Overview of Design Patterns

CS480 Software Engineering
http://cs480.yusun.io
February 20, 2015

Yu Sun, Ph.D.
http://yusun.io
yusun@cpp.edu

CAL POLY POMONA

Overview

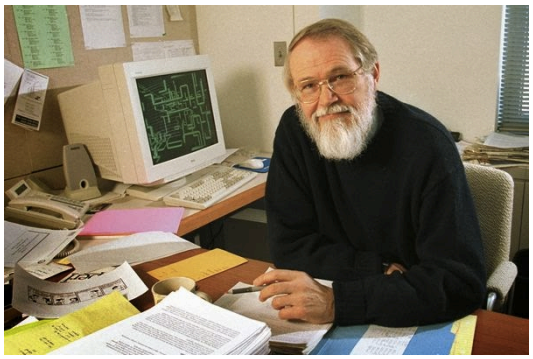
- Motivate the importance of design experience & leveraging recurring design structure in becoming a master software developer

Becoming a Master







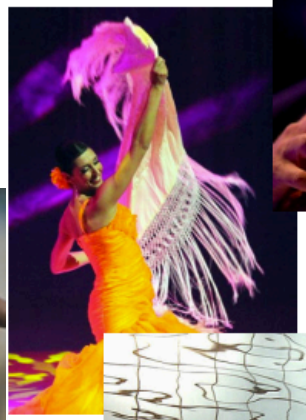


The top 10 greatest programmers in the world of all time

http://www.thecrazyprogrammer.com/2014/02/the-top-10-greatest-programmers-in-the-world-of-all-time.html

Becoming a Master

- Experts perform differently than beginners
  - Unlike novices, professional athletes, musicians & dancers move fluidly & effortlessly, without focusing on each individual movement

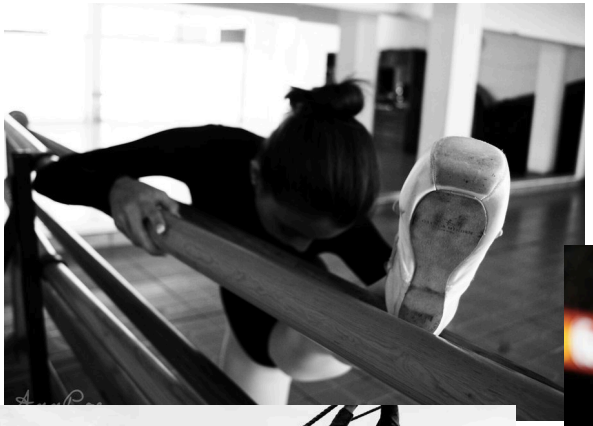


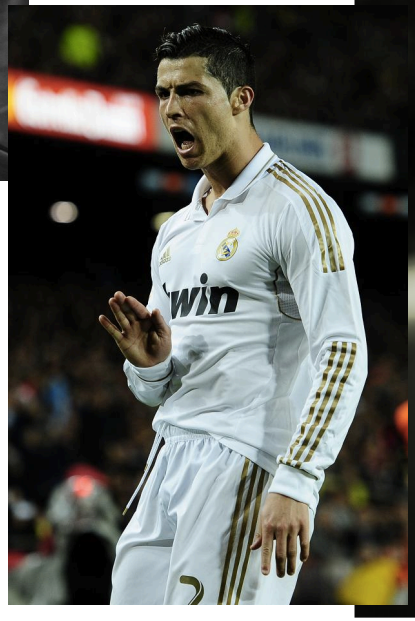
Becoming a Master

- Experts perform differently than beginners

Becoming a Master

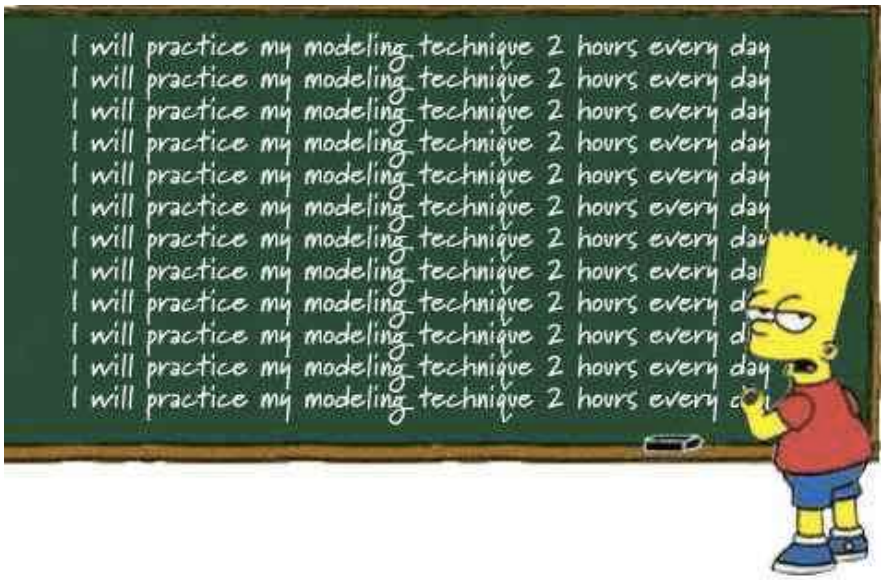
- When watching experts perform, it's easy to forget how much effort they've put into reaching high levels of achievement





Becoming a Master

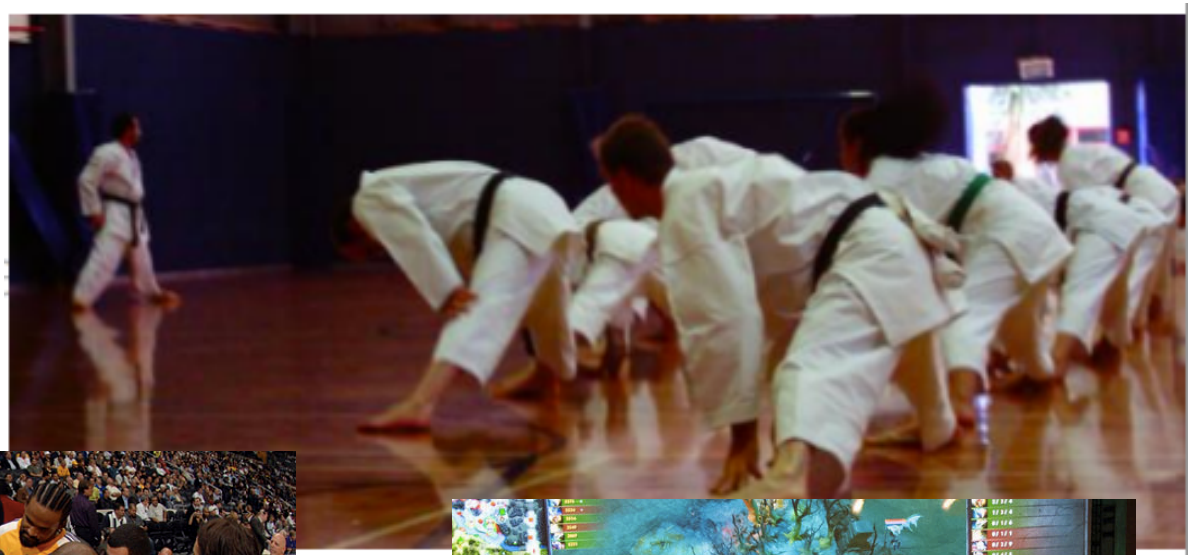
- Continuous repetition & practice are crucial to success



Ted Talk: **The Skill of Self Confidence**
Dr. Ivan Joseph

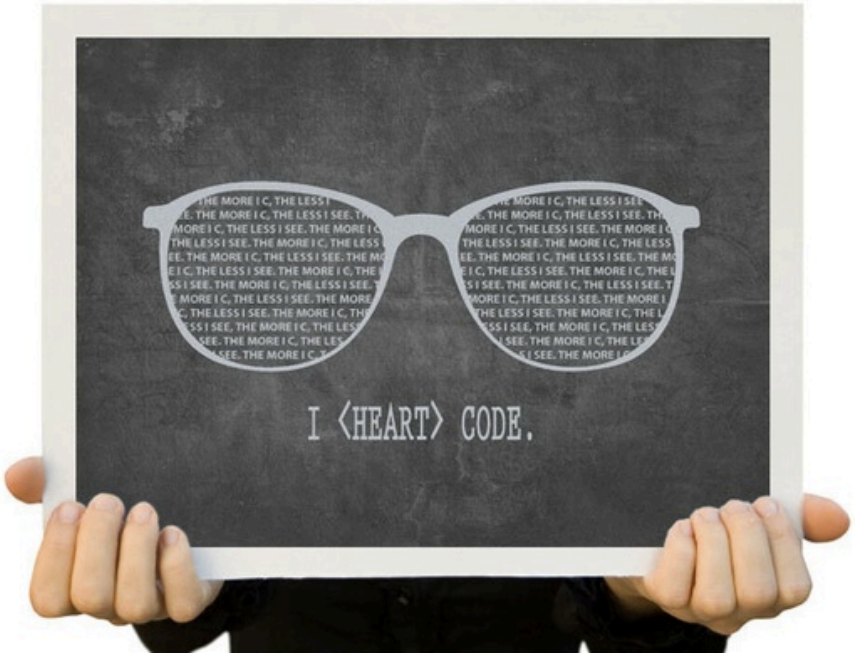
Becoming a Master

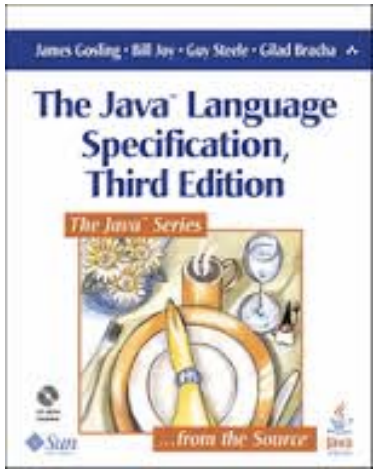
- Mentoring from other experts is also essential to becoming a master

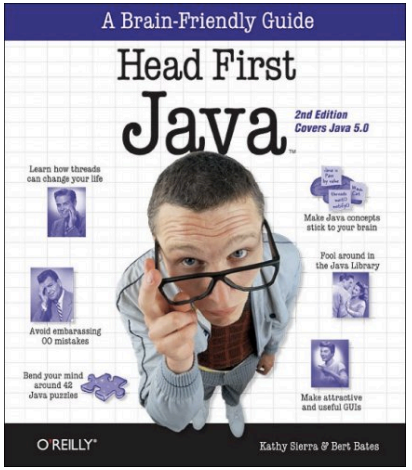


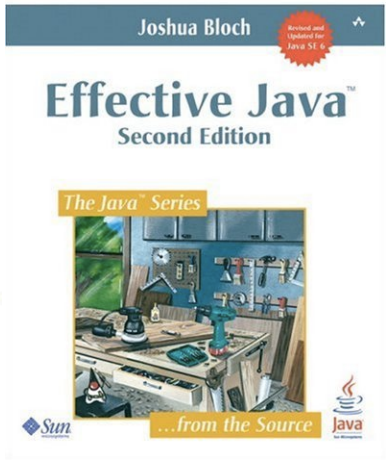
Becoming a Master Software Developer

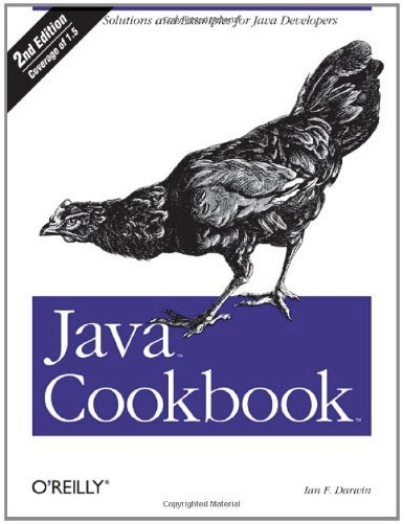
- Knowledge of programming languages is necessary, but not sufficient









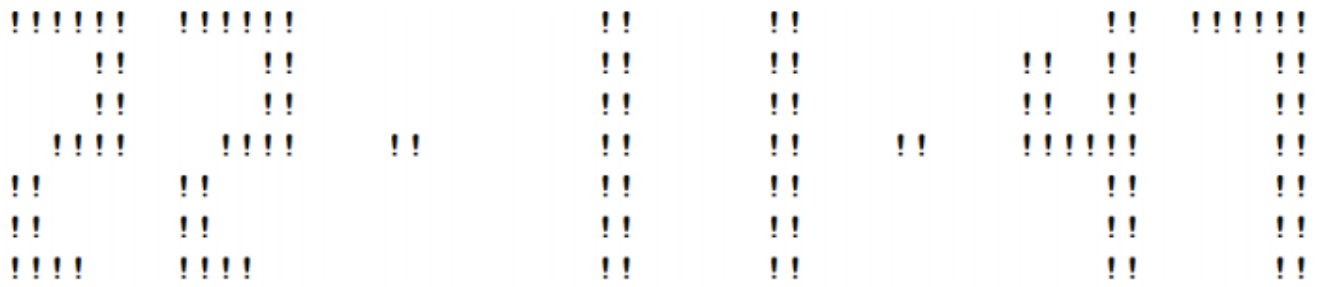


Becoming a Master Software Developer

- Knowledge of programming languages is necessary, but not sufficient
  - e.g., "Best one-liner" from 2006 "Obfuscated C Code" contest

```
main(_){_^448&&main(-~_);putchar(--_%64?32|-~7[
__TIME__-_/8%8]["">'txiZ^(~z?"-48]>>";;;====~$::199"
[_*2&8|_/64]/(_&2?1:8)%8&1:10);}
```

  - This program pints out the time when it was compiled!



- http://www.ioccc.org/

Becoming a Master Software Developer

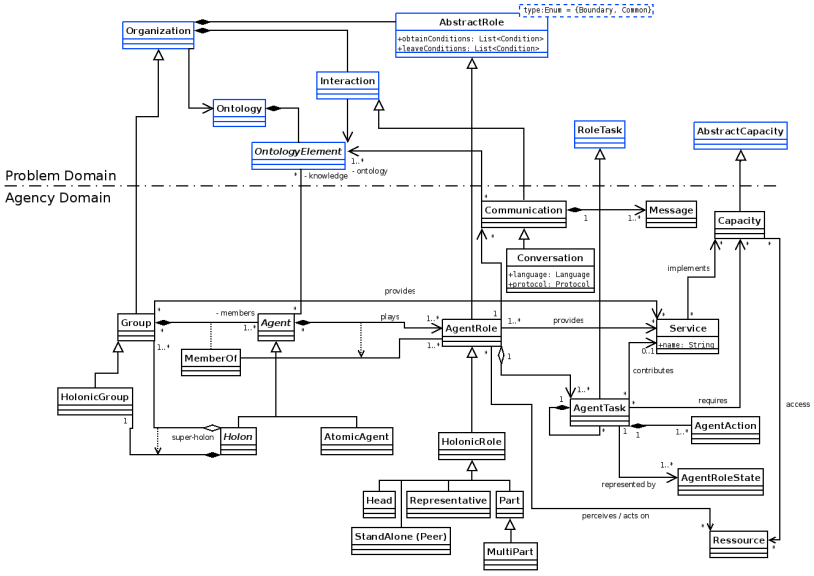
Becoming a Master Software Developer

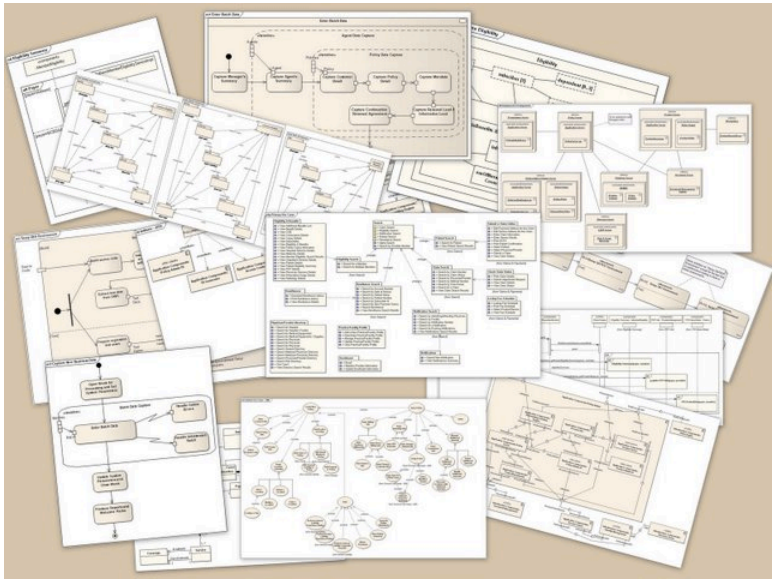
```
#include <ncurses.h>/*****************************************************/
              int                 m[256                   ] [         256   ],a
 ,b   ;;;   ;;;   WINDOW*w;   char*l=""    "\176qxl"    "q"   "q"    "k"   "w\
xm"   "x"   "t"          "j"          "v"          "u"          "n"         ,Q[
 ]=   "Z"   "pt!ftd`"   "qdc!`eu"   "dq!$c!nnwf"/**   ***   */"t\040\t";c(
int   u ,          int          v){                  v?m   [u]         [v-
 1]   |=2,m[u][v-1] &   48?W][v-1   ] &   15]]):0:0;u?m[u   -1][v]|=1   ,m[
 u-                 1][   v]&          48?                  W-1   ][v         ]&
15]   ]):0:0;v<   255   ?m[   u][v+1]|=8,m[u][v+1]&   48?   W][   v+1]&15]]
):0          :0;          u <                  255   ?m[   u+1         ][v   ]|=
4,m[u+1][   v]&48?W+1][v]&15]]):0:0;W][   v]&   15]   ]);}cu(char*q){   return
 *q                 ?cu   (q+          1)&          1?q   [0]                 ++:
q[0   ]--   :1;   }d(   int   u ,   int/**/v,   int/**/x,   int   y){   int
Y=y   -v,   X=x          -u;   int          S,s   ;Y<          0?Y   =-Y   ,s,
s=-   1:(   s=1);X<0?X=-X,S   =-1   :(S=   1);   Y<<=   1;X<<=1;   if(X>Y){
int   f=Y                -(X   >>1   );;                  while(u!=         x){
f>=   0?v+=s,f-=X:0;u   +=S   ;f+=   Y;m[u][v]|=32;mvwaddch(w,v   ,u,   m[u
 ][                v]&   64?   60:          46)          ;if          (m[   u][
v]&16){c(u,v);;   ;;;   ;;;   return;}}   }else{int   f=X   -(Y>>1);;   while
 (v   !=y          ){f   >=0          ?u   +=S,                  f-=         Y:0
 ;v   +=s   ;f+=X;m[u][v]|=   32;mvwaddch(w,v   ,u,m[u][v]&64?60:46);if(m[u
 ][                v]&          16)   {c(   u,v                           );
  ;   return;;;}}}}Z(   int/**/a,   int   b){   }e(   int/**/y,int/**/   x){
int               i ;          for          (i=          a;i                 <=a
+S;i++)d(y,x,i,b),d(y,x,i,b+L);for(i=b;i<=b+L;i++)d(y,x,a,i),d(y,x,a+   S,i
 );                ;;;          ;;;          ;;;          ;;;         ;
  mvwaddch(w,x,y,64);   ;;;   ;;;   ;;;   prefresh(   w,b,a,0,0   ,L-   1,S-1
);}          main(          int                  V ,   char                *C[
  ]   ){FILE*f=   fopen(V==1?"arachnid.c"/**/   :C[   1],"r");int/**/x,y,c,
v=0          ;;;      initscr          ();                  Z(Z         (raw
 ()   ,Z(   curs_set(0),Z(1   ,noecho()))),keypad(   stdscr,TRUE));w   =newpad
  (   300,   300                ) ;   for          (x=   255   ; x   >=0   ;x--
  )   for   (y=   255   ;y>=0;y--   )m[   x][   y]=   0;x=y=0;refresh( );while
  (   (c=                         fgetc (f)   )+1)                    {if(
0||c==10||   x==   256){x=0;y++;if(y==256   )break;;}   else{m[x][y]=(c   ==
'~'   ?64   : c   ==32          ?0:          16)   ;;x                   ++;
     }}for(x=0   ;x<   256;x++)m   [x][0]=16   ,m[   x][   255]=16;for(y=0
;y<          256   ; y          ++)   m[0          ][y   ] =                  16,
m[255][y]   =16   ;a=b=c=0;   x=y   =1;   do{v++;mvwaddch   (w,   y,x   ,m[
x][          y]&          32?   m[x                  ][y   ] &   16?
 0|   acs_map[l[m[x][y]&15]]:46 :   32);c==0163&&!(m[x][y+1]&16)?y++:   0;c
 ==   119          &&!          (m[                        x][
 y-   1]&   16)   ?y--:0;;c   ==97   &&!(m[x-1][y]&16)?x--:0;c==100&&!(m[x+1
 ][   y]&   16)          ? x   ++:0                ;if(                      c==
 3-   1+1   ){endwin(   );;   return(0)   ;}x   -a<5?a>S-   5?a-=S-5:(a=0):
0;x               -a>          S-5?a<255   -S*          2?a                 +=S
-5:(a=256-S):0;   y-b<5?b>L-5?b-=L-5:(b   =0)   :0;   y-b>L-5?b<255-L   *2?
b+=                            L-5   :(b                                  =256
-L)   :0;e(x,y);if(m[x][y]&64)break;}while((c=getch())!=-1);endwin();cu(Q);
printf(Q,y);}
```

Becoming a Master Software Developer

Becoming a Master Software Developer

- Software methods emphasize design notations, such as UML
  - Fine for specification & documentation
  - e.g., omits mundane implementation details & focuses on relationships between key design entities



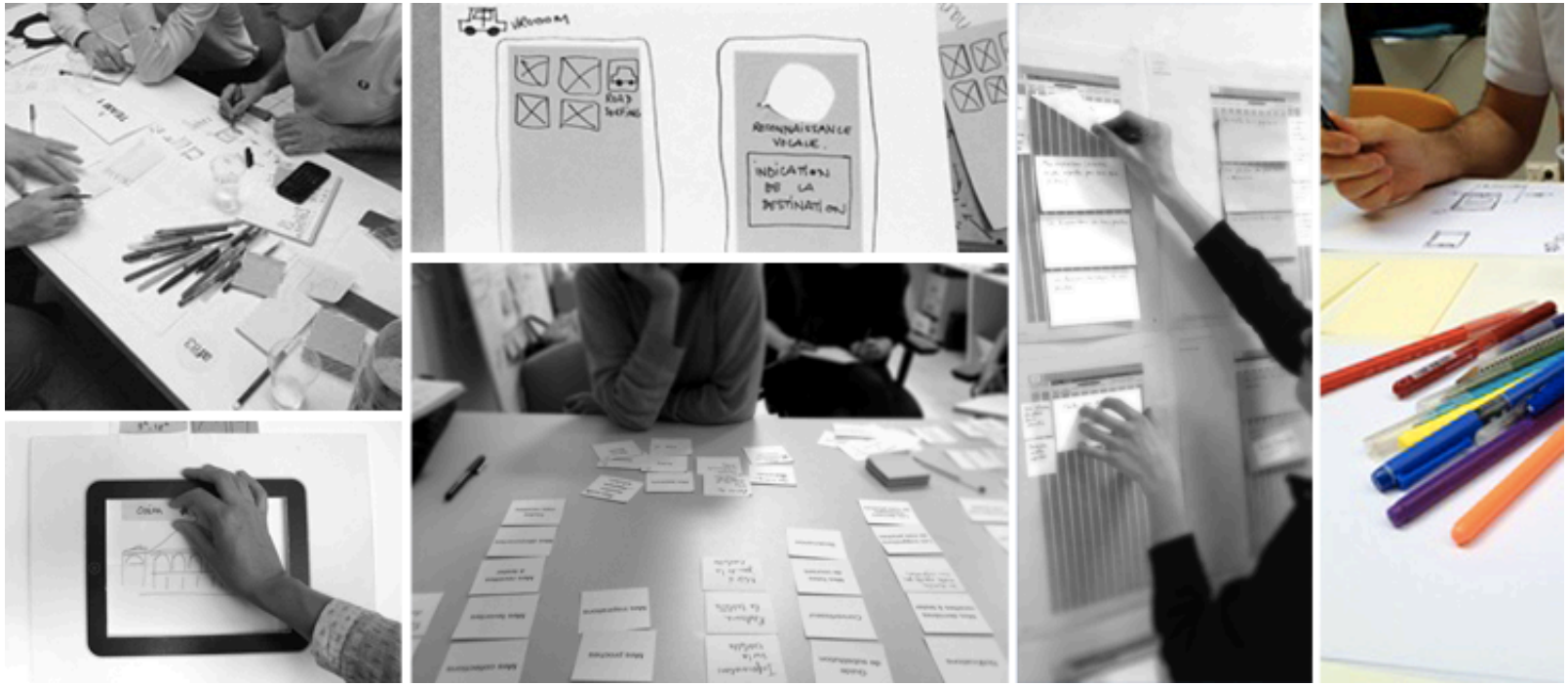


Becoming a Master Software Developer

- But good software design is more than drawing diagrams
  - Good draftsmen/artists are not necessarily good architects!

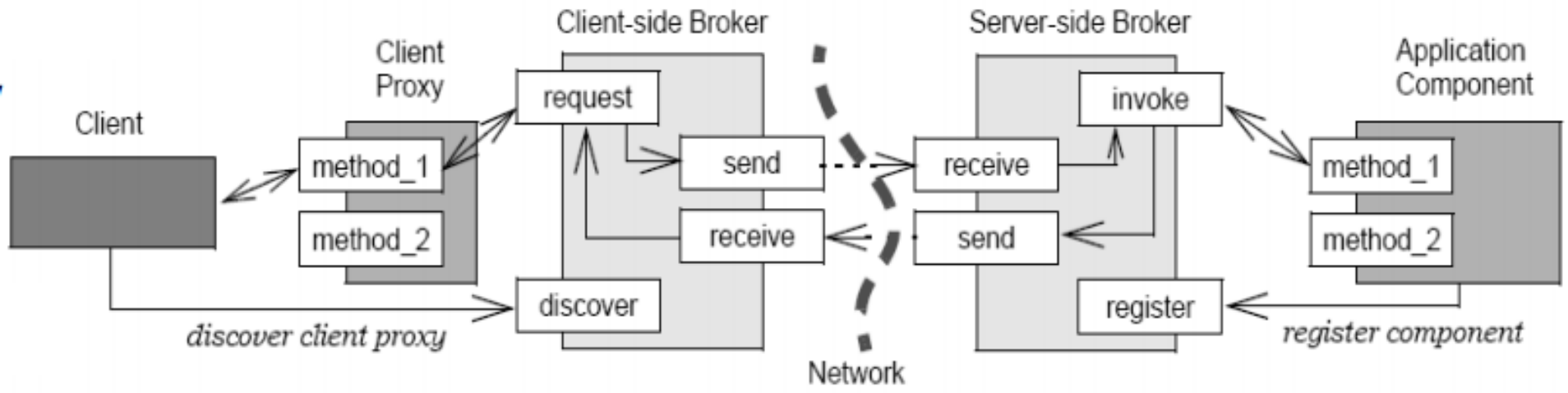
Becoming a Master Software Developer

- Bottom-line: Master software developers rely on design experiences



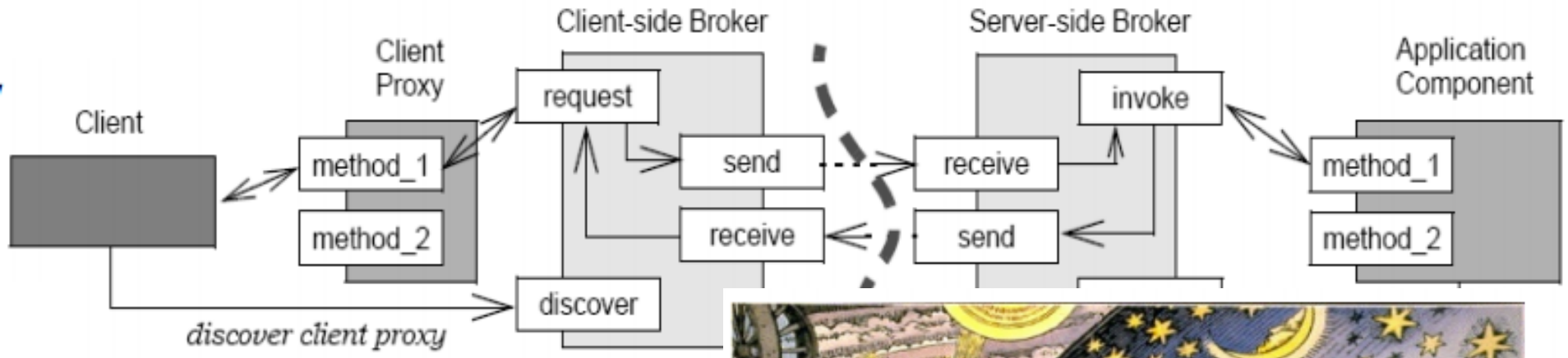
Where should design experience reside?

- Well-designed software exhibits recurring structures & behaviors that promote
  - Abstraction
  - Flexibility
  - Reuse
  - Quality
  - Modularity



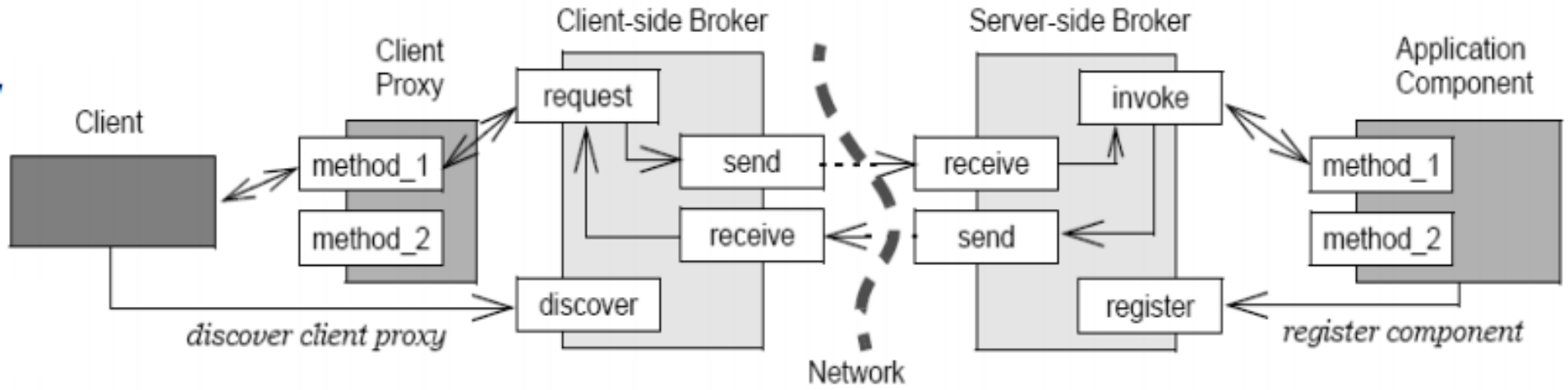
Where should design experience reside?

- Well-designed software exhibits recurring structures & behaviors that promote

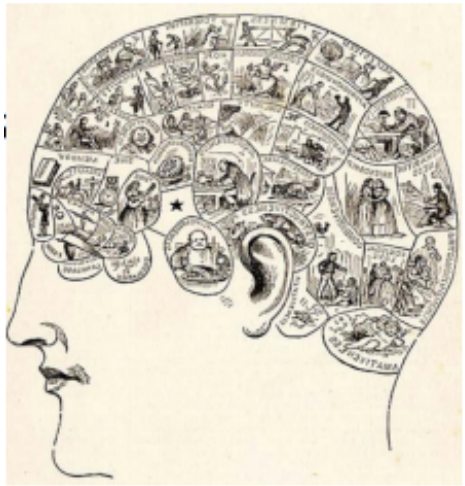


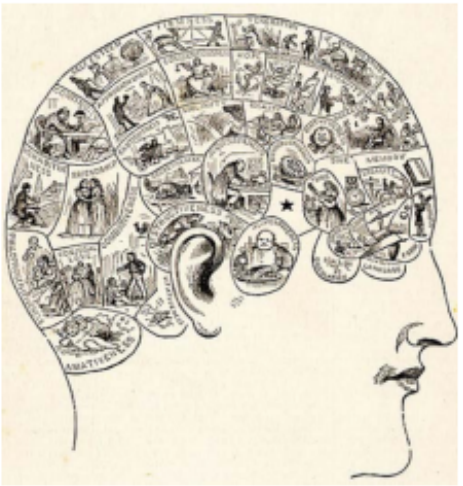
- Therein lies valuable design knowledge

Where should design experience reside?

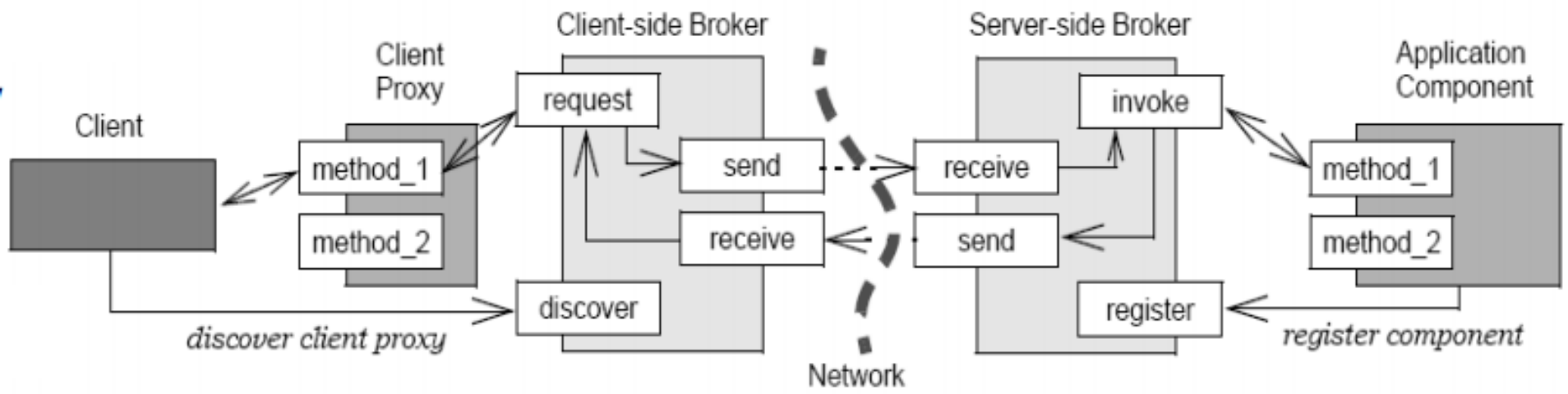


- Unfortunately, this design knowledge is typically located in:
  - The heads of the experts





Where should design experience reside?



- Unfortunately, this design knowledge is typically located in:
  - The bowels of the source code

```
public class KeyGeneratorImpl extends Service {
    private Set<UUID> keys = new HashSet<UUID>();
    private final KeyGenerator.Stub binder = new KeyGenerator.Stub() {
            public void setCallback (final KeyGeneratorCallback callback) {
                UUID id;
                synchronized (keys) {
                    do { id = UUID.randomUUID(); } while (keys.contains(id));
                    keys.add(id);
                }
                final String key = id.toString();
                try {
                    Log.d(getClass().getName(), "sending key" + key);
                    callback.sendKey(key);
                } catch (RemoteException e) { e.printStackTrace(); }
            }
    };
    public IBinder onBind(Intent intent) { return this.binder; }
}
```

Where should design experience reside?

- Unfortunately, this design knowledge is typically located in:
  - The heads of the experts
  - The bowels of the source code
- Both locations are fraught with danger!

Where should design experience reside?

- What we need is a means of extracting, documenting, conveying, applying, & preserving this design knowledge without undue time, effort, & risk!

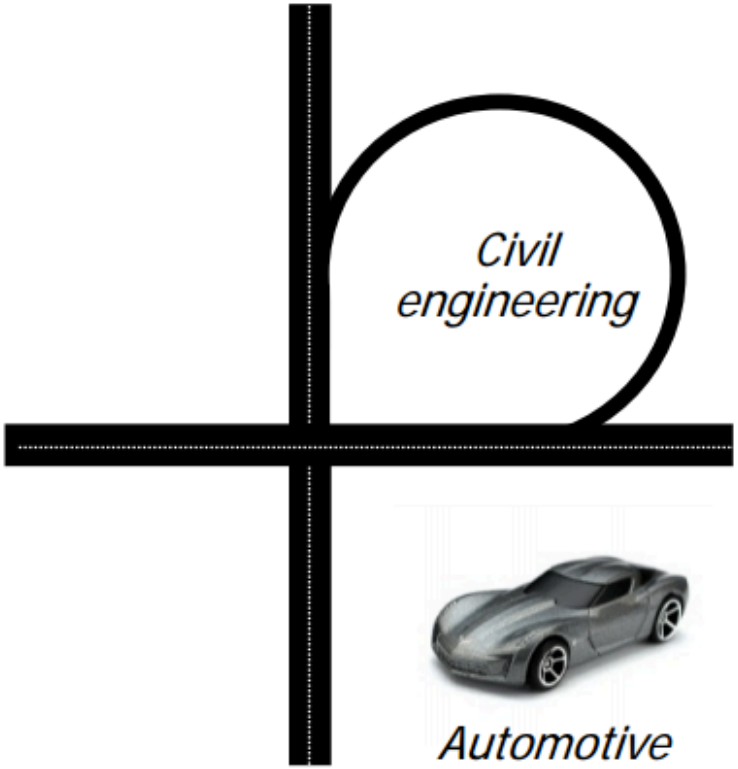
Key to Mastery: Knowledge of Software Patterns

- Describe a **solution** to a common **problem** arising within a **context**

Mobile devices

Aerospace

Electronic Trading



Automotive

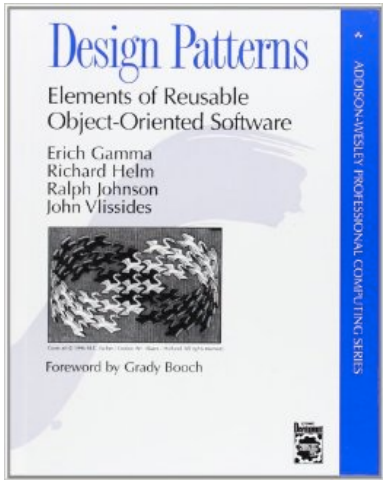
What is a Pattern? The "Alexandrian" Definition

Each pattern describes a problem
which occurs over and over again in our environment,
and then describes
the core of the solution to that problem,
in such a way that
you can use this solution a million times over,
without ever doing it the same way twice

C. Alexander, "*The Timeless Way of Building*", 1979

Design Patterns

- "A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context." – [GoF]



What Makes it a Pattern? A pattern must…

- …solve a problem
  - It must be useful
- …have a context
  - It must describe where the solution can be used
- …recur
  - Must be relevant in other situations; rule of three
- ... teach
  - Provide sufficient understanding to tailor the solution
- ... have a name
  - Referred consistently

Why are Patterns Important?

- "Patterns provide an incredibly dense means of efficient and effective communication between those who know the language." – [Nate Kirby]
- "Human communication is the bottleneck in software development. If patterns can help developers communicate with their clients, their customers, and each other, then patterns help fill a crucial need in our industry." – [Jim Coplien]
- "Patterns don't give you code you can drop into your application, they give you experience you can drop into your head." – [Patrick Logan]
- "Giving someone a piece of code is like giving him a fish; giving him a pattern is like teaching him to fish." – [Don Dwiggins]

Patterns to help with design changes…

Design Pattern Space

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

When your only tool is a hammer…

- …all the problems look like a nail

- When first learning patterns, all problems begin to look like the problem under consideration – try to avoid this!
  - Similar to someone just learning to play chess and using the same strategy everywhere – eventually you will get burned!

When your only tool is a hammer…

- Seek generality, but don't brand everything as a pattern

More Pattern Information

- Robert C. Martin's Chess Analogy
  - http://www.cs.wustl.edu/~schmidt/cs242/learning.html
- John Vlissides' "Top 10 Misconceptions"
  - http://www.research.ibm.com/designpatterns/pubs/top10misc.html
- Seven Habits of Successful Pattern Writers
  - http://www.research.ibm.com/designpatterns/pubs/7habits.html
- Brad Appleton's "Patterns in a Nutshell"
  - http://www.cmcrossroads.com/bradapp/docs/patterns-nutshell.html
- Mike Duell's non-software examples
  - http://www.cours.polymtl.ca/inf3700/divers/nonSoftwareExample/patexamples.html