

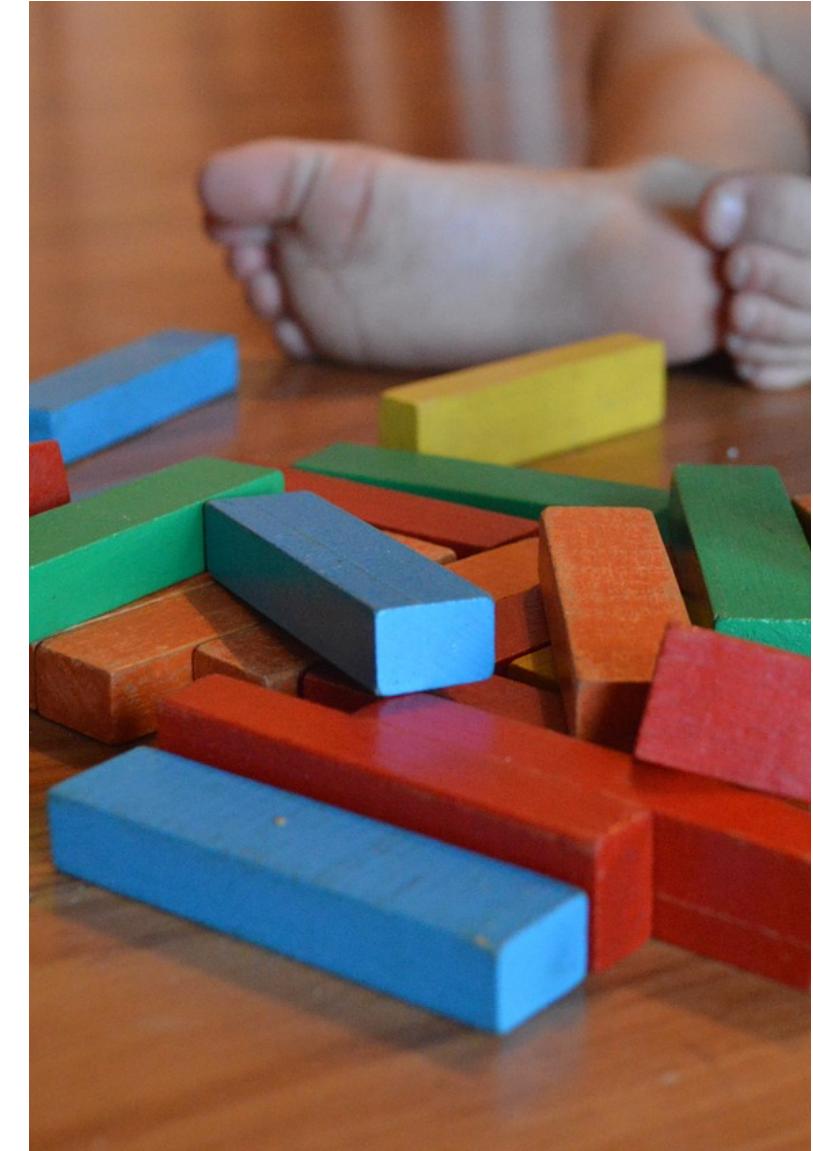
Design for Usability

Jin L.C. Guo

Usability

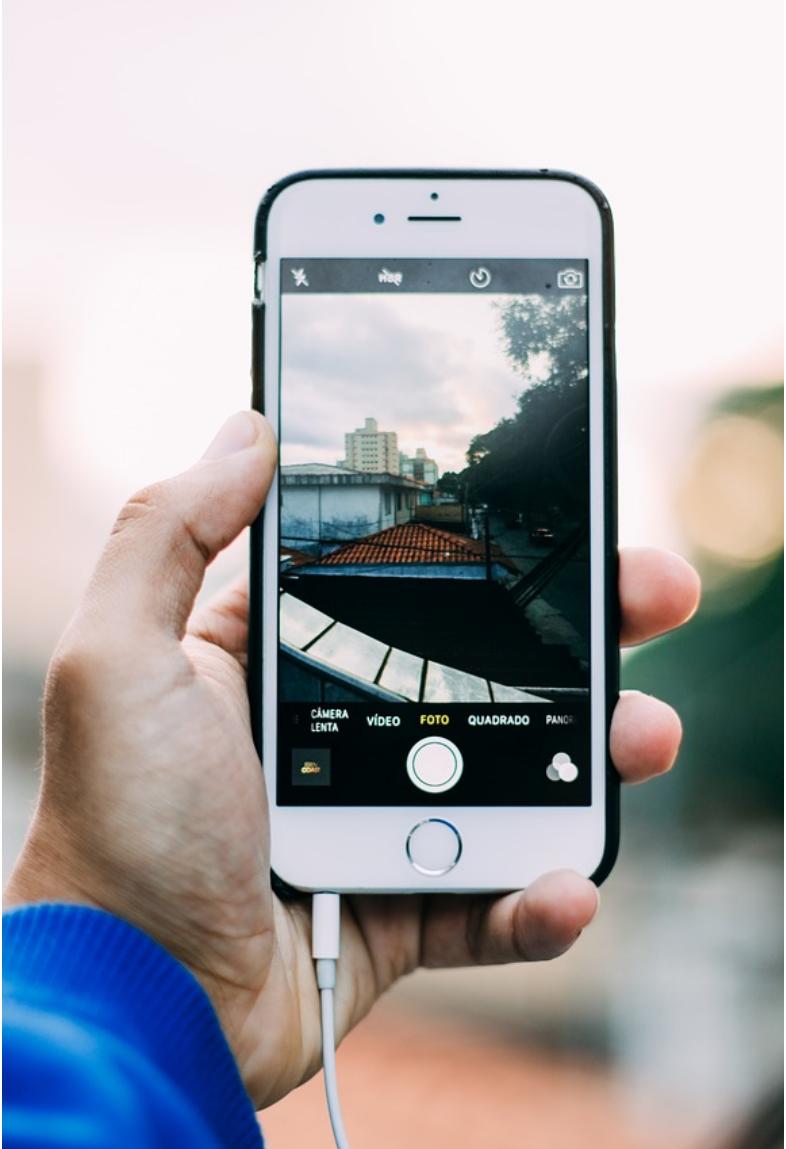
- **A Quality Attribute**

How easy and pleasant user interfaces are to use.



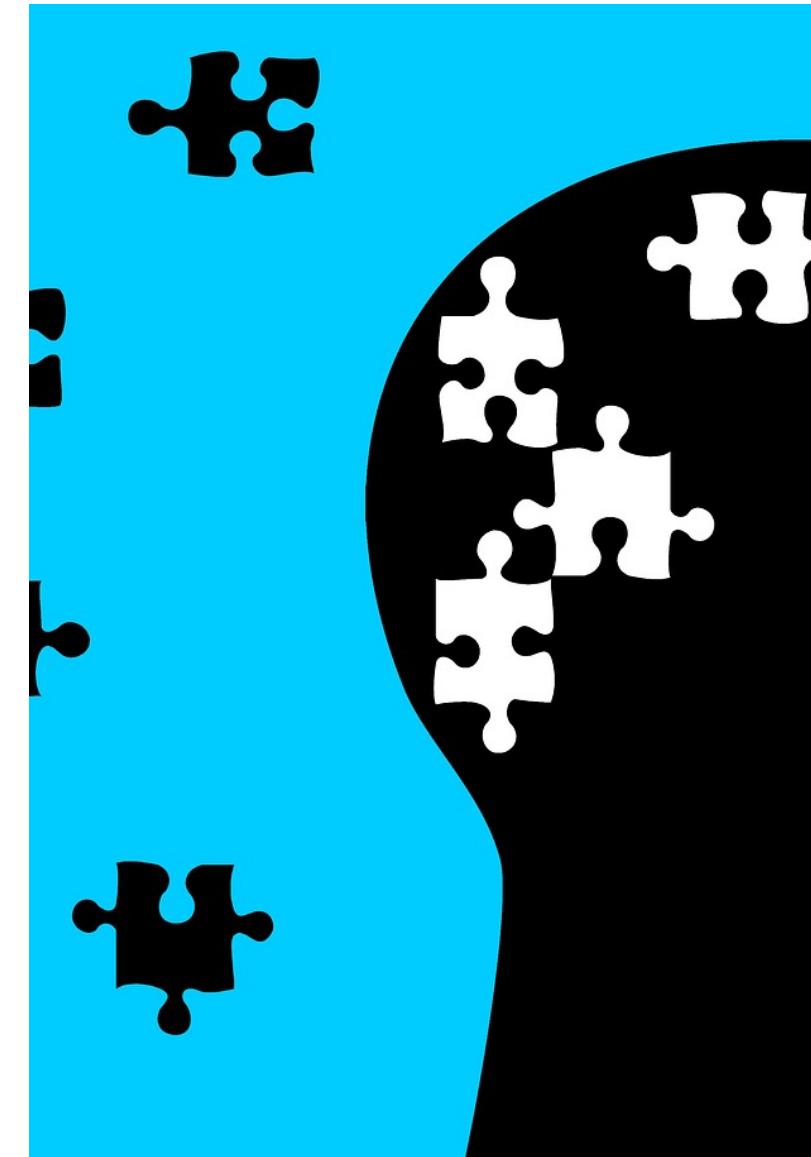
Usability

- **Learnability:** How easy is it for users to accomplish basic tasks the first time they encounter the design?



Usability

- **Efficiency:** Once users have learned the design, how quickly can they perform tasks?



Usability

- **Memorability:** When users return to the design after a period of not using it, how easily can they re-establish proficiency?



Usability

- **Errors:** How many errors do users make, how severe are these errors, and how easily can they recover from the errors?



Usability

- **Satisfaction:** How pleasant is it to use the design?

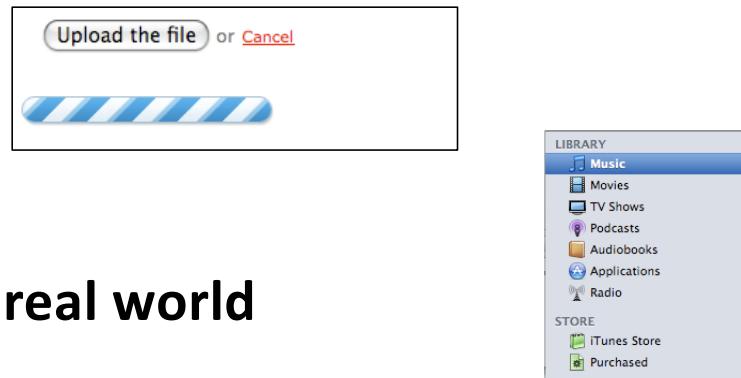
User Testing

- Find representative users
- Ask users to perform representative tasks
- **Observe**
 - what the users do;
 - where they succeed;
 - and where they have difficulties with the user interface



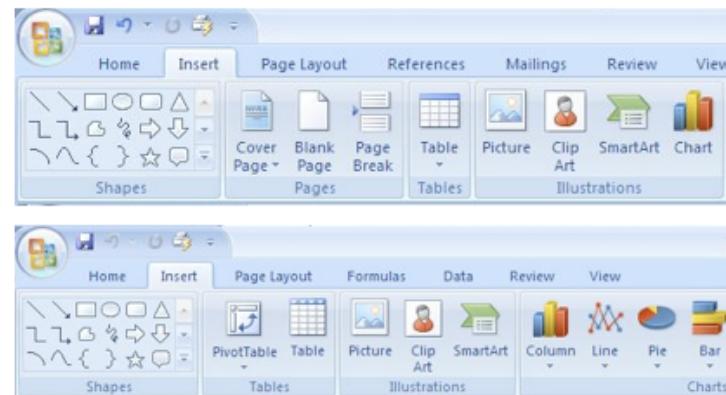
Usability Heuristics for User Interface Design

Visibility of system status



Match between system and the real world

User control and freedom



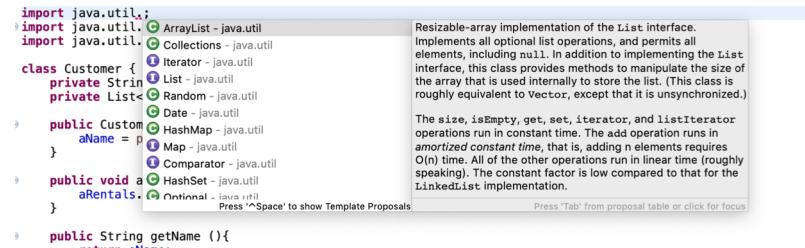
Consistency and standards

Usability Heuristics for User Interface Design

Error prevention



Recognition rather than recall



Flexibility and efficiency of use

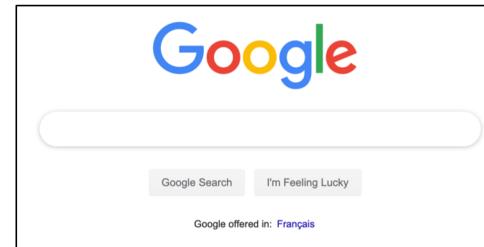
Common Shortcuts

Add Action	Return
New Window	⌘N
Synchronize with Server	⌃⌘S
Clean Up	⌘K
Planning Mode	⌘1
Context Mode	⌘2
Inbox	⌘3
Quick Entry	⌃⌘Space

Quick Entry's shortcut can be customized in Preferences

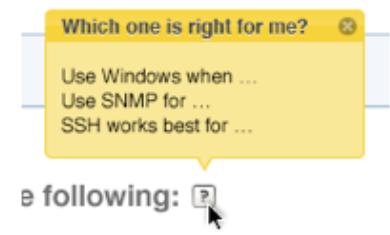
Usability Heuristics for User Interface Design

Aesthetic and minimalist design



Help users recognize, diagnose, and recover from errors

Help and documentation

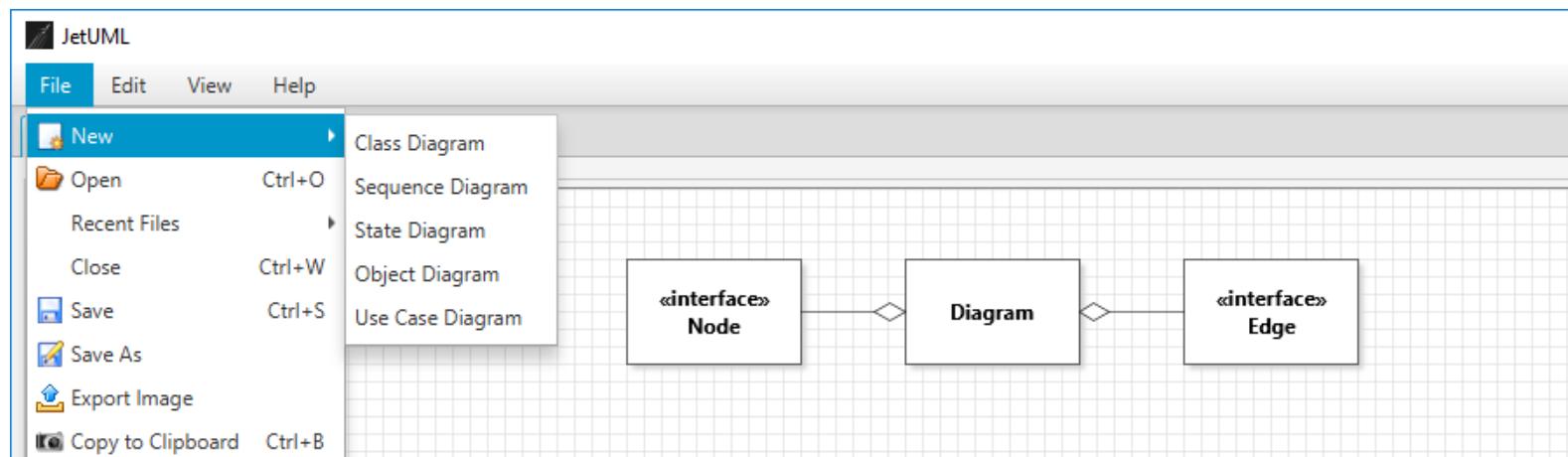


Or start a new account

A screenshot of a registration form. The form fields include "Choose a username (no spaces)" with the value "bert", "Choose a password" with the value "***", "Retype password" (empty), "Email address (must be real!)" with the value "not an email", and a checkbox "Send me occasional Digg updates." which is checked. To the right of the form, there are three error messages in red boxes with exclamation marks: 1) "bert is already taken. Please choose a different username.", 2) "Passwords must be at least 6 characters and can only contain letters and numbers.", and 3) "The email provided does not appear to be valid".

Case Study - JetUML

- Evaluate JetUML tool using the Usability Heuristics.
- Finish the activity form.



Why API Usability

- APIs provide a mechanism for code reuse.
- Nearly every line of code most programmers write will use API calls.
- API for accessing system resources
- API for Web services

Why API Usability

- APIs are difficult to use.

Causes includes the semantic design of the API, the level of abstraction, the quality of the documentation, error handling, and unclear preconditions and dependencies

Resulting bugs and/or significant security problems.



Why API Usability

- “Programmers Are Users Too”
- Developer Experience Matters

API users were between 2.4 and 11.2 times faster when a method was on the expected class, rather than on a different class.

Stylos, J. and Myers., B.A. The implications of method placement on API learnability. FSE 2008. ACM

API Designers in the Field

- In a recent study, only four of 24 professional API designers have learned anything about API design in school
- API designers must have a strong understanding of software engineering, but also have the ability and personal drive to stay focused on their user's perspective as their primary goal

Murphy, L., Kery, M.B., Alliyu, O., Macvean, A. and Myers, B.A., 2018, October. API Designers in the Field: Design Practices and Challenges for Creating Usable APIs. VL/HCC 2018. IEEE.

User Centered Methods during API Design

Usability Studies

Design Reviews

Heuristic Evaluations

Visibility of system status

It should be easy for the API user to check the state (such as whether a file is open or not), and mismatches between the state and operations should provide appropriate feedback (such as writing to a closed file should result in a helpful error message);

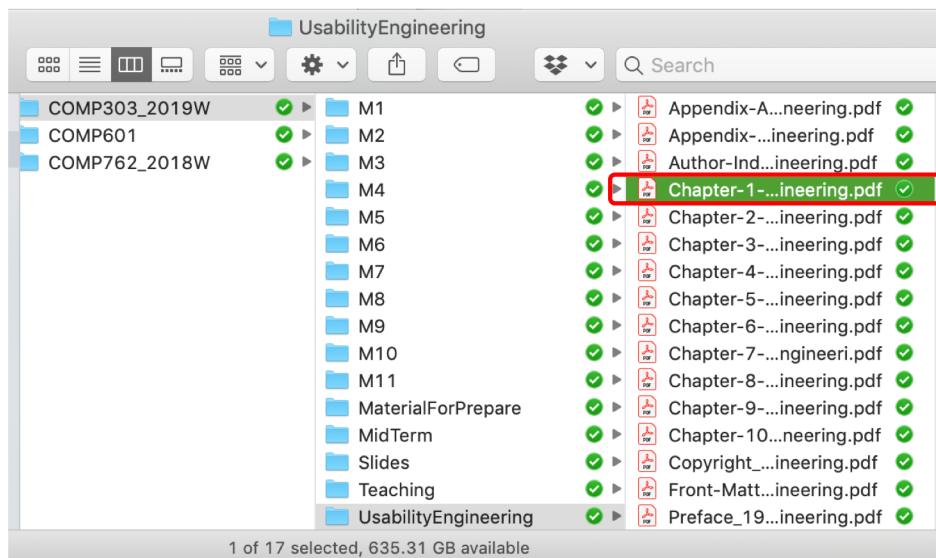
Match between system and the real world

Names given to methods and the organization of methods into classes should match the API users' expectations.

Match between system and the real world

java.io

public class **File** extends [Object](#) implements [Serializable](#), [Comparable<File>](#)



An abstract representation of file and directory pathnames. User interfaces and operating systems use system-dependent *pathname strings* to name files and directories. This class presents an abstract, system-independent view of hierarchical pathnames.

User control and freedom

API users should be able to abort or reset operations and easily get the API back to a normal state

Consistency and standards

All parts of the design should be consistent throughout the API.

Consistency and standards

javax.xml.stream

interface **XMLStreamWriter**

The XMLStreamWriter interface specifies how to write XML.

```
void writeEmptyElement( String namespaceURI, String localName)  
throws XMLStreamException
```

Writes an empty element tag to the output

```
void writeEmptyElement( String prefix, String localName, String namespaceURI)  
throws XMLStreamException
```

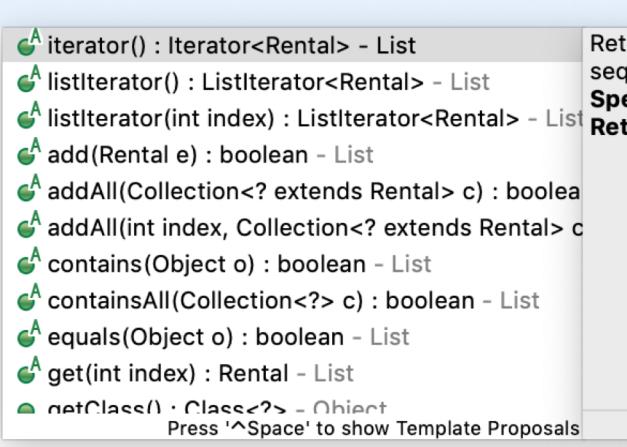
Error Prevention

The API should guide the user into using the API correctly, including having defaults that do the right thing.

Recognition rather than recall

Make the names clear and understandable, enabling users to recognize which element they want.

```
public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Iterator<Rental> rentals = aRentals.
```



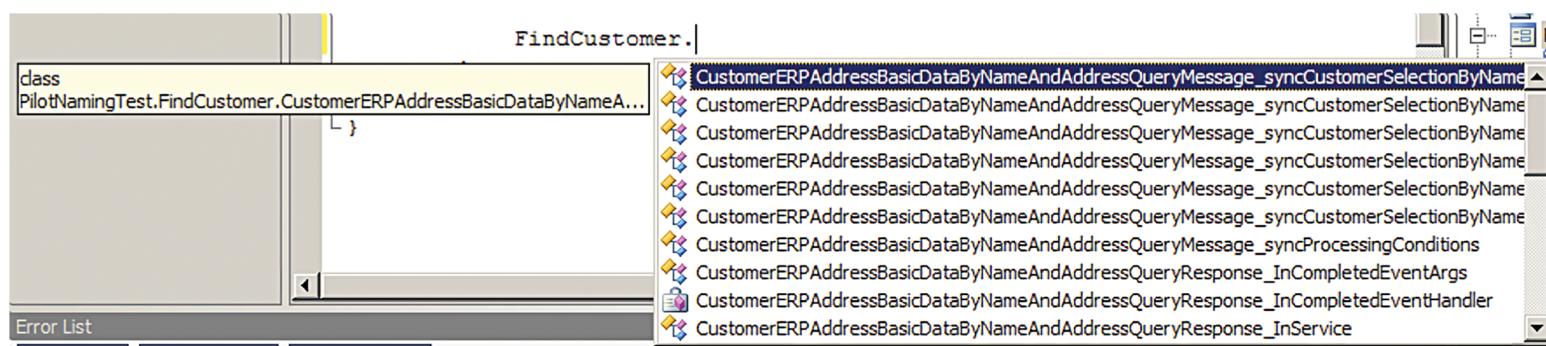
A screenshot of an IDE's code completion feature. A tooltip window is open over the variable 'rentals'. It lists several methods starting with 'A' (Abstract), each with a green circular icon and a brief description:

- iterator() : Iterator<Rental> - List
- listIterator() : ListIterator<Rental> - List
- listIterator(int index) : ListIterator<Rental> - List
- add(Rental e) : boolean - List
- addAll(Collection<? extends Rental> c) : boolean
- addAll(int index, Collection<? extends Rental> c)
- contains(Object o) : boolean - List
- containsAll(Collection<?> c) : boolean - List
- equals(Object o) : boolean - List
- get(int index) : Rental - List
- getClass() : Class<?> - Object

At the bottom right of the tooltip, the text 'Press 'Space' to show Template Proposals' is visible.

Recognition rather than recall

Make the names clear and understandable, enabling users to recognize which element they want.



Myers, B.A. and Stylos, J., 2016. Improving API
usability. *Communications of the ACM*, 59(6), pp.62-69.

More heuristics

Flexibility and efficiency of use

Aesthetic and minimalist design

Help users recognize, diagnose, and recover from errors

Help and documentation

Final words

- Usability is a key quality attribute of API and should be evaluated as other quality attributes.
- When designers decide usability must be compromised in favor of other goals, this decision will be made knowingly, and appropriate mitigations will be put in place.