Taming Software Complexity (1)

September 26, 2017

Byung-Gon Chun

(Slide credits: George Candea, EPFL)

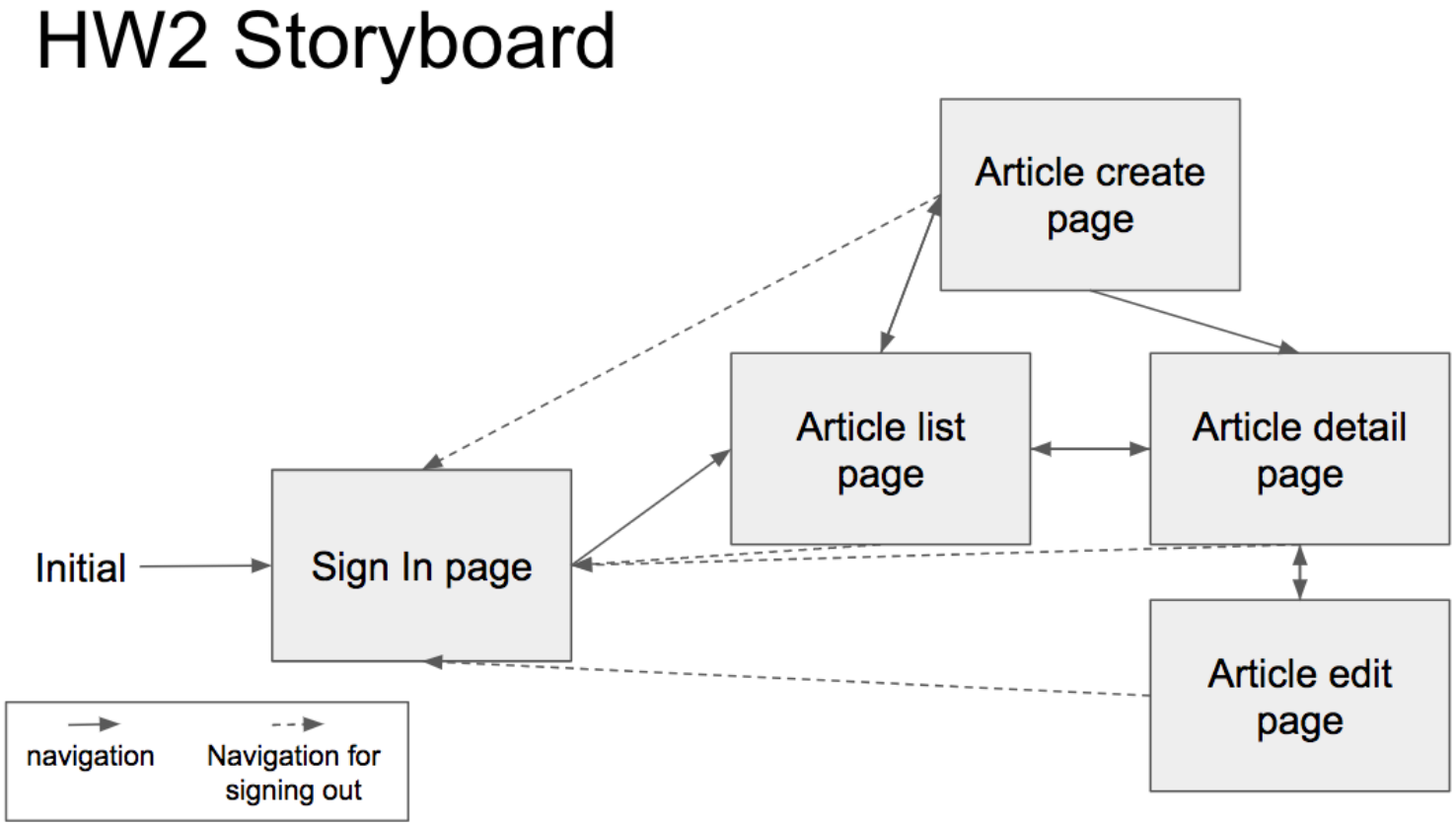
Announcement

- Homework 1 done!
- Homework 2 out last Saturday – due Oct. 7 (Sat), 20:59
  - Wonwook will hold **special virtual office hours** (e.g., skype group call): 2-3pm, Oct. 2 (Mon) and Oct. 6 (Fri).
- Project proposal done! – Exciting proposals!
  - Feedback given
- Teams 1-7: Dongjin, Teams 8-13: Wonwook
- Guest lecture on SW Development at Facebook, Google, and Microsoft by three lab grad students who did internship at the companies

Homework 2 - Angular2 (Due 10/7(Sat) 20:59)

A front-end for a blogging service using Angular2.

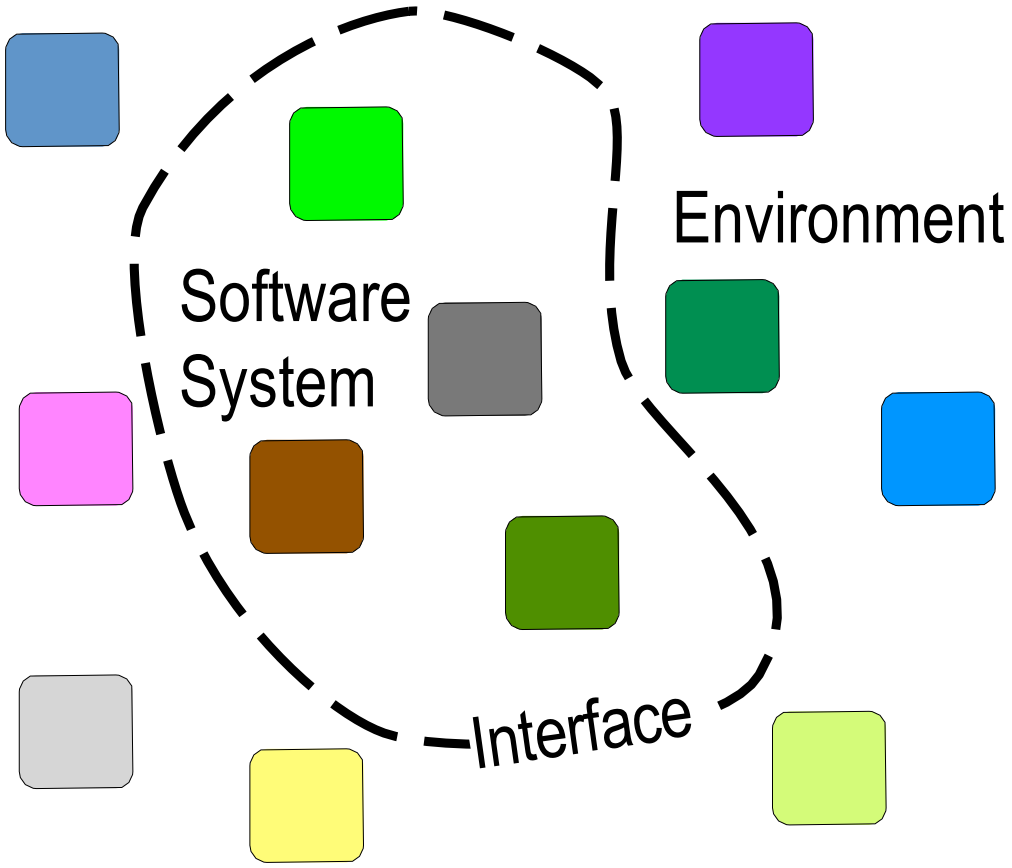
https://github.com/swsnu/swppfall2017/tree/master/hw2



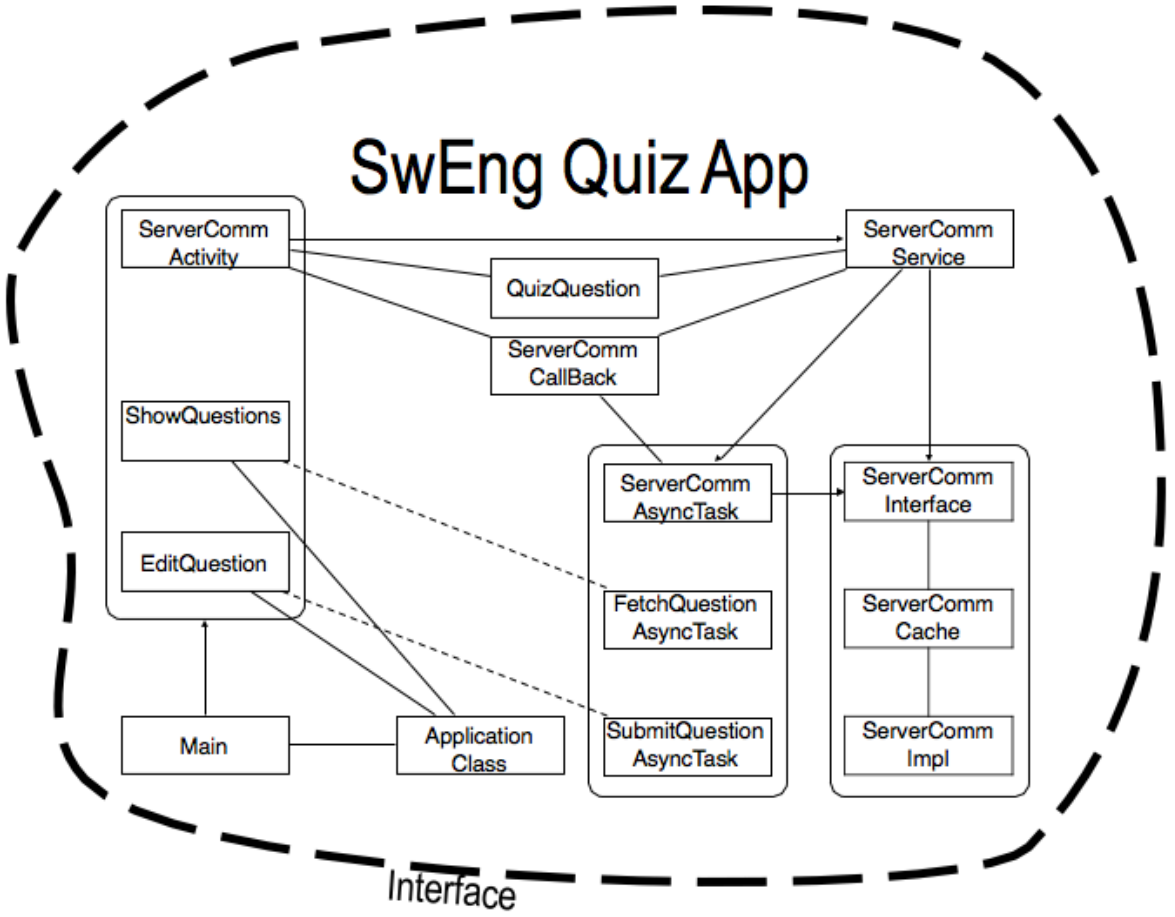
Symptoms of Complexity

What is a Software System?

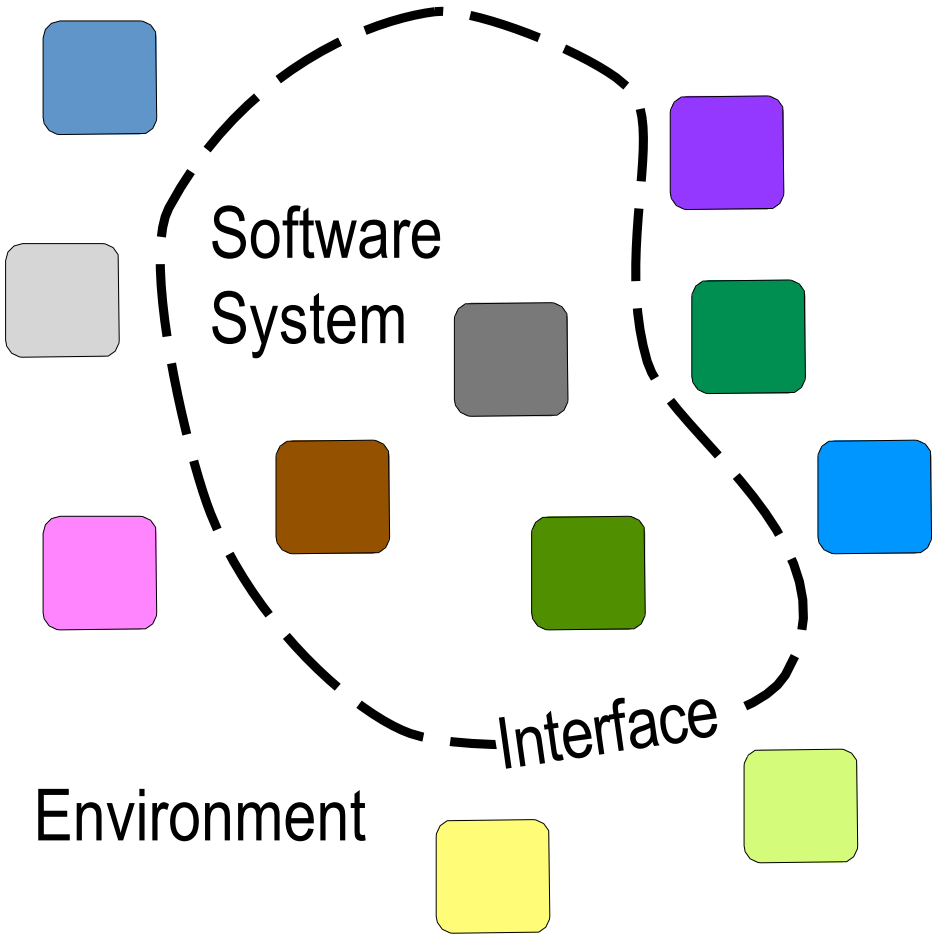
A group of interconnected components that exhibits an expected collective behavior observed at the interfaces with its environment.



Examples of Systems

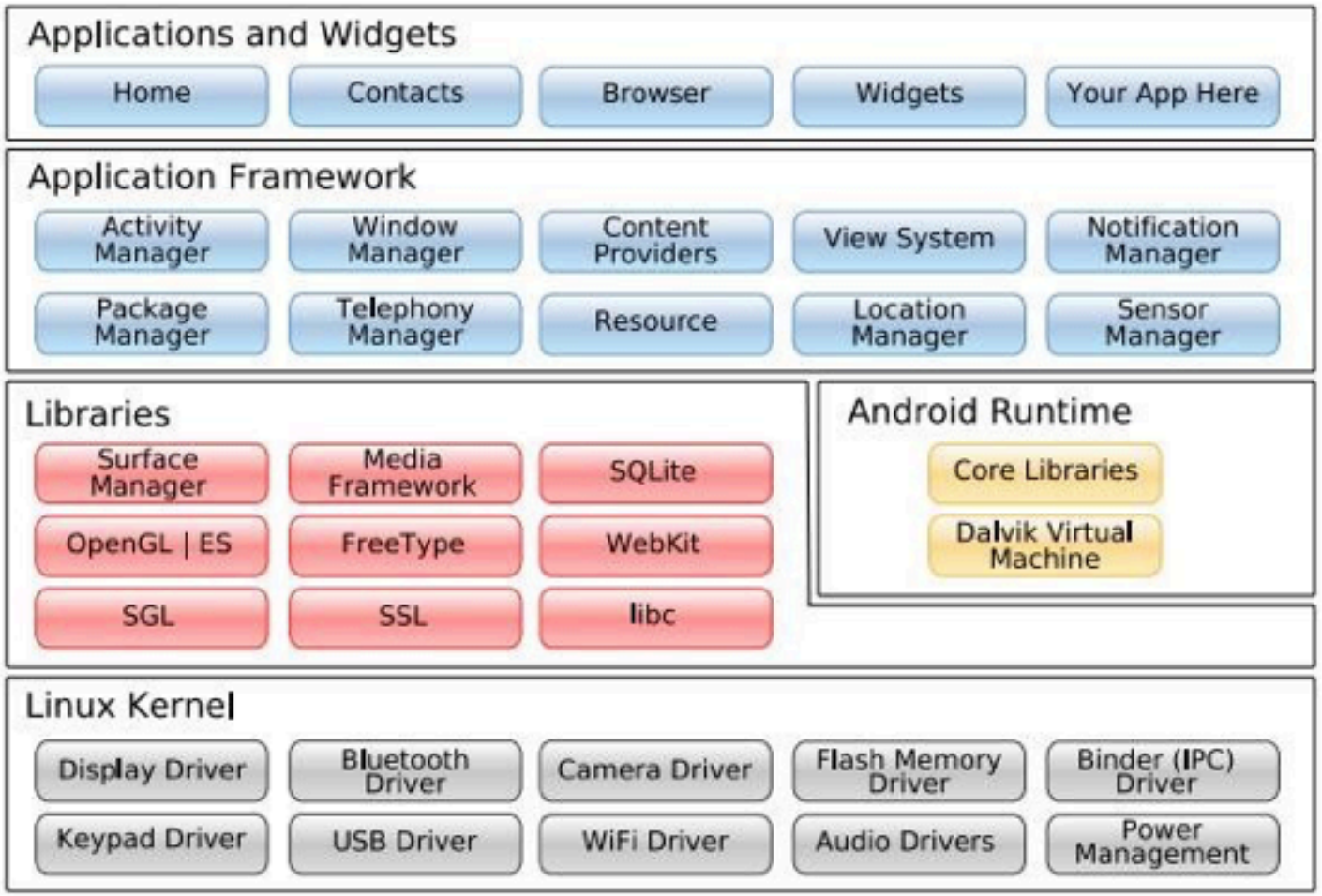


Four Symptoms of Complexity

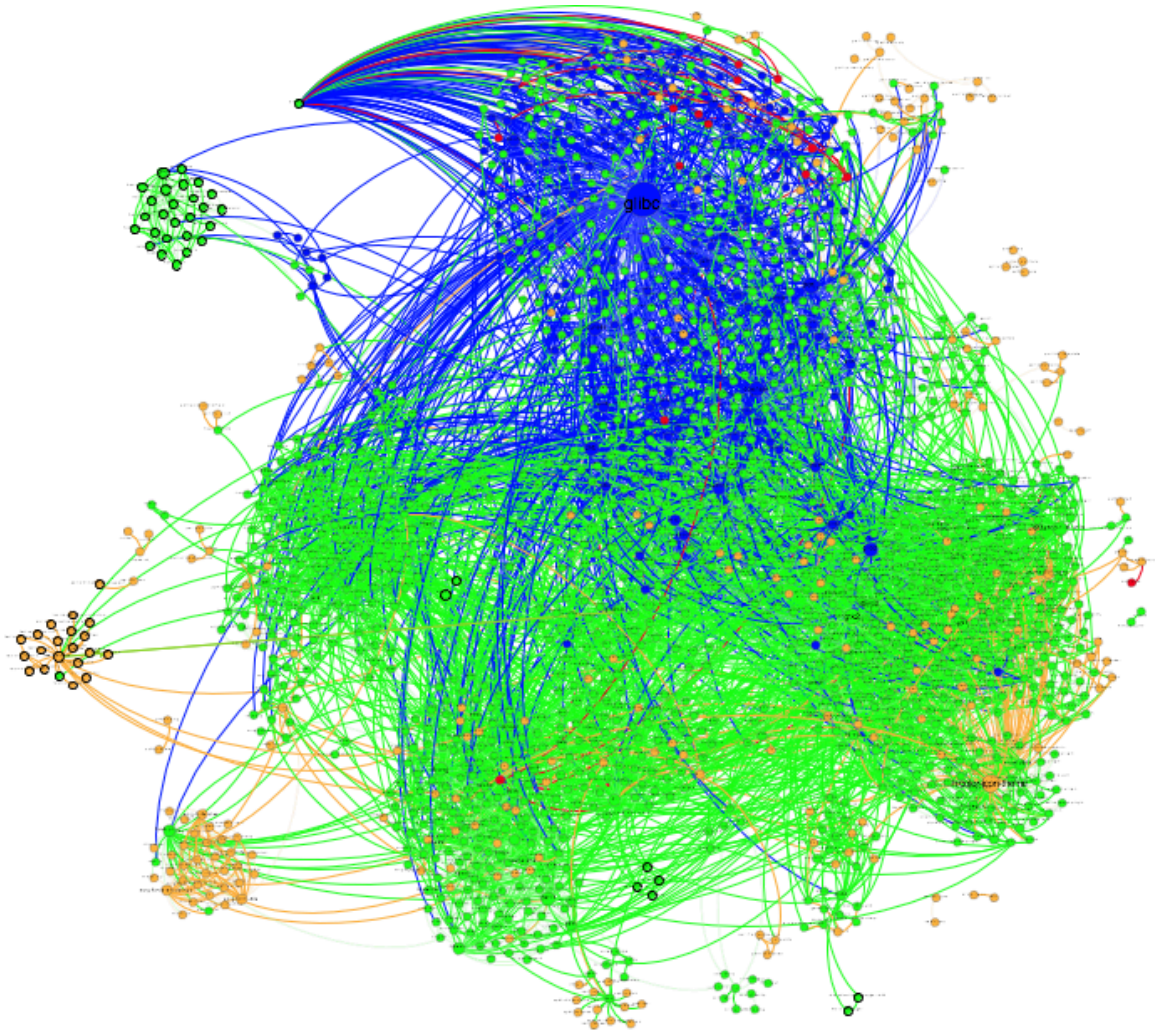


- Large number of components
- Large number of interconnections
- Many irregularities and exceptions
- High “Kolmogorov complexity”

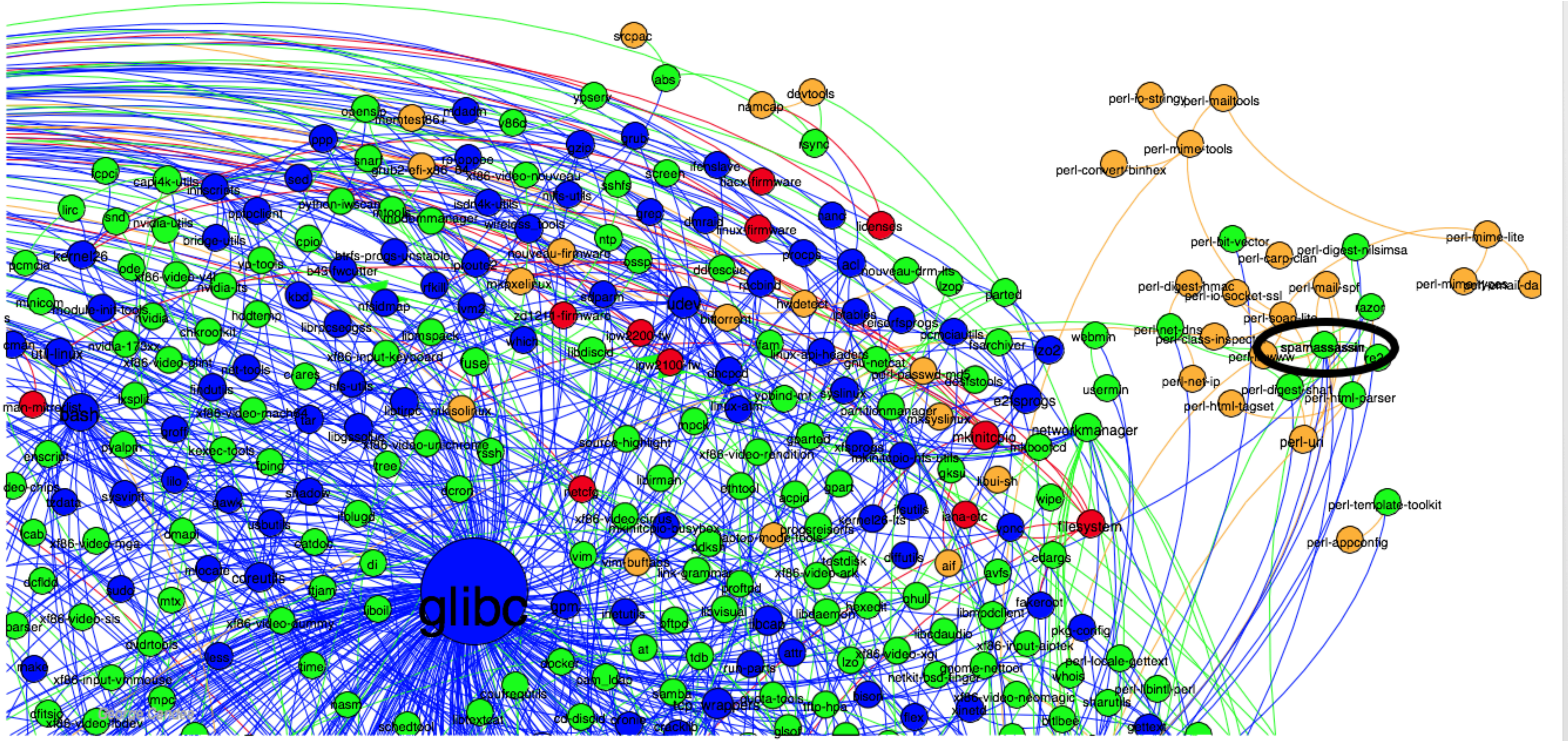
Symptom #1: Many Components



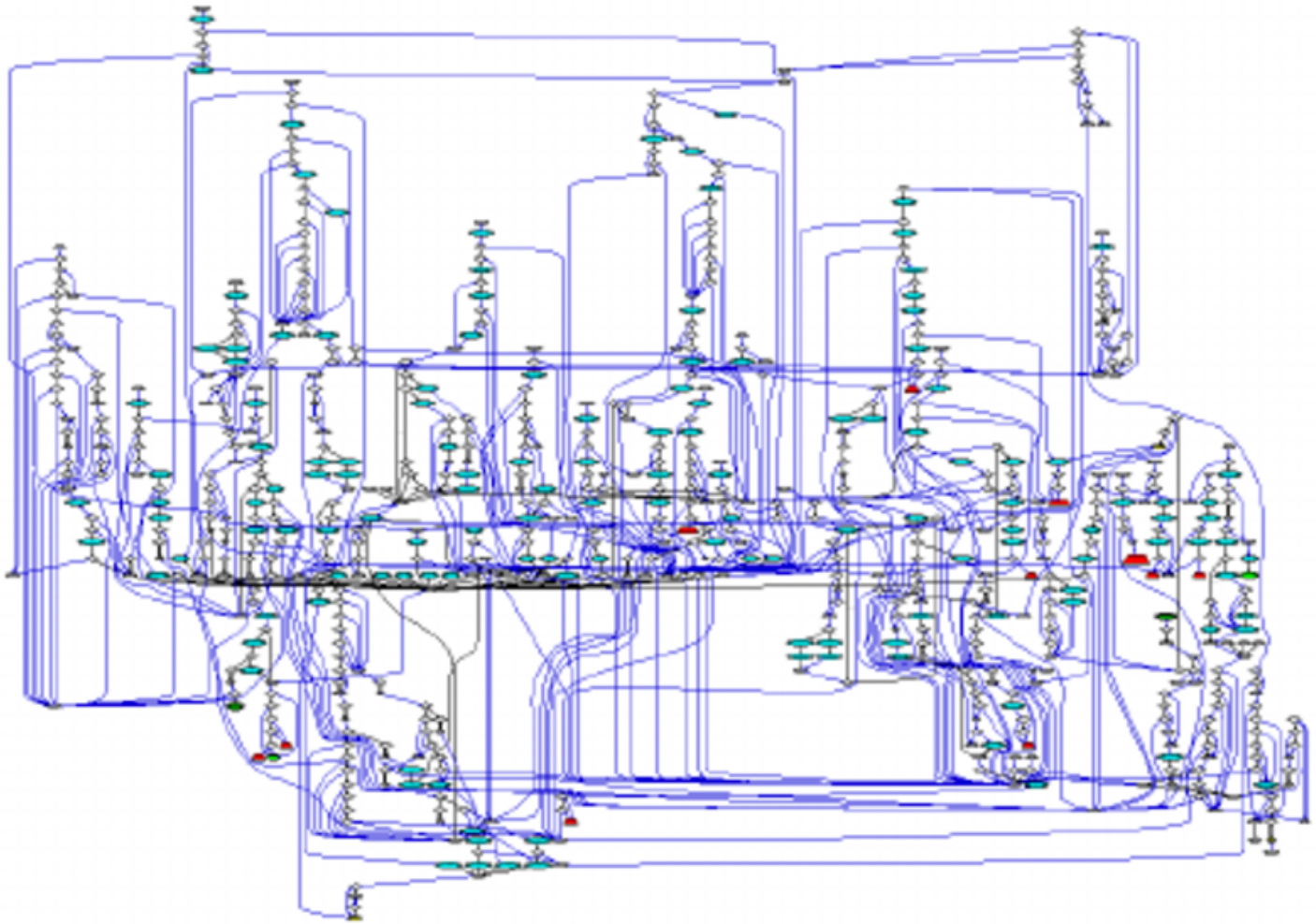
Symptom #2: Many Interconnections



Symptom #2: Many Interconnections



Symptom #3: Irregularity and Exceptions



Symptom #4: High Kolmogorov Complexity

|*AAAAAAA … AAAAB*| = 10^6+1

$\mathbf{K}$(*AAAAAAA … AAAAB*) =
|"1 million *A*s followed by 1 *B*"|

⇒ simple

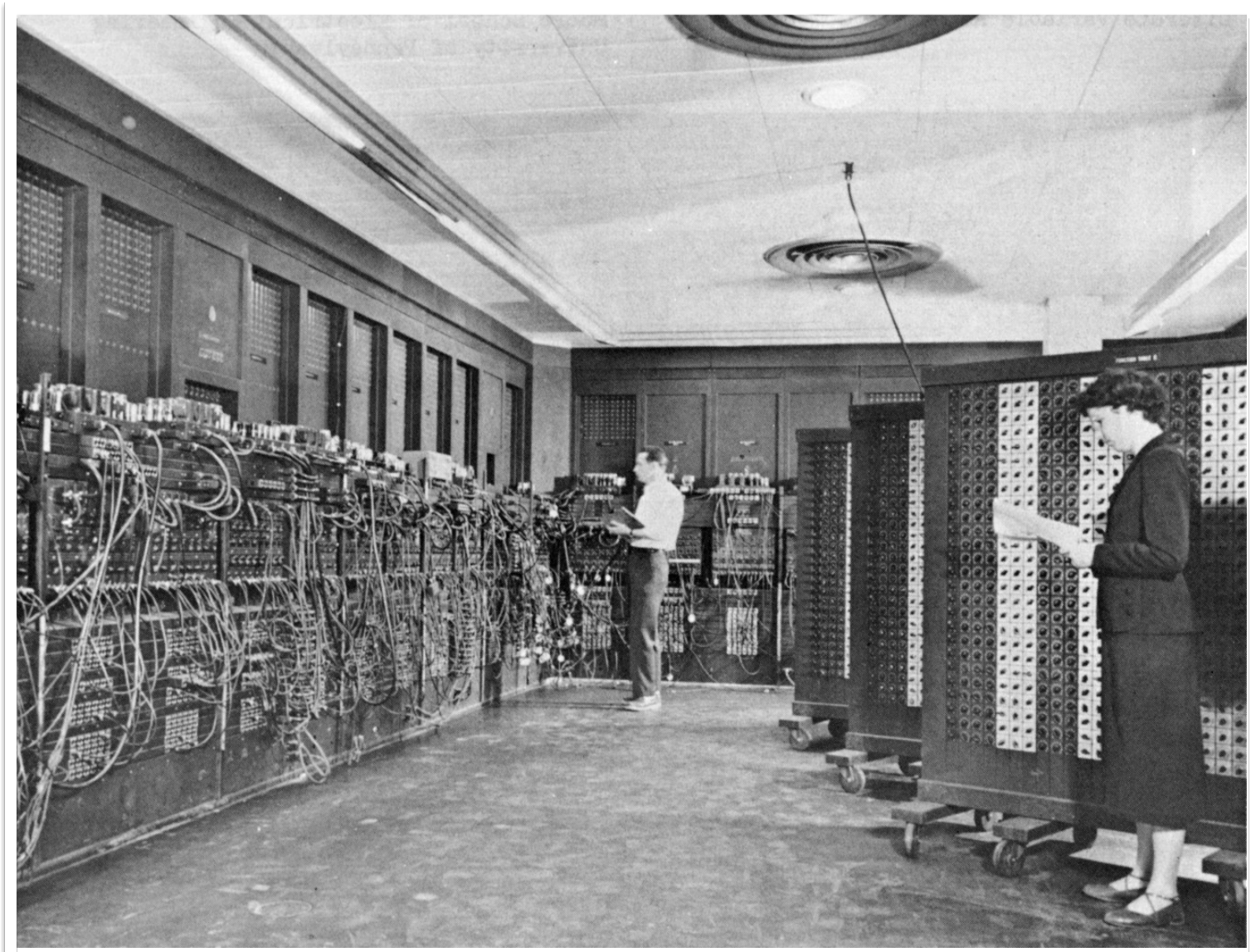
|*ABDAGHDBBCAD*…| = 10^6+1

$\mathbf{K}$(*ABDAGHDBBCAD*…) = 10^6+1

⇒ complex

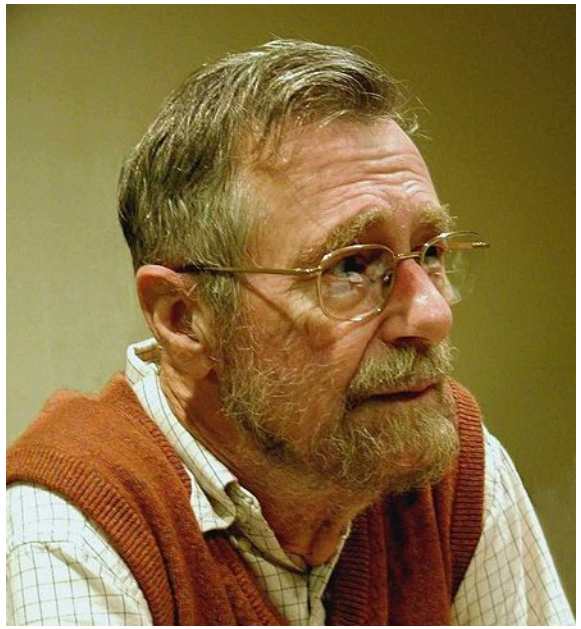
- Kolmogorov complexity
  - *minimal length of a description of the object*
  - *computation resources needed to specify an object*
- $\mathbf{K}$(object) >= |object| => complex
  $\mathbf{K}$(object) << |object| => simple

Modularity



```
MAIN0001* PROGRAM TO SOLVE THE QUADRATIC EQUATION
MAIN0002         READ 10,A,B,C $
MAIN0003         DISC = B*B-4*A*C $
MAIN0004         IF (DISC) NEGA,ZERO,POSI $
MAIN0005    NEGA R = 0.0 - 0.5 * B/A $
MAIN0006         AI = 0.5 * SQRTF(0.0-DISC)/A $
MAIN0007         PRINT 11,R,AI $
MAIN0008         GO TO FINISH $
MAIN0009    ZERO R = 0.0 - 0.5 * B/A $
MAIN0010         PRINT 21,R $
MAIN0011         GO TO FINISH $
MAIN0012    POSI SD = SQRTF(DISC) $
MAIN0013         R1 = 0.5*(SD-B)/A $
MAIN0014         R2 = 0.5*(0.0-(B+SD))/A $
MAIN0015         PRINT 31,R2,R1 $
MAIN0016 FINISH STOP $
MAIN0017      10 FORMAT( 3F12.5 ) $
MAIN0018      11 FORMAT( 19H TWO COMPLEX ROOTS:, F12.5,14H PLUS OR MINUS,
MAIN0019         F12.5, 2H I ) $
MAIN0020      21 FORMAT( 15H ONE REAL ROOT:, F12.5 ) $
MAIN0021      31 FORMAT( 16H TWO REAL ROOTS:, F12.5, 5H AND , F12.5 ) $
MAIN0022         END $
```

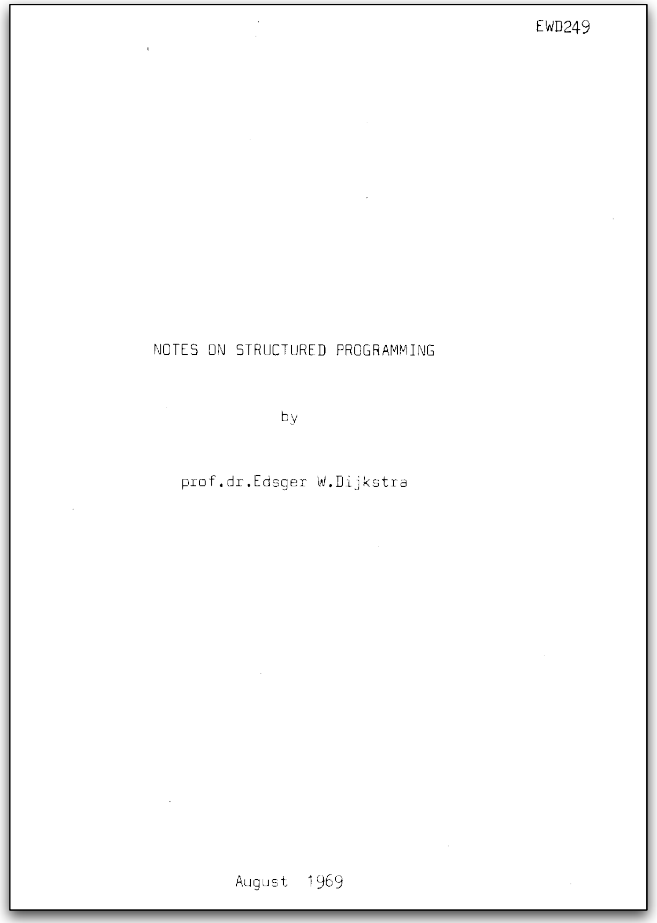
Structured Programming



Edsger Dijkstra

The competent programmer is fully aware of the strictly limited size of his own skull and therefore approaches the programming task in full humility.

Structured Programming



EWD249

NOTES ON STRUCTURED PROGRAMMING

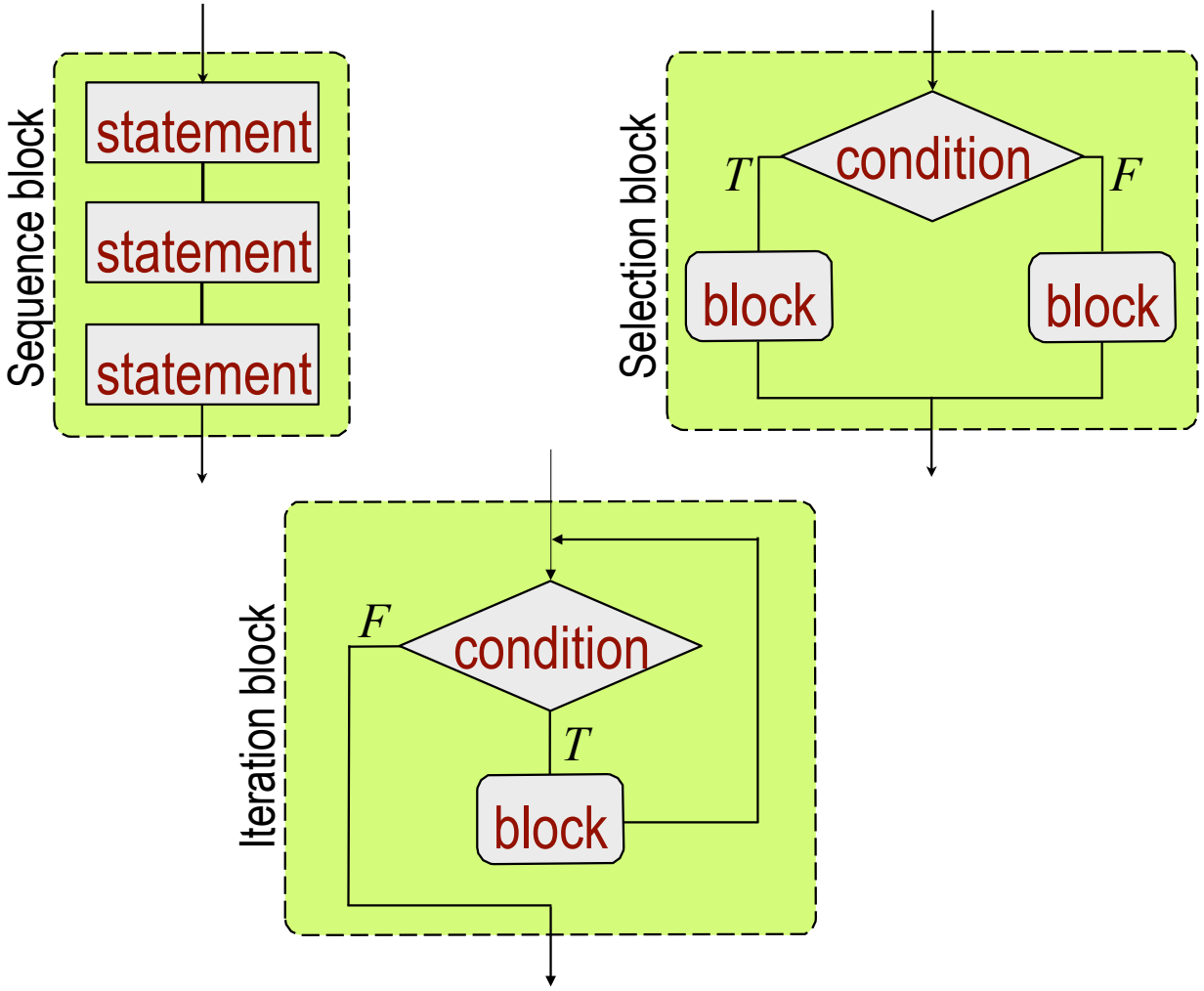
by

prof.dr.Edsger W.Dijkstra

August 1969

- Three basic constructs
  - *single-entry / single-exit control constructs*
  - *sequence, selection, iteration*
- Structured program
  - *ordered, disciplined, doesn't jump around unpredictably*
  - *can read easily and reason about ⇒ higher quality*

Structured Programming



- Three basic constructs
  - *single-entry / single-exit control constructs*
  - *sequence, selection, iteration*
- Structured program
  - *ordered, disciplined, doesn't jump around unpredictably*
  - *can read easily and reason about ⇒ higher quality*

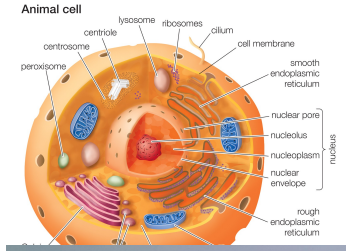
Object-Oriented Programming

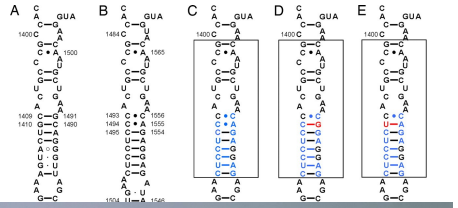
Simula67

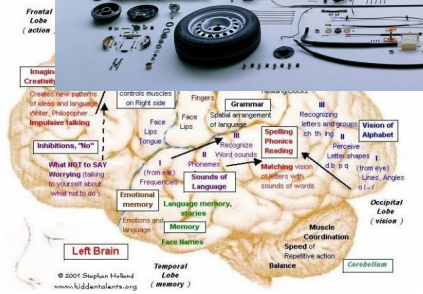
```
Begin
   Class Glyph;
      Virtual: Procedure print Is Procedure print;;
   Begin
   End;

   Glyph Class Char (c);
      Character c;
   Begin
      Procedure print;
        OutChar(c);
   End;

   Glyph Class Line (elements);
      Ref (Glyph) Array elements;
   Begin
      Procedure print;
      Begin
         Integer i;
         For i:= 1 Step 1 Until UpperBound (elements, 1) Do
            elements (i).print;
         OutImage;
      End;
   End;

   Ref (Glyph) rg;
   Ref (Glyph) Array rgs (1 : 4);

   ! Main program;
   rgs (1):- New Char ('A');
   rgs (2):- New Char ('b');
   rgs (3):- New Char ('b');
   rgs (4):- New Char ('a');
   rg:- New Line (rgs);
   rg.print;
End;
```

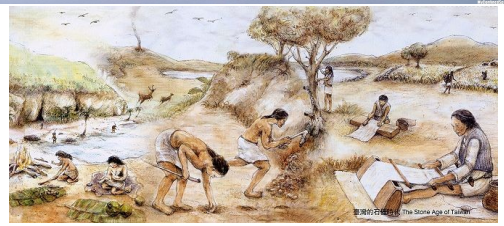
- Encapsulation in OOP
  - *self-contained modules*
  - *bind the data to the methods that process it*
- Simula67 developed in the late 60s
  - *followed by Smalltalk, C++, C#, Java, etc.*

Modularity Outside Computer Science



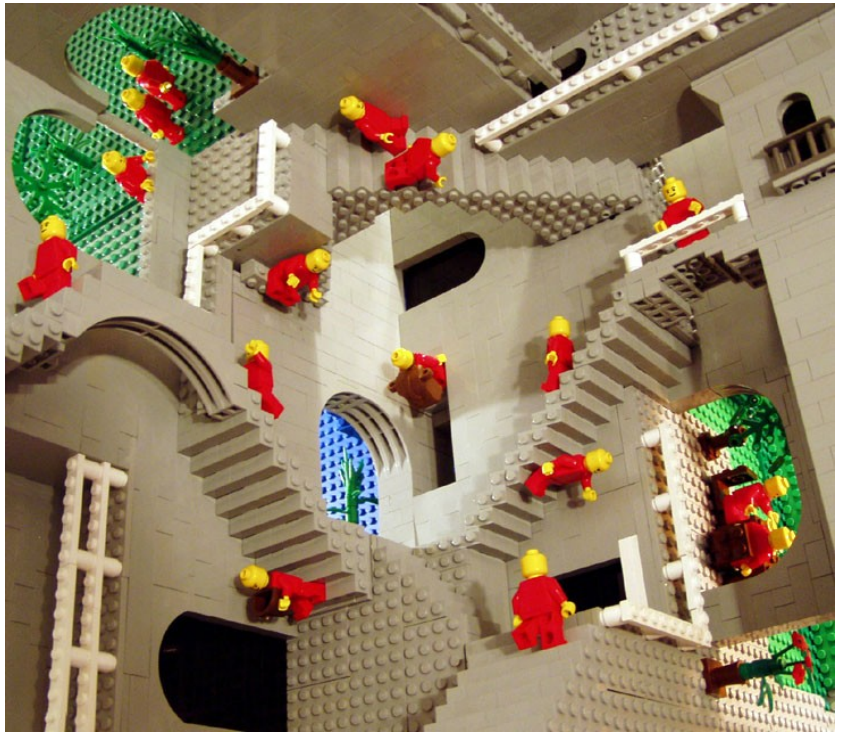






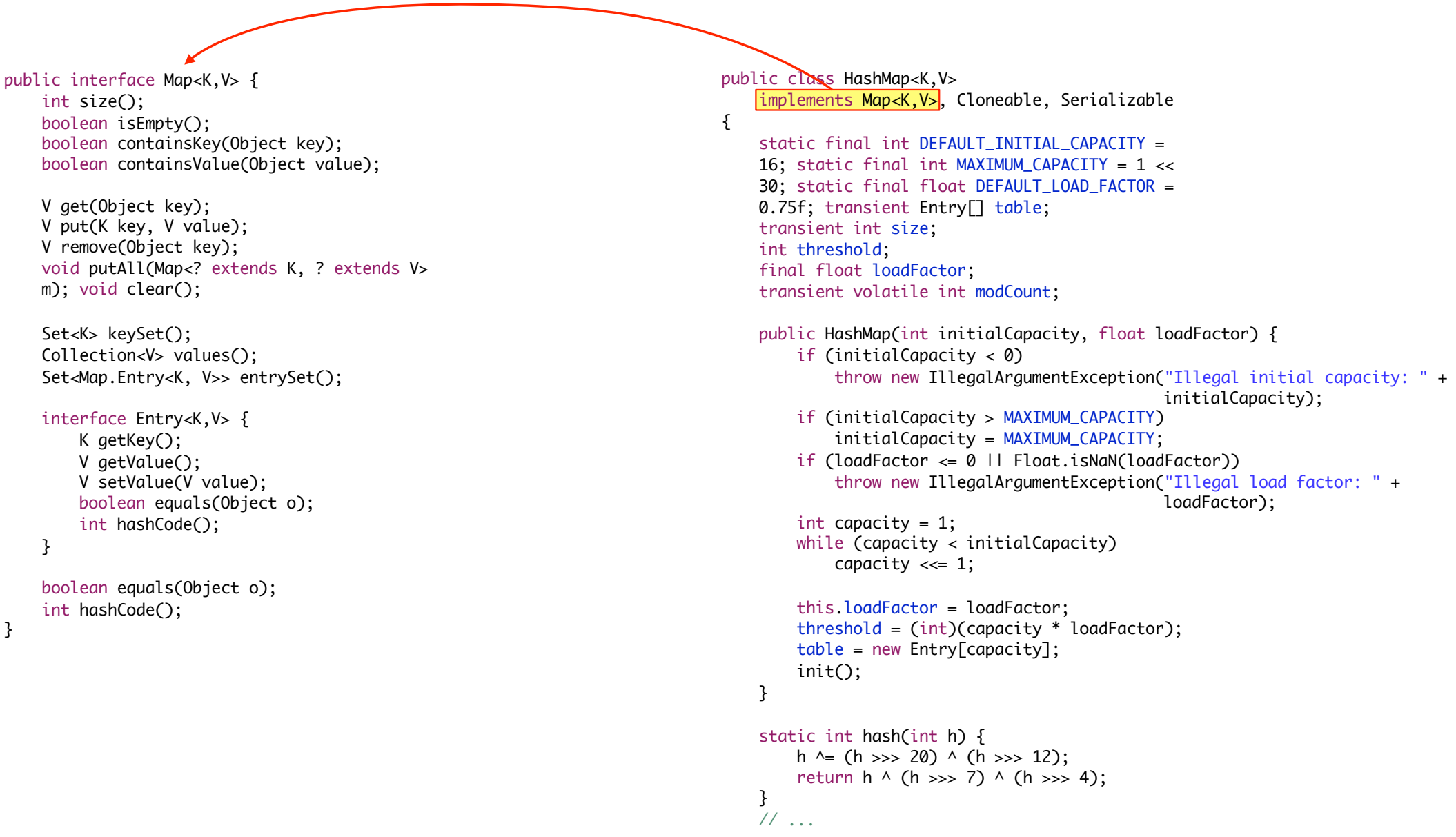
- Cell = module used to build organisms
- Gene = unit (module) of evolution
- Cognitive activity (may be) modularized
- Division of labor = modularization
- Spare parts = modules
- IKEA … modular furniture

Modularity in Software Engineering



- Modules = replaceable units
- Classes in OOP
  - *contain behaviors inside the modules*
  - *implement them in isolation*
  - *test them in isolation*
  - *configure them separately*
  - *maintain them separately*
- Ultimate goal
  - *productively reason about combination of classes*

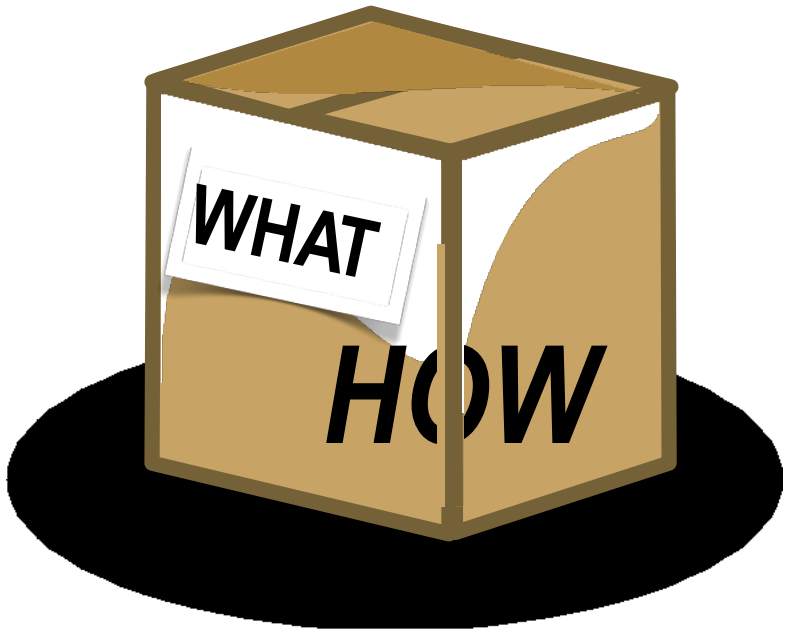
Abstraction



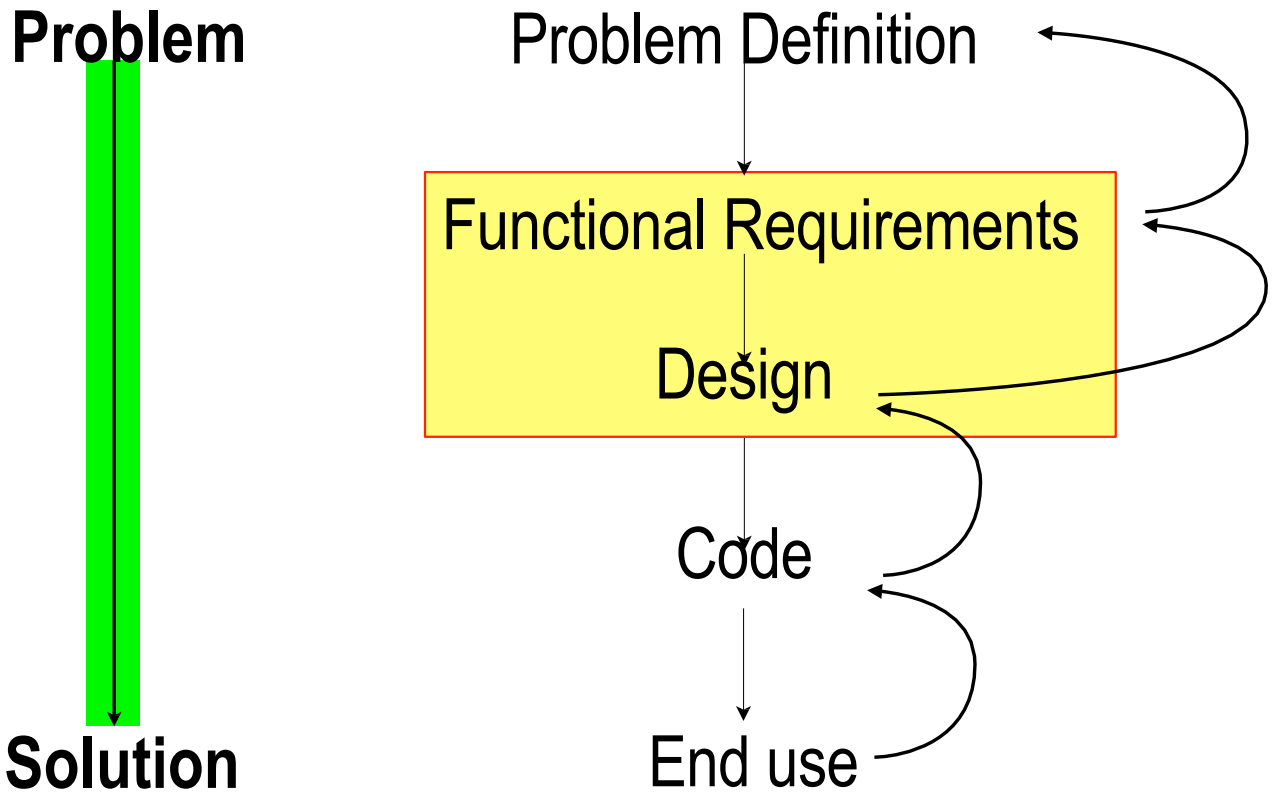
```java
public interface Map<K,V> {
    int size();
    boolean isEmpty();
    boolean containsKey(Object key);
    boolean containsValue(Object value);

    V get(Object key);
    V put(K key, V value);
    V remove(Object key);
    void putAll(Map<? extends K, ? extends V>
    m); void clear();

    Set<K> keySet();
    Collection<V> values();
    Set<Map.Entry<K, V>> entrySet();

    interface Entry<K,V> {
        K getKey();
        V getValue();
        V setValue(V value);
        boolean equals(Object o);
        int hashCode();
    }

    boolean equals(Object o);
    int hashCode();
}
```

```java
public class HashMap<K,V>
    implements Map<K,V>, Cloneable, Serializable
{
    static final int DEFAULT_INITIAL_CAPACITY =
    16; static final int MAXIMUM_CAPACITY = 1 <<
    30; static final float DEFAULT_LOAD_FACTOR =
    0.75f; transient Entry[] table;
    transient int size;
    int threshold;
    final float loadFactor;
    transient volatile int modCount;

    public HashMap(int initialCapacity, float loadFactor) {
        if (initialCapacity < 0)
            throw new IllegalArgumentException("Illegal initial capacity: " +
                                               initialCapacity);
        if (initialCapacity > MAXIMUM_CAPACITY)
            initialCapacity = MAXIMUM_CAPACITY;
        if (loadFactor <= 0 || Float.isNaN(loadFactor))
            throw new IllegalArgumentException("Illegal load factor: " +
                                               loadFactor);
        int capacity = 1;
        while (capacity < initialCapacity)
            capacity <<= 1;

        this.loadFactor = loadFactor;
        threshold = (int)(capacity * loadFactor);
        table = new Entry[capacity];
        init();
    }

    static int hash(int h) {
        h ^= (h >>> 20) ^ (h >>> 12);
        return h ^ (h >>> 7) ^ (h >>> 4);
    }
    // ...
```

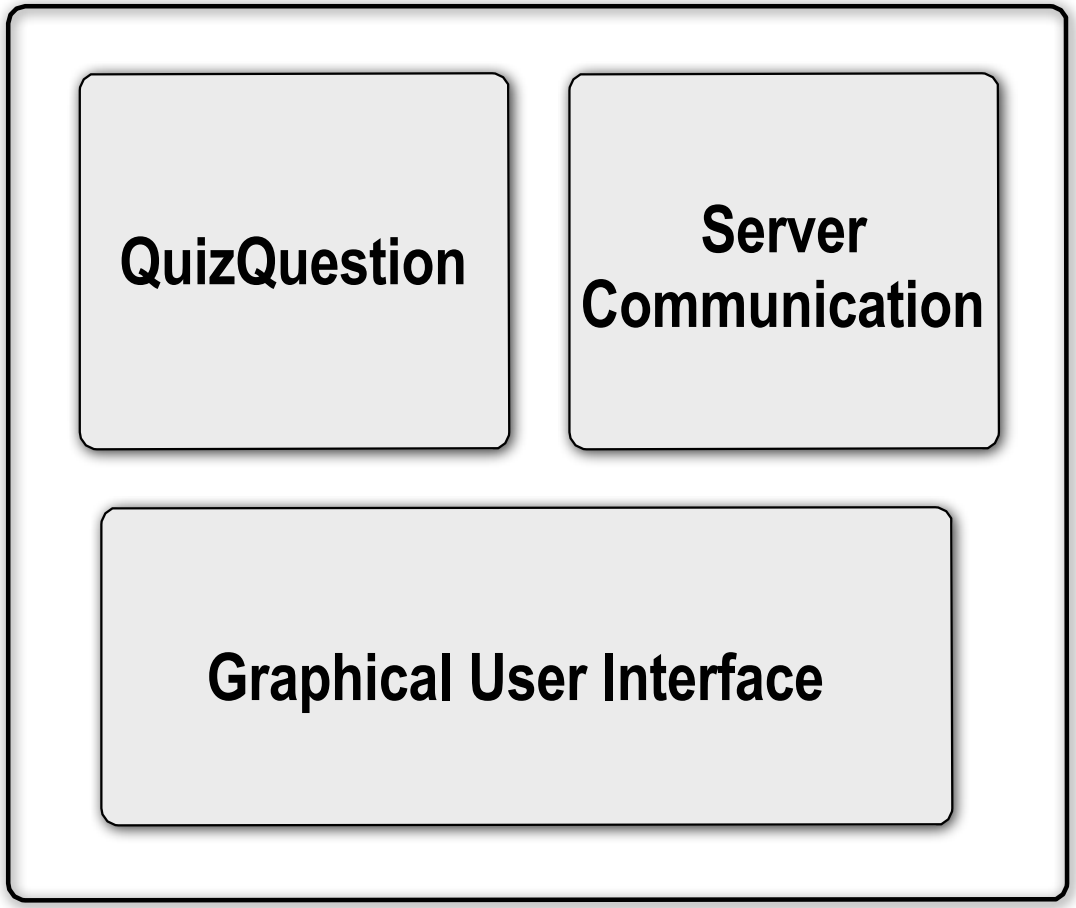
Abstraction



- Abstraction
  - *specifies "what" a component/subsystem does together*
  - *with modularity, it separates "what" from "how"*

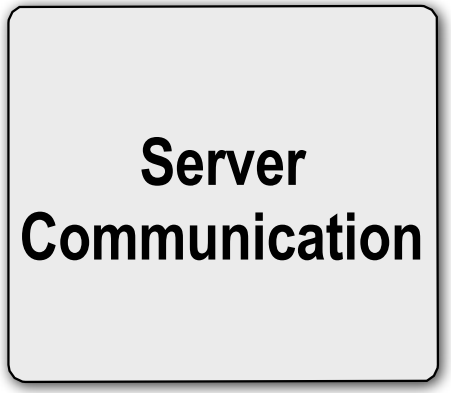

Problem
Solution
Problem Definition
Functional Requirements
Design
Code
End use

Top-Down Design



- Break down into major subsystems
  - *e.g., platform dependencies, database access, business logic, menu hierarchies*

Top-Down Design



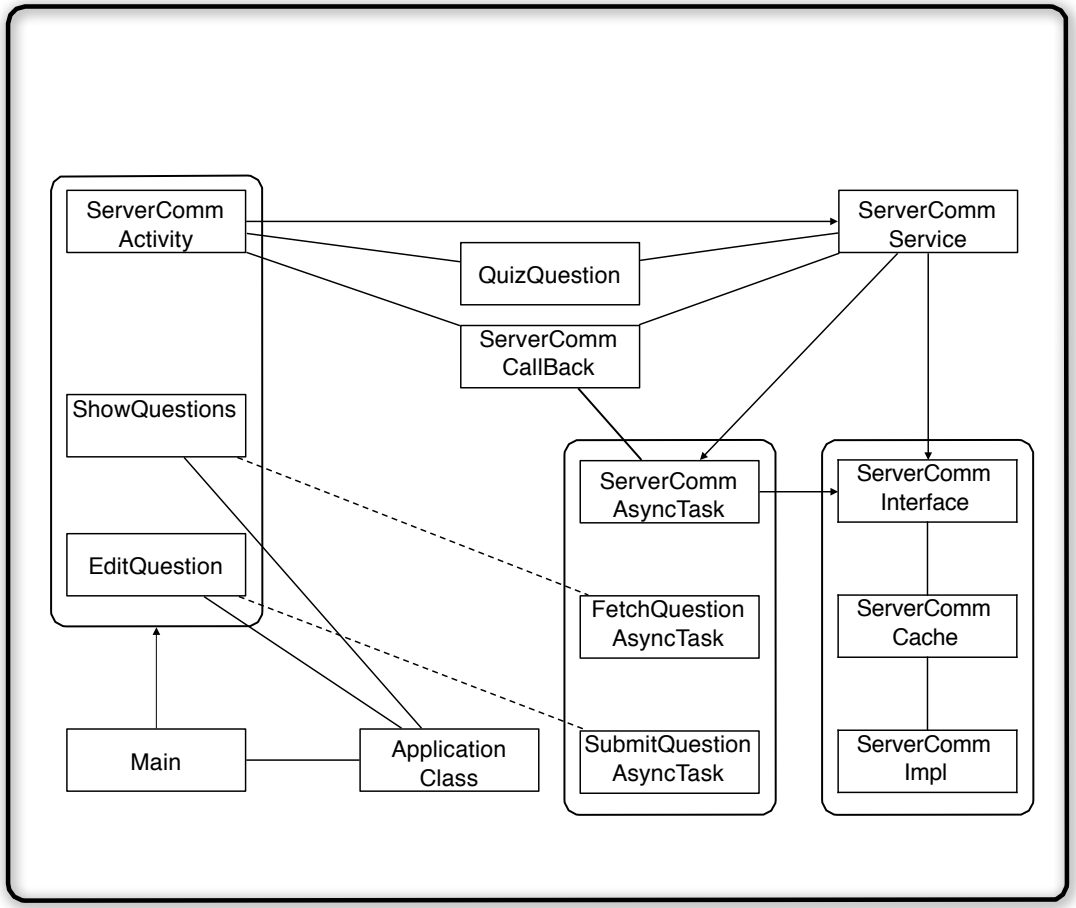
- Break down into major subsystems
  - *e.g., platform dependencies, database access, business logic, menu hierarchies*

Top-Down Design

```
public class ServerCommunication {
    public static ServerCommunicationFactory getInstance() {}

    public static void getRandomQuestion(AsyncTaskCallBack<QuizQuestion> parent,
         String session) {}
    public static void postQuestion(QuizQuestion question,
         AsyncTaskCallBack<QuizQuestion> parent, String session)
    {} public static void updateQuestion(QuizQuestion question,
         AsyncTaskCallBack<Void> parent, String session) {}
    public static void deleteQuestion(String questionId,
         AsyncTaskCallBack<Void> parent, String session) {}
    public static void authenticate(String username, String password,
         AsyncTaskCallBack<LoginStatus> parent) {}
    public static void getQuestionsOwnedBy(String username,
         AsyncTaskCallBack<List<QuizQuestion>> parent) {}
    public static void getQuestionsTaggedWith(String tag,
         AsyncTaskCallBack<List<QuizQuestion>> parent) {}
    public static void getRatings(QuizQuestion question, String session,
         AsyncTaskCallBack<AggregatedRatings> parent) {}
    public static void setRating(QuizQuestion question, String session,
         RatingType rating, AsyncTaskCallBack<RatingType> parent) {}
    public static void getMyRating(QuizQuestion question, String
    session,
         AsyncTaskCallBack<RatingType> parent) {}
    public static void getMyKarma(String session,
         AsyncTaskCallBack<Karma> parent) {}
    public static void fetchTests(String session,
         AsyncTaskCallBack<List<QuizTest>> parent) {}
    public static void fetchSingleTest(String session, String testId,
         AsyncTaskCallBack<QuizTest> parent) {}
    public static void postTestAnswers(String session, String testId,
}        int[] answers, AsyncTaskCallBack<Double> parent) {}
```

- Break down into major subsystems
  - *e.g., platform dependencies, database access, business logic, menu hierarchies*
- Define interfaces for subsystems

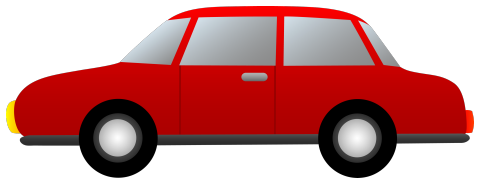
Top-Down Design



- Break down into major subsystems
  - *e.g., platform dependencies, database access, business logic, menu hierarchies*
- Define interfaces for subsystems
- Identify all major classes
- Define interfaces for these classes
  - enough detail to be directly implementable
  - not more detail than is strictly necessary
  - only specify the what, not the how

Classes <-> Real-world Objects

```
public interface StudentInterface {
    public void Student(String first, String last,
            String email, int section);
    public String toString();
    public String getLastName();
    public void setLastName(String lastName);
    public String getEmailAddr();
    public void setEmailAddr(String emailAddr);
    public String getFirstName();
    public void setFirstName(String firstName);
}
```



- Identify objects and their attributes
  - e.g., student, car, teacher, ...
- What can be done to object ?
- What can object do to other objects ?
  - containment / inheritance
- Essential vs. incidental properties
  - essential for a car: engine, wheels, doors, ...
  - incidental for a car: turbo engine, color red, ...
- Minimize complexity
  - reduce what is required to fit in one brain
  - keep incidental complexity from proliferating