       |  |
| 104 |                                       |  |
| 105 |                                       |  |
| 106 |                                       |  |
| 107 |                                       |  |
| 108 |                                       |  |
| 109 |                                       |  |
| 110 |                                       |  |
| 111 |                                       |  |
| 112 |                                       |  |
| 113 |                                       |  |
| 114 |                                       |  |
| 115 |                                       |  |
| 116 |                                       |  |
| 117 |                                       |  |
| 118 |                                       |  |
| 119 |                                       |  |
| 120 |                                       |  |
| 121 |                                       |  |
| 122 |                                       |  |
| 123 |                                       |  |
| 124 |                                       |  |
| 125 |                                       |  |
| 126 |                                       |  |
| 127 |                                       |  |
| 128 |                                       |  |
| 129 |                                       |  |
| 130 |                                       |  |
| 131 |                                       |  |
| 132 |                                       |  |
| 133 |                                       |  |
| 134 |                                       |  |
| 135 |                                       |  |
| 136 |                                       |  |
| 137 |                                       |  |
| 138 |                                       |  |
| 139 |                                       |  |
| 140 |                                       |  |
| 141 |                                       |  |
| 142 |                                       |  |
| 143 |                                       |  |
| 144 |                                       |  |
| 145 |                                       |  |
| 146 |                                       |  |
| 147 |                                       |  |
| 148 |                                       |  |
| 149 |                                       |  |
| 150 |                                       |  |
| 151 |                                       |  |
| 152 |                                       |  |
| 153 |                                       |  |
| 154 |                                       |  |
| 155 |                                       |  |
| 156 |                                       |  |
| 157 |                                       |  |
| 158 |                                       |  |
| 159 |                                       |  |
| 160 |                                       |  |
| 161 |                                       |  |
| 162 |                                       |  |
| 163 |                                       |  |
| 164 |                                       |  |
| 165 |                                       |  |
| 166 |                                       |  |
| 167 |                                       |  |
| 168 |                                       |  |
| 169 |                                       |  |
| 170 |                                       |  |
| 171 |                                       |  |
| 172 |                                       |  |
| 173 |                                       |  |
| 174 |                                       |  |
| 175 |                                       |  |
| 176 |                                       |  |
| 177 |                                       |  |
| 178 |                                       |  |
| 179 |                                       |  |
| 180 |                                       |  |
| 181 |                                       |  |
| 182 |                                       |  |
| 183 |                                       |  |
| 184 |                                       |  |
| 185 |                                       |  |

*“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) **abstract data types.**
- (vi) **object oriented without** information hiding and data encapsulation.
- (vii) **object oriented with** information hiding and data encapsulation.

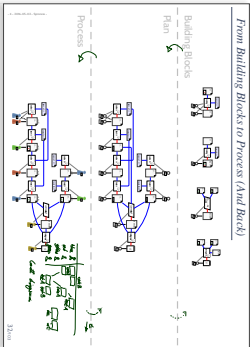


Definition: [Fek] A model is an abstract, formal, mathematical representation or description of structure or behaviour of a (software) system.

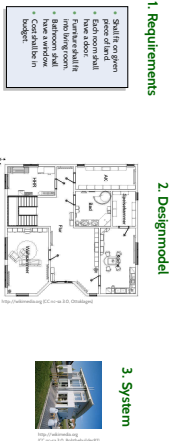
Software Modelling

Definition: [Fek, 2008, 425]  
A model is a concrete or mental image (Abbild.) of something or a concrete or mental archetype (Vorbild) for something. Three properties are constituent:  
(i) the image attribute (Abbildungsmerkmal), i.e. there is an entity (called original) whose image or archetype the model is;  
(ii) the reduction attribute (Verkürzungsmerkmal), i.e. only those attributes of the original that are relevant in the modelling context are represented;  
(iii) the pragmatic attribute, i.e. the model is built in a specific context for a specific purpose.

Example: Process Model

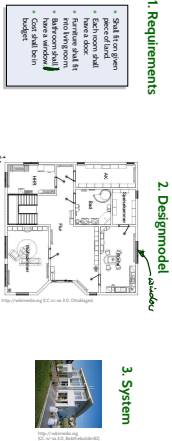


Example: Design-Models in Construction Engineering



**Observation (1):** Floorplan abstracts from certain system properties, e.g. ...  
• kind, number and placement of bricks,  
• wall decoration  
• water pipes/wiring, and  
• subsystem details (e.g. window style).  
→ architects can efficiently work on appropriate level of abstraction

Example: Design-Models in Construction Engineering



**Observation (2):** Floorplan preserves/determines certain system properties, e.g. ...  
• house and room extensions (to scale),  
• placement of subsystems (such as windows).  
→ find design errors before building the system (e.g. bathroom windows)

## A Better Analogy is Maybe Regional Planning



© 2005-06-06 - Samskri

34/48

## Views and Viewpoints

**view** – A representation of a whole system from the perspective of a related set of concerns.

IEEE M71 (2000)

**viewpoint** – A specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis.

IEEE M71 (2000)

- A **perspective** is determined by **concerns** and **information needs**:
- team leader**, e.g., needs to know which team is working on what component,
- operator**, e.g., needs to know which component is running on which host,
- developer**, e.g., needs to know interfaces of other components,
- etc.

© 2005-06-06 - Samskri

35/48

## An Early Proposal: The 4+1 View (Kivinen, 1995)



(Lubbing and Lohrer, 2013):

**system view**: how is the system under development organized by its environment, with which other systems (including user) does it interact, how?

**static view** (~ **developer view**): components of the architecture, their interfaces and their data, design of components of development, test, etc. onto them.

**dynamic view** (~ **process view**): how and how often do the components of the system work together at runtime.

**deployment view** (~ **physical view**): how are component instances mapped onto infrastructure and hardware units.

© 2005-06-06 - Samskri

36/48

## Process and Physical View

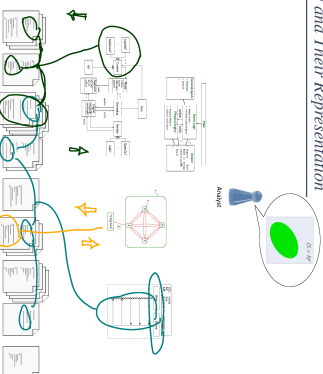


- Example: modern cars**
- large number of electronic control units (ECUs) spread all over the car,
  - which part of the overall software is running on which ECU?
  - which function is used when? Event triggered, time triggered, continuous, etc.?
- For e.g. a simple **smartphone app**, process and physical view may be trivial or determined by the employed framework (→ **libel**) – so no need for (extensive) particular documentation.

© 2005-06-06 - Samskri

37/48

## Views and Their Representation



© 2005-06-06 - Samskri

38/48

## Structure vs. Behaviour

- Form of the states** in  $\Sigma$  (also actions  $A$ ):
- structure of  $S$**
- Computation paths**  $\pi$  of  $S$ :
- behaviour of  $S$**

Definition. **Behaviour** is a finite sequence  $\sigma$  of  $A$ -labeled paths of the form

$$\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_n, \sigma_{n+1}, \dots$$

where

- $\sigma_i \in S$ ,  $i \in \mathbb{N}$ ,  $R, K$  called **states** (or **configurations**) and
- $\sigma_i \in A$ ,  $i \in \mathbb{N}$ ,  $R, K$  called **actions** (or **events**).

The (possibly partial) function  $\pi : S \rightarrow \{S\}$  is called **interpretation of  $S$** .

- (Harel, 1997) proposes to distinguish **constructive** and **reflective** descriptions of behaviour:
- constructive**:
    - "constructs for description contain information needed in executing the model or in translating it into executable code."
    - **how things are computed**.
  - reflective (or assertive)**:
    - "description used to derive and present views of the model, statically or during execution, or to set constraints on behaviour in preparation for verification."
    - **what should (or should not) be computed**.
- Note:** No sharp boundaries! (would be too easy...)

© 2005-06-06 - Samskri

39/48

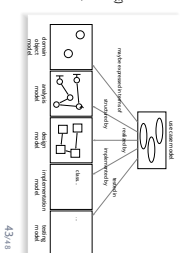




# A Brief History of the Unified Modelling Language (UML)

- Boxes/lines and finite automata are used to visualise software **for ages**
- **1970's, Software Crisis™**
  - Ideas learnt from engineering disciplines to handle growing complexity
- Modelling languages: Flowcharts, Nass-Shneiderman, Entity-Relation Diagrams
- Mid 1980's: **Sanchechans** (Fueet, 1987), **Sanchechans™** (Vareel et al., 1990)
- Early 1990's: advent of **Object-Oriented Analysis/Design/Programming**
  - Inflation of notations and methods, most prominent:
- **Object-Modelling Technique (OMT)** (Rumbaugh et al., 1990)
- **Booch Method and Notation** (Booch, 1993)
- **Object-Oriented Software Engineering (OOSE)** (Jacobson et al., 1993)

Each "persuasion" selling books, tools, seminars...



43/48

# A Brief History of the Unified Modelling Language (UML)

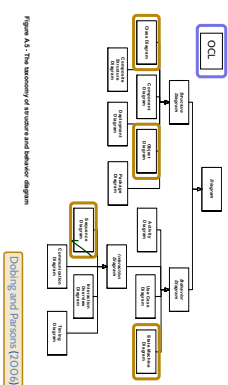
- Boxes/lines and finite automata are used to visualise software **for ages**
- **1970's, Software Crisis™**
  - Ideas learnt from engineering disciplines to handle growing complexity
- Modelling languages: Flowcharts, Nass-Shneiderman, Entity-Relation Diagrams
- Mid 1980's: **Sanchechans** (Fueet, 1987), **Sanchechans™** (Vareel et al., 1990)
- Early 1990's: advent of **Object-Oriented Analysis/Design/Programming**
  - Inflation of notations and methods, most prominent:
- **Object-Modelling Technique (OMT)** (Rumbaugh et al., 1990)
- **Booch Method and Notation** (Booch, 1993)
- **Object-Oriented Software Engineering (OOSE)** (Jacobson et al., 1993)

Each "persuasion" selling books, tools, seminars...

- Late 1990's: joint effort of "the three empires": **UML 0.2** and **1.x**
- Standards published by **Object Management Group (OMG)**, "international open membership, not-for-profit **computer industry consortium**" Much criticised for lack of formality
- Since 2005: **UML 2.x**, split into infra- and superstructure documents.

43/48

# UML Overview (OMG, 2007, 684)



44/48

# Topic Area Architecture & Design: Content

- **VL1**
  - **Introduction and Vocabulary**
    - (i) modularity
    - (ii) separation of concerns
    - (iii) information hiding and data encapsulation
    - (iv) abstract data types, object orientation
  - **Software Modelling**
    - (i) views and viewpoints, the 4+1 view
    - (ii) model-driven/-based software engineering
    - (iii) Unified Modelling Language (UML)
    - (iv) **modelling structure**
      - a) (simplified) class diagrams
      - b) (simplified) object diagrams
      - c) (simplified) object constraint logic (OCL)
    - (v) **modelling behaviour**
      - a) communicating finite automata
      - b) state transition diagrams
      - c) block state machines
      - d) an outlook on hierarchical state-machines
- **VL2**
- **VL3**
- **VL4**
- **Design Patterns**

45/48

# Tell Them What You've Told Them...

- **Design structures a system into manageable units.**
- **Principles of (Good) Design:**
  - modularity, separation of concerns,
  - information hiding / data encapsulation
- **Models: a concrete or mental image or archetype with**
  - image attribute,
  - reduction attribute,
  - pragmatic attribute,
  - here: **abstract, formal, mathematical** description.
- **Software Modelling:** views and viewpoints, e.g. 4+1
- **Model-driven Software Engineering**
- **Unified Modelling Language:**
  - a family of **modelling languages**.

46/48

# References

47/48

## References

- [illegible]