

# SWE 632 - Design & Development of User Interfaces

---



George Mason  
University

---

Instructor:  
Dr. Kevin Moran

Teaching Assistant:  
David Gonzalez Samudio

Class will start in:  
**20:00**

# SWE 632 - Design & Development of User Interfaces

---



George Mason  
University

---

Instructor:  
Dr. Kevin Moran

Teaching Assistant:  
David Gonzalez Samudio

Class will start in:  
**20:00**

# SWE 632 - Design & Development of User Interfaces

Fall 2020



George Mason  
University

---

Dr. Kevin Moran

*Week 10:*  
Preventing  
Error





# Administrivia

- Midterm Exam Resubmissions Returned
- Project Checkpoint 5 due today
- Project Checkpoint 6 out now, **Due Nov 17th**
- Discussion Question 10 - Posted after class
  
- **No class** or **discussion question** next week
  - *Make sure you vote!*



# Project Checkpoint 4 - Defining Tasks

- ***Suggestions for Usability Tasks:***
  - Should strike a balance, not too specific, not too general
  - Should have a clear goal for the user (e.g. “purchase an item from the online store”, not “shop in the online store”)
  - Can include a “preamble” or background information.
    - For example, if you need to educate a user, or set the scene.



# Project Checkpoint 6: Interaction Design Critique & Iteration

## Description

In this Project Checkpoint, you will critique the interaction design of your app based on the principles in the Lectures from Week 9, 10, and 11 to identify 7 interaction design weaknesses and then revise your interaction design and app to address these weaknesses.

1. You should provide separate URLs for (1) the previous HW5 version of your web app and (2) the new, updated HW6 version of your web app.
2. Critique the HW5 version of your web app using the principles and guidelines outlined in the Site Design, Interaction Techniques, and Preventing Error lectures and readings, identifying 7 weaknesses of the interaction design. For each weakness, provide a (1) representative screenshot of the issue and (2) a one paragraph explanation of the weakness referencing the underlying principle, guideline, or theory. Across the 7 weaknesses you identify, you should make use of at least 5 distinct principles, guidelines, and theories.
3. Address the interaction design weaknesses by revising the interaction design for your web app. For each weakness, provide (1) a representative screenshot of the new design and (2) a short paragraph describing the change you made and how it addresses the issue.
4. In grading your assignment, we will evaluate the (1) degree to which the identified weaknesses follow from the interaction design principles, guidelines, and theories and (2) the effectiveness with which the changes address the identified issues.



# Class Overview

1. Human Error: Understanding why Humans Make Mistakes
2. Designing for Error: Designing to Help Prevent Error
3. Direct Manipulation: Acting “Physically” upon objects
4. 7 Minute Break
5. Group Activity: Designing a Direct Manipulation App
6. Socket.io: Hasan, Sabiha, and Safwat
7. Ember.js: Krishna, Sashank, Charles

# Human Error





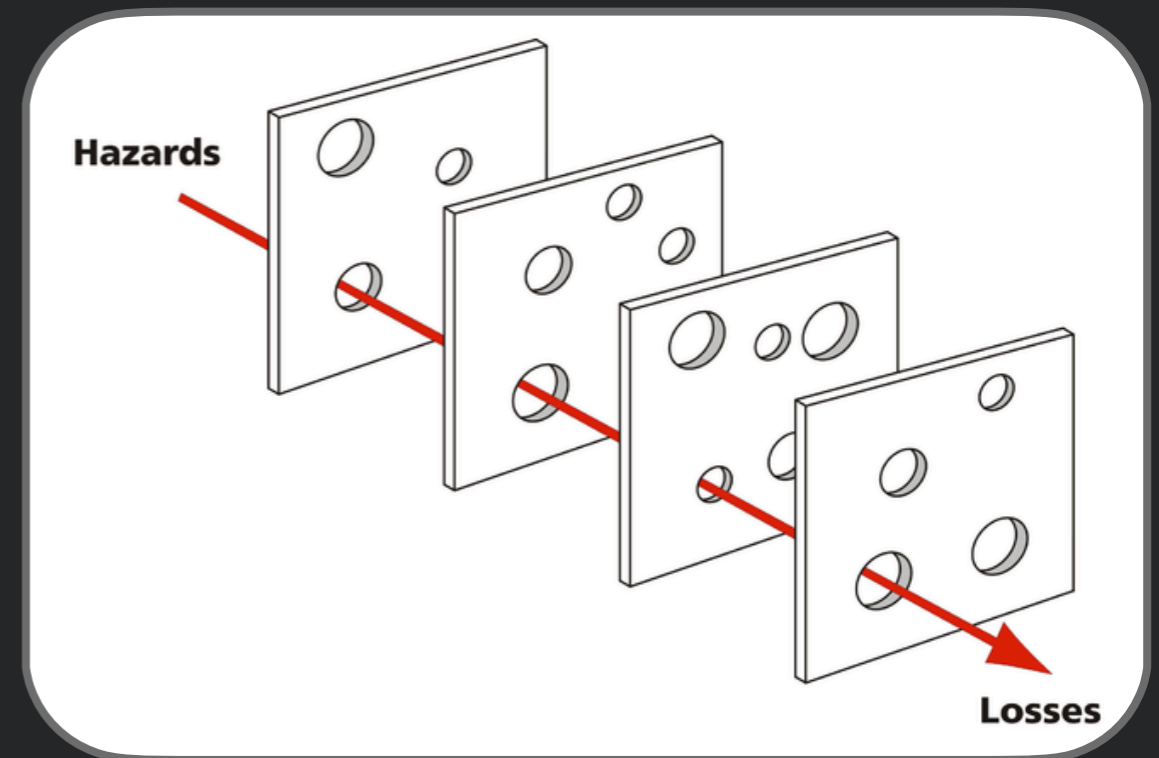


# What Causes Disasters?

- Mechanical malfunction?
- Poor design?
- Human error?

# Swiss Cheese Model

- Accidents must penetrate levels of system defenses
- Reduce accidents by
  - Adding more layers
  - Reduce the size and number of holes
  - Alert users when holes line up





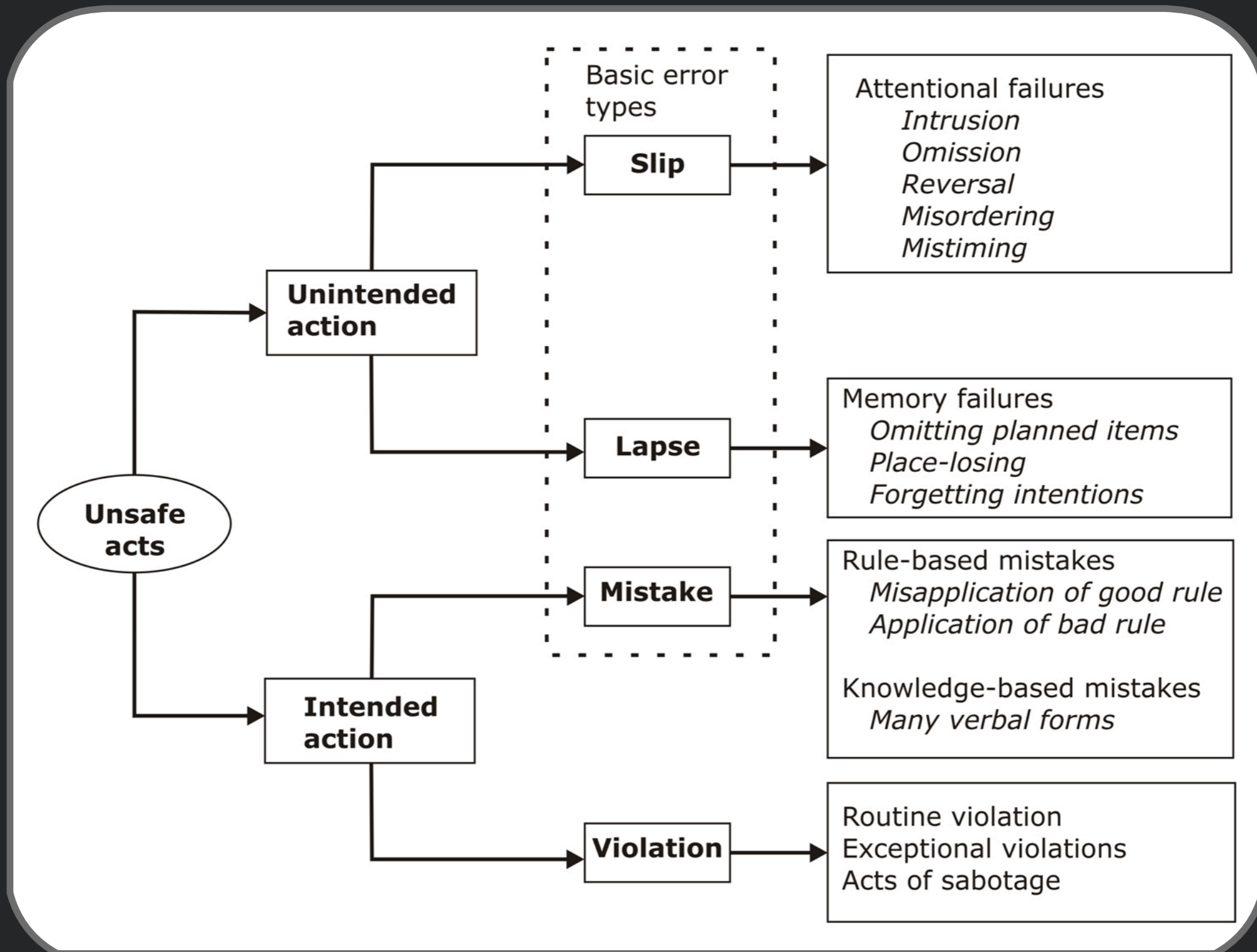
# Root Cause Analysis

- Keep asking *why* to determine causes for erroneous actions, and the causes of these causes
- Example
  - 2010 F-22 crash that killed pilot
  - Official cause: pilot error - pilot failed to take corrective action
  - Why did the pilot not take the action?
    - Pilot was not receiving oxygen and was probably unconscious.

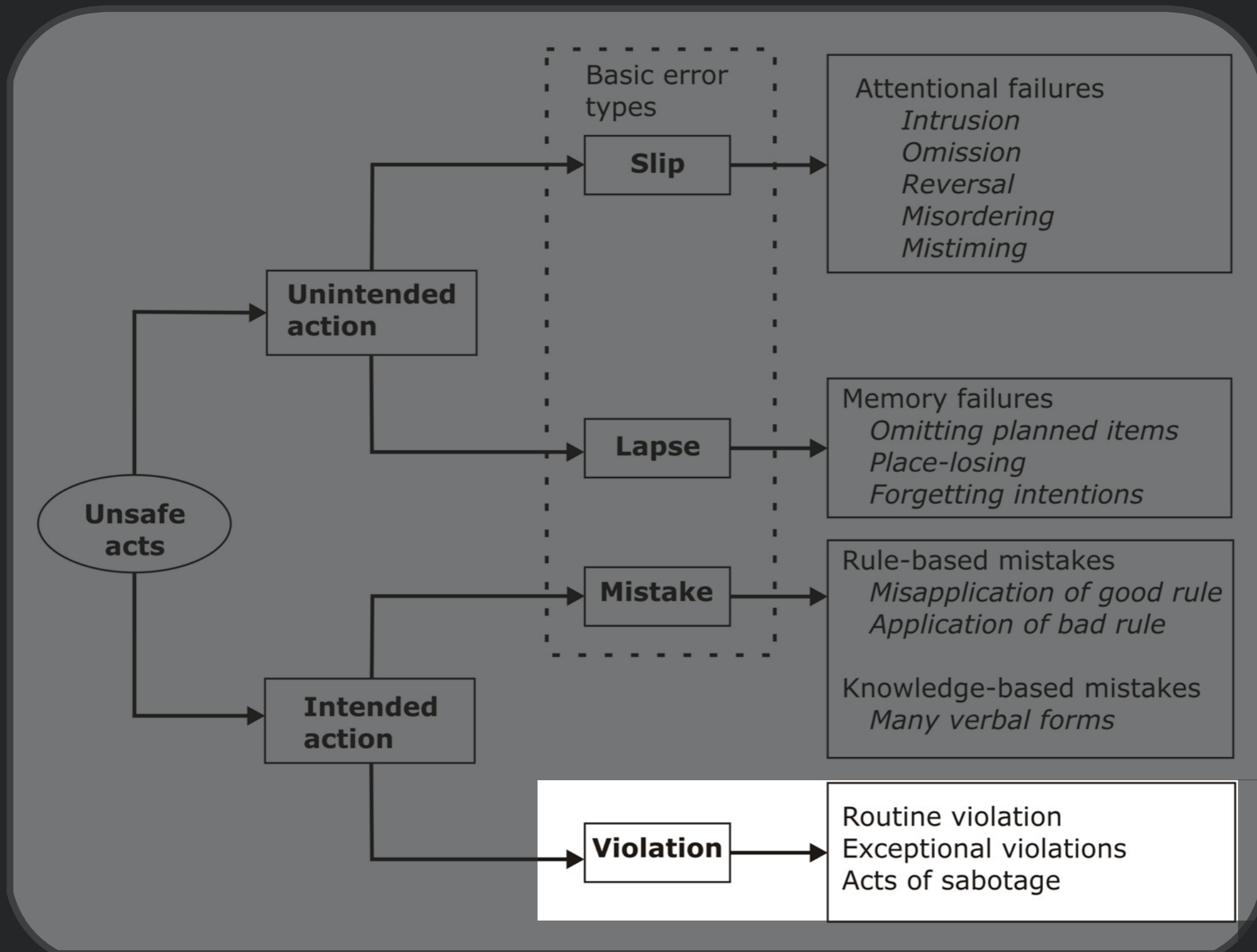
# Psychological Types of Unsafe Acts



# Psychological Types of Unsafe Acts



# Psychological Types of Unsafe Acts

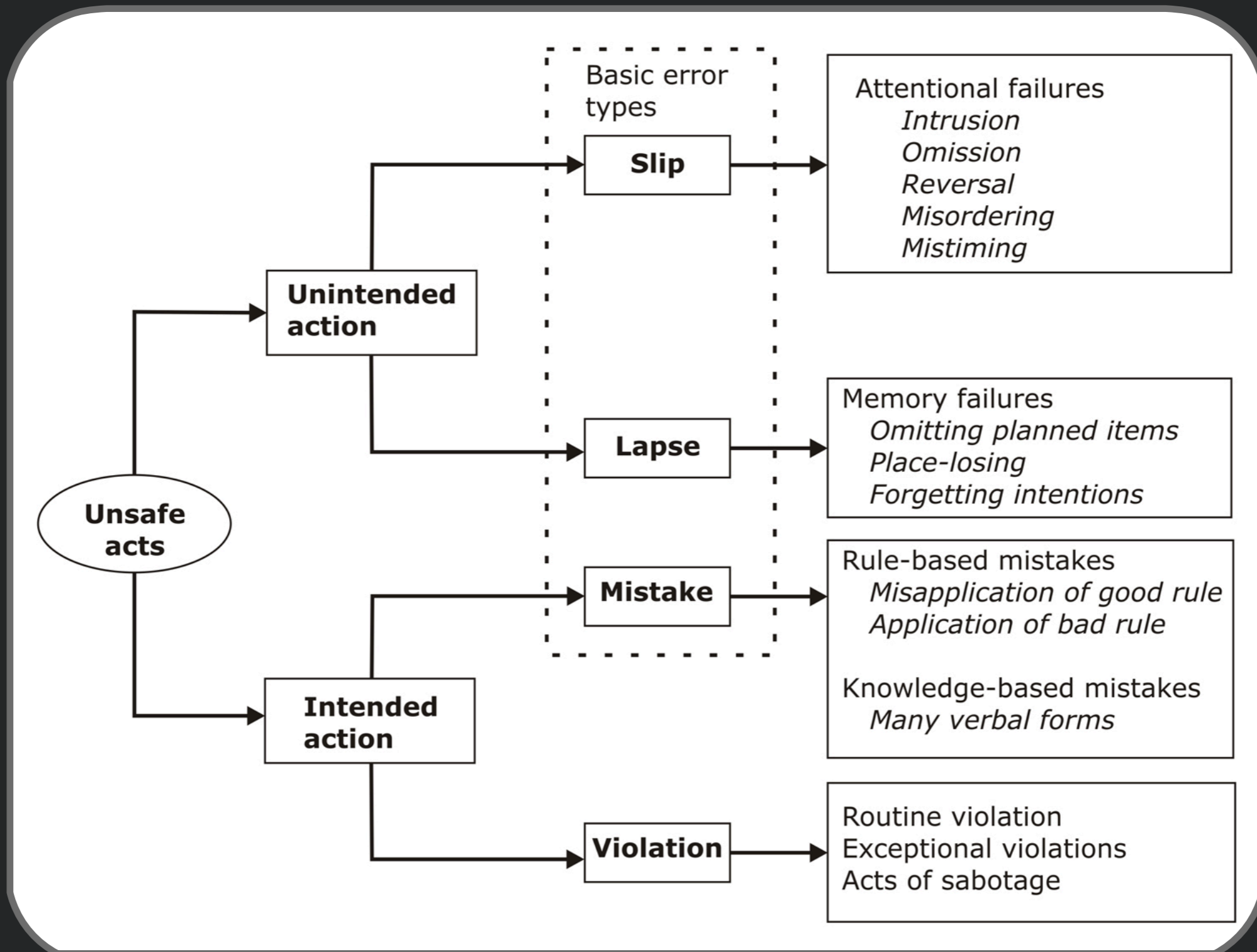




# Deliberate Violations

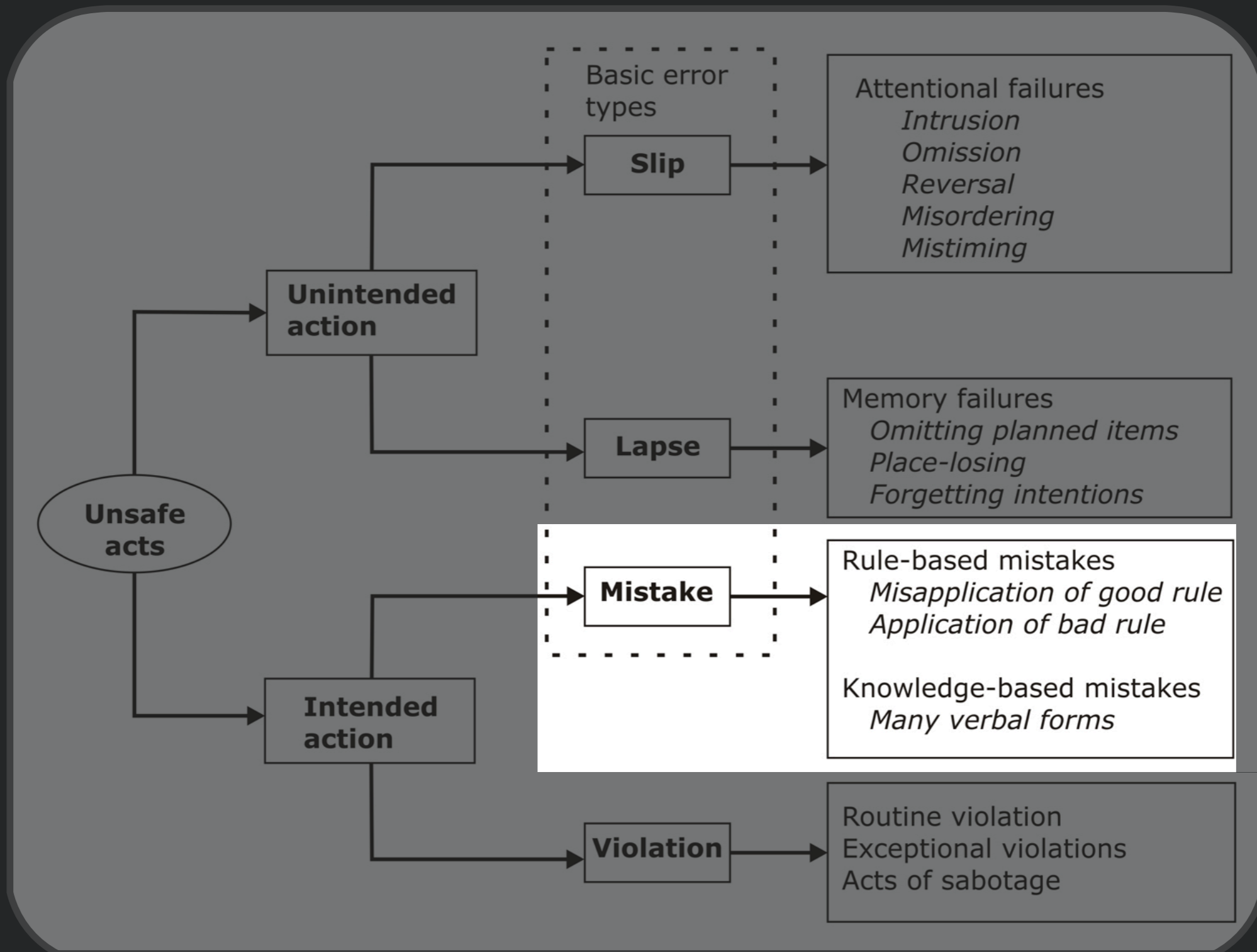
- Error occurred because user *intended* the erroneous output
- Routine violation - user always intends to do it
  - Noncompliance is so frequent it is ignored
  - E.g., running a red light
- Exceptional - only in some cases
- Sabotage - intended destruction

# Psychological Types of Unsafe Acts





# Psychological Types of Unsafe Acts

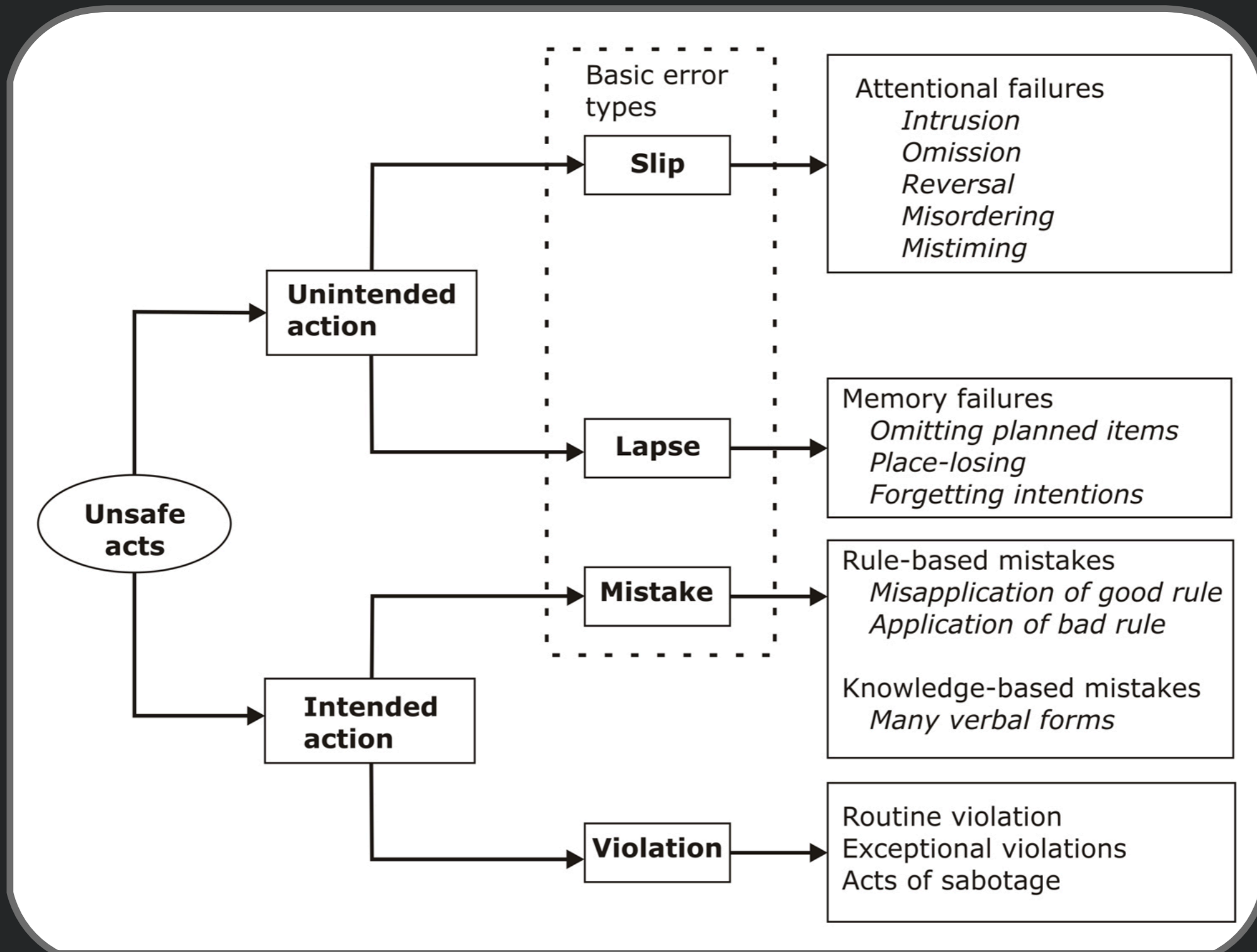




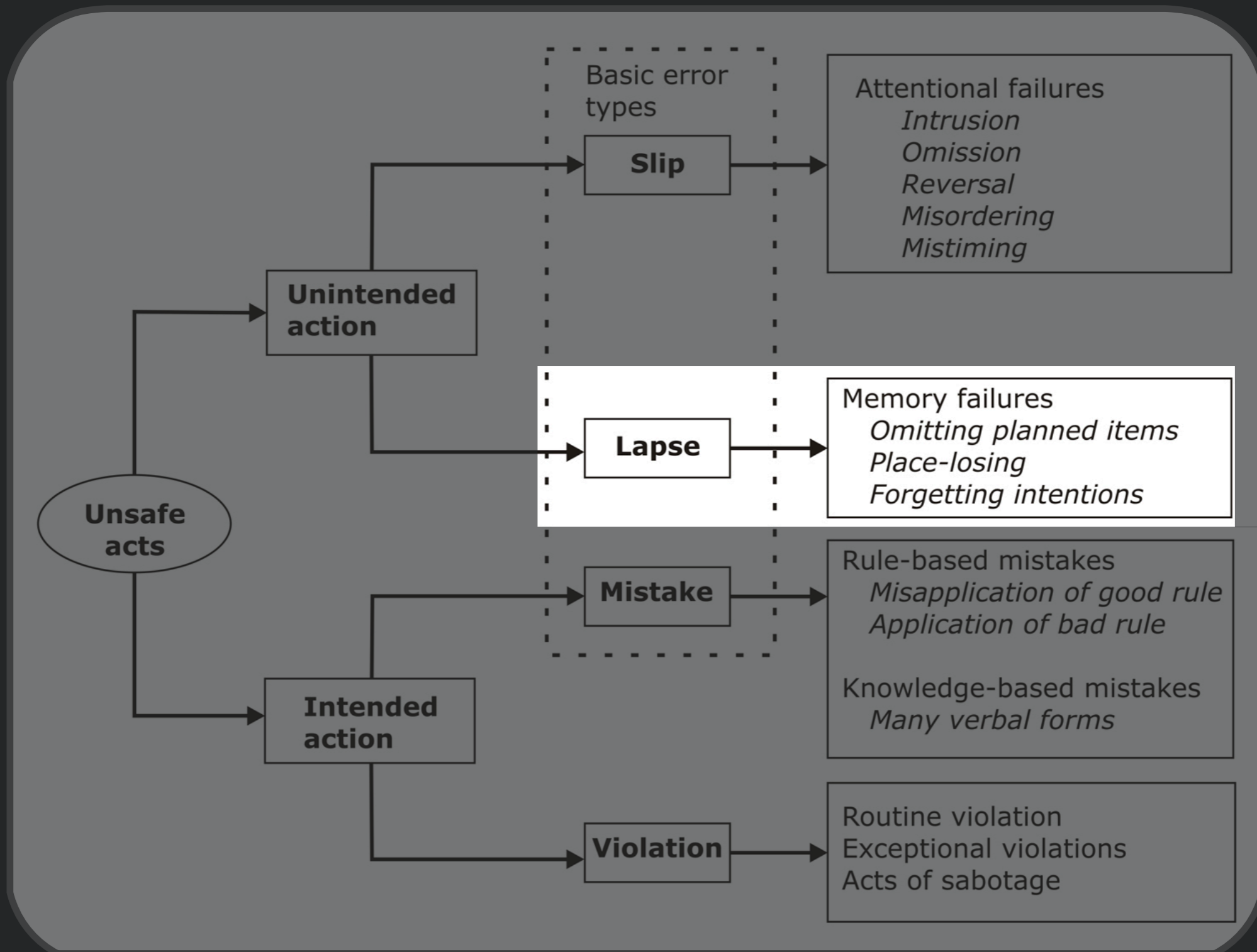
# Mistakes

- User formulated the wrong goal or plan
  - Executing action will not achieve goal
- Rule based: appropriately diagnosed situation, but chose erroneous course of action
  - Example: Night club attendees blocked from leaving during fire because bouncers thought they were breaking rules
- Knowledge based: does not have correct information
  - Example: Skidding driver feels brake vibrations, believes indicates malfunctioning breaks and takes foot off break, stopping ABS

# Psychological Types of Unsafe Acts



# Psychological Types of Unsafe Acts

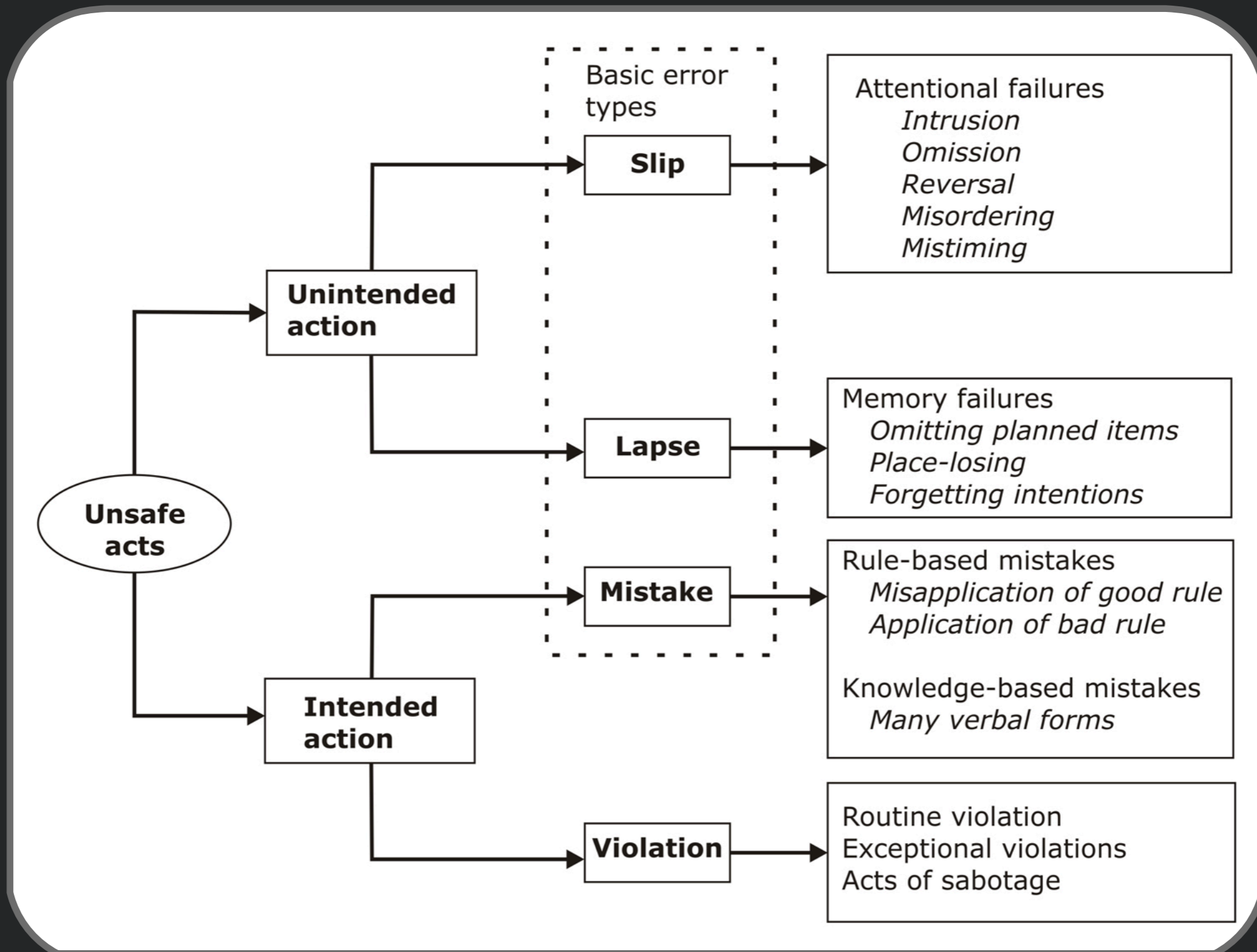




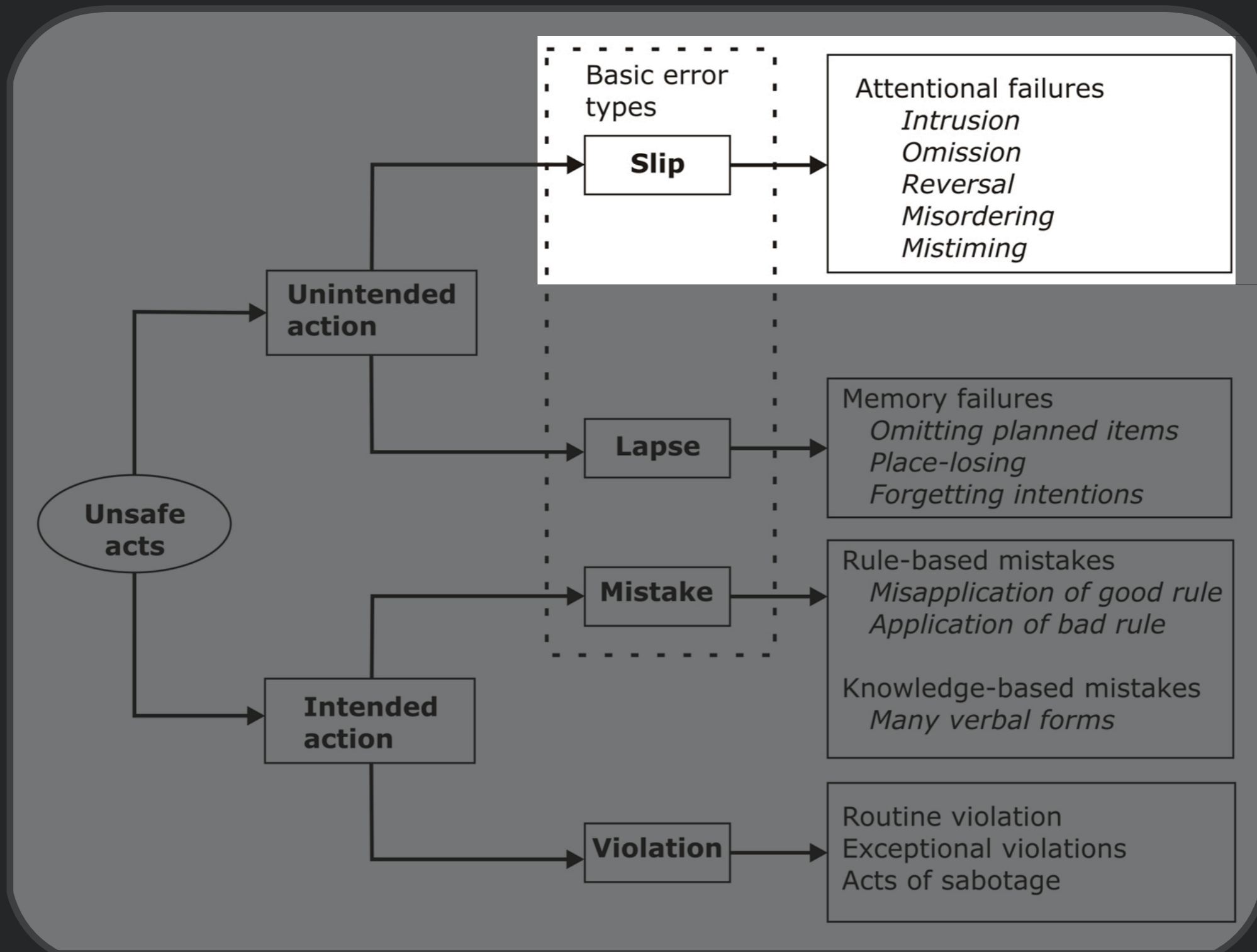
# Memory Lapse

- Failing to do all steps of a procedure, repeating steps, forgetting the outcome of an action, forgetting the goal or plan
- Often caused by interruption
  - Time between when plan was formulated and plan was executed leads to forgetting plan
  - Take a pen out to sign form, get interrupted talking to someone, leave it on desk rather than put it back in bag

# Psychological Types of Unsafe Acts



# Psychological Types of Unsafe Acts

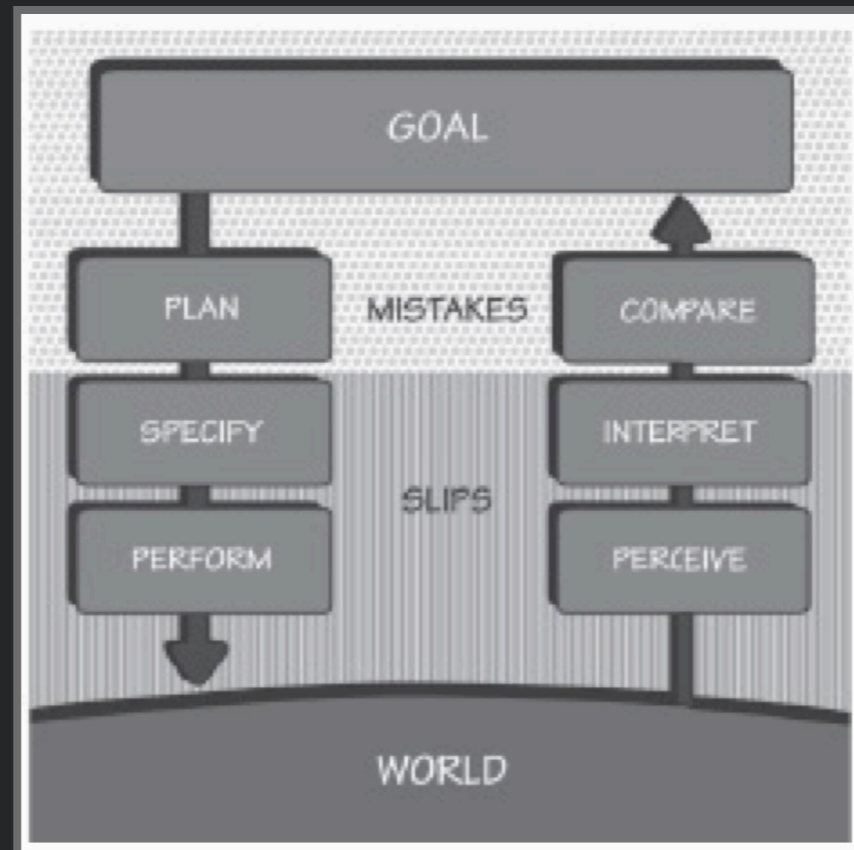


- Attentional failure - user *intended* to do correct action, but did not actually execute action
- Example: I poured some milk into my coffee and then put the coffee cup into the refrigerator. This is the correct action applied to the wrong object.





# Error & the Seven Stages of Action



- Novices are more likely to make mistakes than slips, and experts are more likely to make slips.



# Potential Underlying Causes

- Strong Habit Intrusion
- Omissions
- Perpetual Confusion
- Mistimed Checks

# Strong Habit Intrusion

- Performance of some well-practiced activity in familiar surroundings
- Intention to depart from custom
- Failure to make an appropriate check
- Example: start trip to frequent destination, forget going somewhere else



# Omissions

- May be interrupted, forgetting intention to act
- “I picked up my coat to go out when the phone rang. I answered it and then went out of the front door without my coat.”





# Perceptual Confusion

- Take frequent action very often, leading to high System 1 automation
- Don't perform perceptual check to verify that System 1 action is the correct one to take
- Example: "I began to pour coffee into the sugar bowl"

# Mistimed Checks

- Highly automated System 1 activity that is interrupted
- Error in resuming activity because usually unconscious.
- Example - interrupted in the middle of tying shoes



# Mistimed Checks

- Highly automated System 1 activity that is interrupted
- Error in resuming activity because usually unconscious.
- Example - interrupted in the middle of tying shoes





# Activity: Raise your Hand





# Activity: Raise your Hand

- Think of the last unsafe act you performed in a piece of software.



# Activity: Raise your Hand

- Think of the last unsafe act you performed in a piece of software.



# Activity: Raise your Hand

- Think of the last unsafe act you performed in a piece of software.
- What was the underlying cause?

# Designing for Error



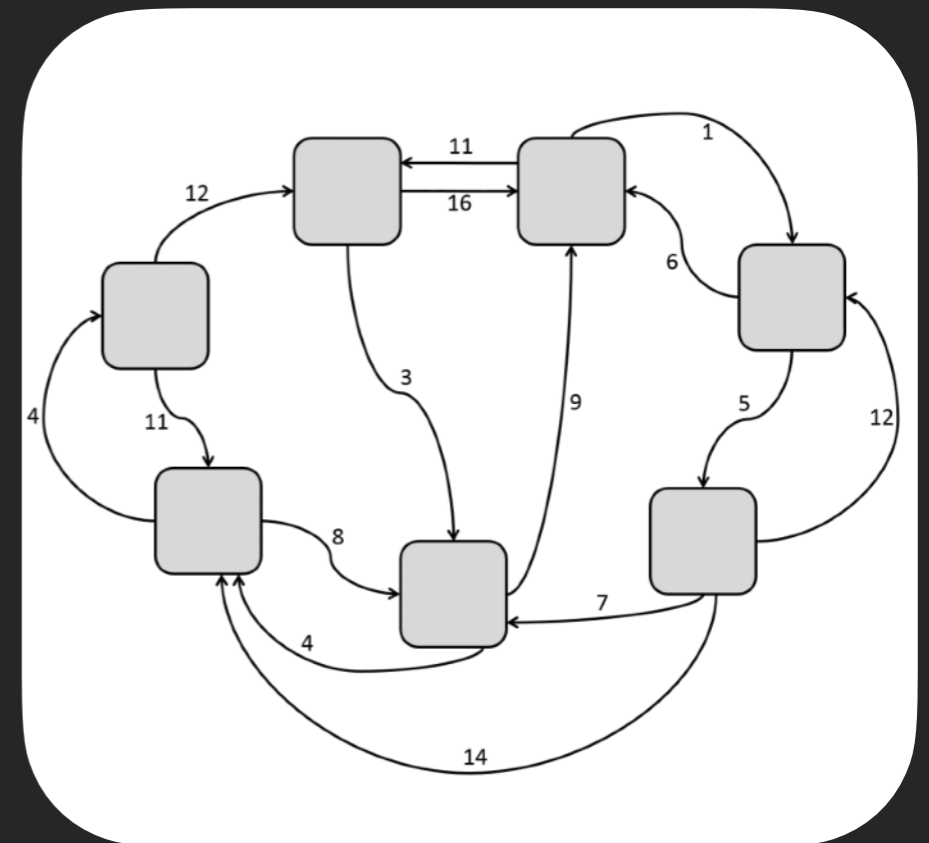


# Designing for Error

- Humans are not automatons and will never behave like automatons
- Easy to design for the situation in which everything goes well
- But important to think about what might go wrong and how the interaction design can ameliorate issues

# Information Foraging Theory Perspective

- Information Foraging Theory (IFT) perspective
  - User exploring patches topology in search of prey
  - Always making a decision about whether a patch is the right place to hunt and changing as new information arrives
- Breaks down when user actions transform the state of the application
  - Patches and topology no longer fixed
  - Visiting a configuration of the system by clicking "Send" on the email editor is a not an undoable action





# Some Strategies for Designing for Errors

- Understand the cause, and fix it
- Make it possible to reverse errors
- Offer feedback that enables users to discover and correct errors
- Don't treat actions as errors, but as manipulations



# Understand the Causes of Errors

- What errors occur? What type are they? How can they be prevented?
- Frequent contributing factors
  - Ambiguous or unclear information about the state of the system
  - Lack of an effective conceptual model
  - Inappropriate procedures
- Must design for users as they exist, rather than users as you'd like them to behave





# Interruptions

- Interruptions are a frequent cause of error
- User may be using your interface perfectly, with the correct plan to get to their goal
  - What happens if, in the middle of the task, they answer a phone call?
  - Or if they run out of time, and come back the next day?



# Designing for Interruptions

- Help user resume task, by remembering where they were in task, what steps have been completed, and what steps remain
- Reduce the number of steps
- Use forcing functions to force users to do forgettable action (e.g., take card from before picking up cash)



# Brief Activity: Interruptions

- In your project groups
- Imagine a user was interrupted while using one of your project apps
- What errors might this create?
- What challenges might users experience when resuming?
- How could you change your design to address these issues?



# Offer Feedback for User Actions

- Feedback helps keep users on track in accomplishing goals
  - Provide feedback early
  - Provide feedback consistently
- Make feedback visible, noticeable, legible, located w/ in users focus of attention
- Requesting confirmation can be used to prevent costly errors (but use sparingly)



# Tone of Feedback

- Establishes relationship with user
- Important not to take user feel “stupid”
- Make the system take blame for errors
- Be positive, to encourage
- Provide helpful messages, not cute messages
- Avoid violent, negative, demeaning, threatening terms (e.g., illegal, invalid)

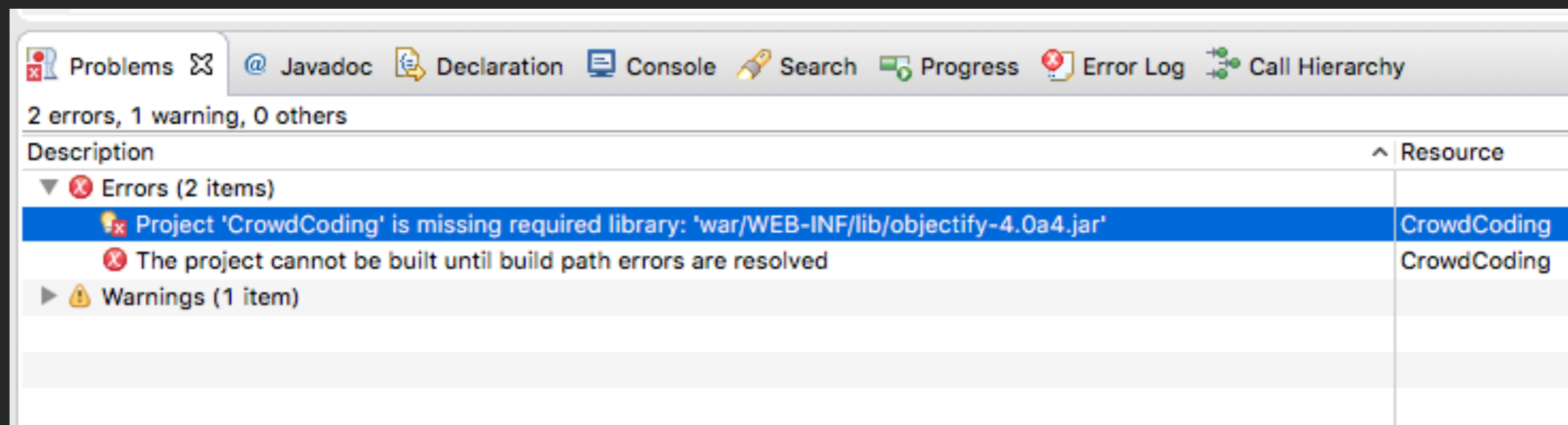


# System Response Times

- 0.1second - reacting *instantaneously*
  - requiring no special feedback except displaying result
  - limit for direct manipulation of objects in UI
- 1.0 second - *freely* navigating commands
  - noticeable delay, limit for keeping user's flow of thought uninterrupted
- 10 seconds - keeping users *attention*
  - limit for keeping user's attention focus in UI
  - longer delays create task breaks
- [Nielsen, Usability Engineering, 1993]

# Show Users How to Fix Errors

- Good: detecting user errors
- Better: directly showing how errors can be fixed
- (Best: using constraints to prevent errors from ever occurring)





# Adding Constraints to Block Errors

- Add specific constraints on actions
- e.g. forcing formatting in form fields
- Separate controls/fields so that those which are easily confused are far apart
- Separate items into different screens or modules





# Undo

- Having an option to undo actions is one of the most powerful mechanisms to mitigate errors.
- However, this is not always possible, e.g. sending an email.



# Norman's Key Design Principles

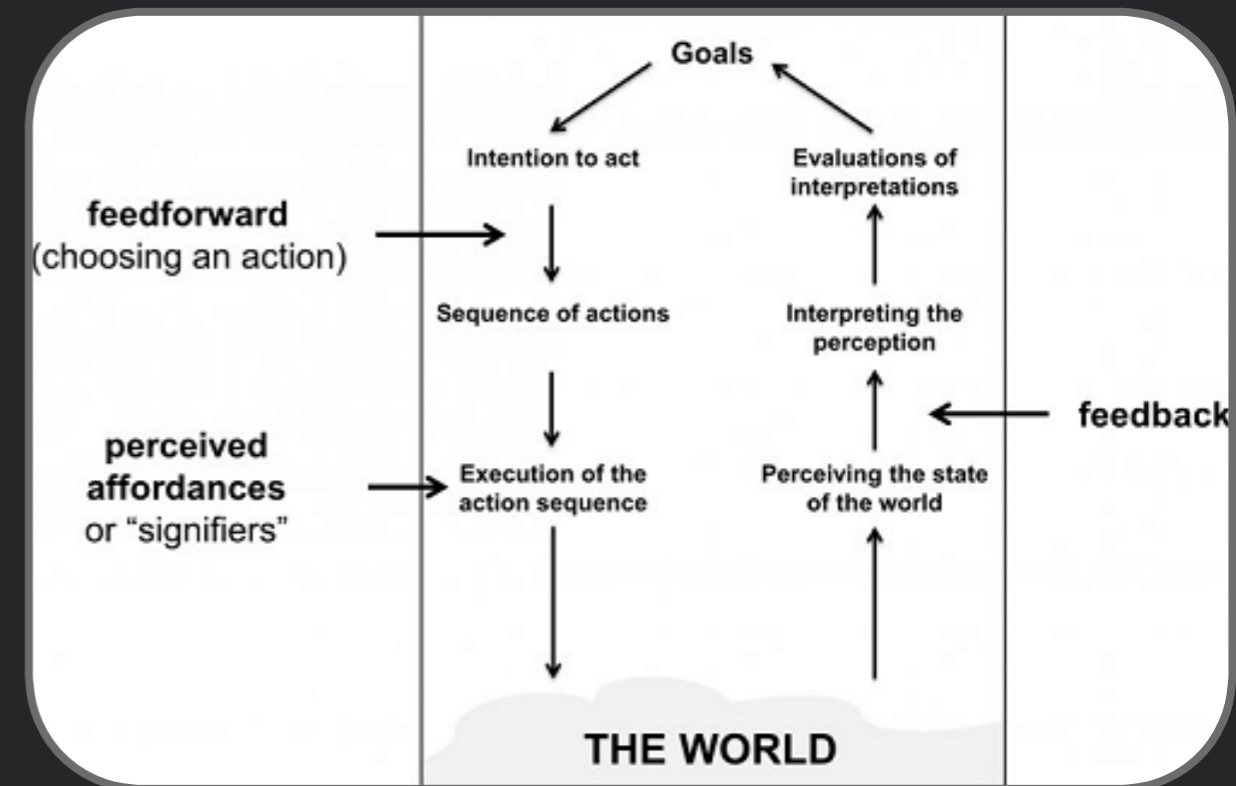
1. Put the knowledge required to to operate the technology in the world
2. Use the power of natural and artificial constraints
3. Bridge the two Gulfs: the Gulf of Execution and the Gulf of Evaluation
  - Execution: Make options readily available
  - Evaluation: Provide Feedback

# Direct Manipulation



# Motivation

- User is trying to do a task, manipulating a [model] of world
- Hard to plan out long sequence of actions in advance
- Gulf of execution: hard to know if took correct action
- Gulf of evaluation: hard to understand if successfully manipulated world
- Hard to compare hidden world to desired world



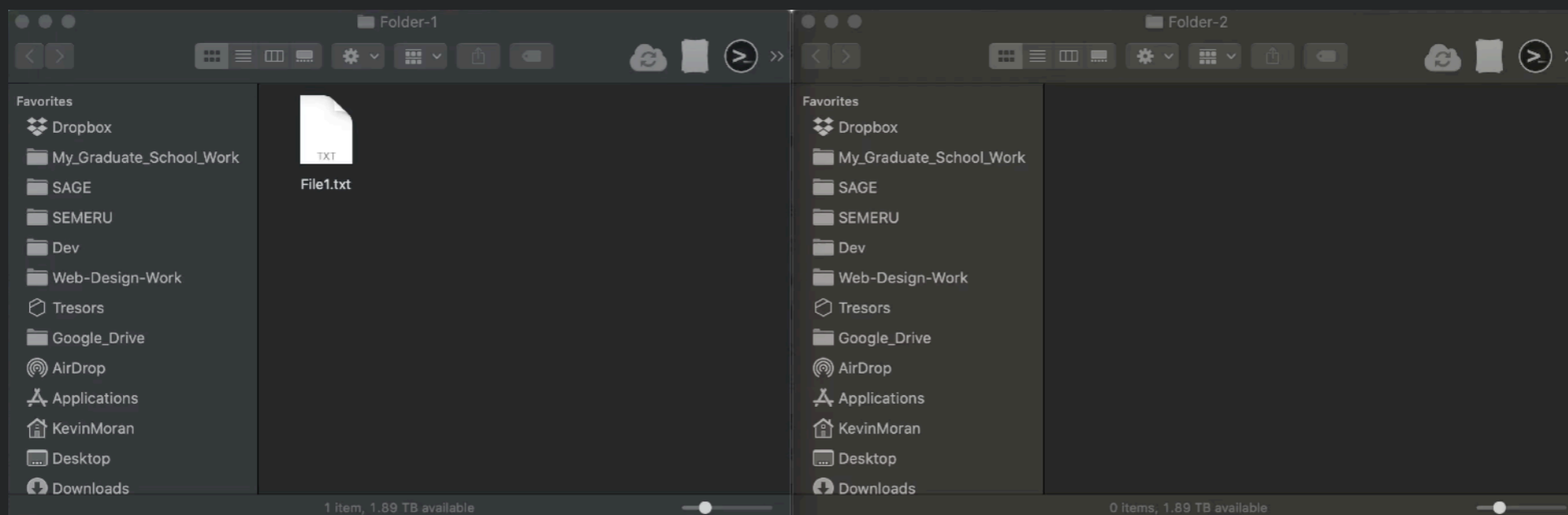
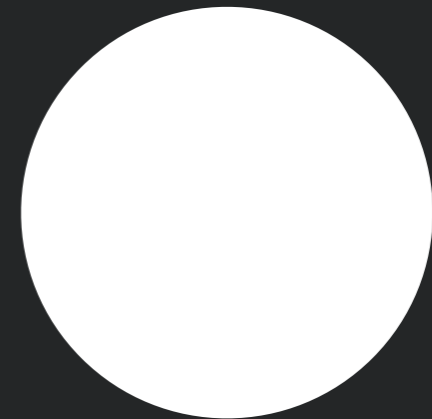
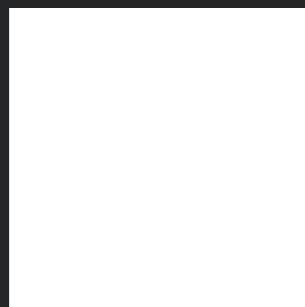


# Key Questions

- What is the cost of an error?
  - Is it low cost or high cost?
  - Is it undoable?
- What feedback is necessary for user to realize the system is not in the desired state?

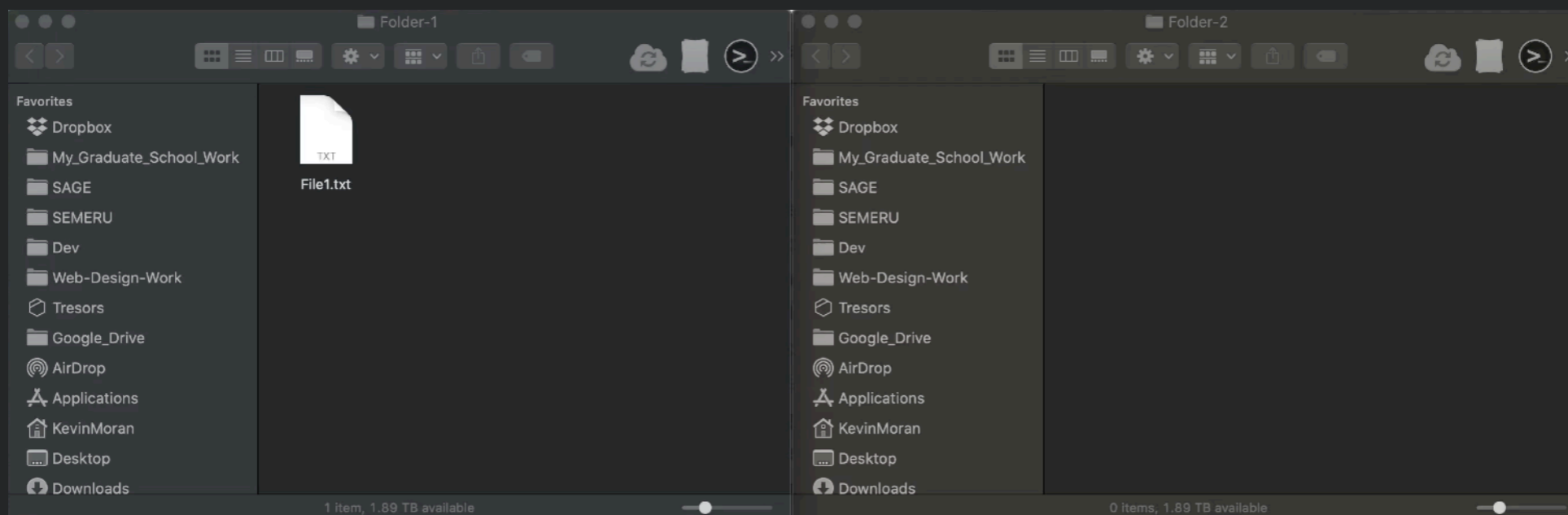
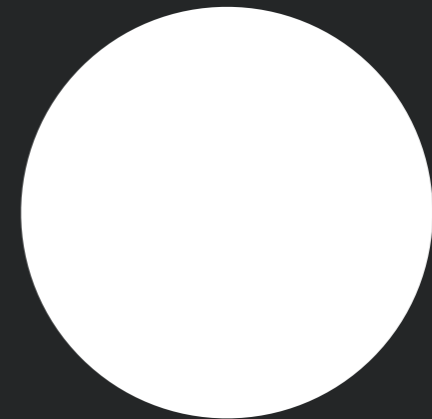
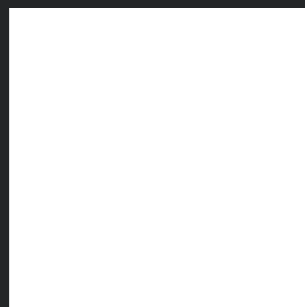
# Direct Manipulation

- “Rapid incremental reversible operations whose impact on the objects of interest is immediately visible” (Shneiderman, 1982)



# Direct Manipulation

- “Rapid incremental reversible operations whose impact on the objects of interest is immediately visible” (Shneiderman, 1982)





# Direct Manipulation Characteristics

- Continuous Representation of the Object of Interest
- Physical Actions instead of complex syntax
- Continuous feedback and reversible, incremental actions





# Benefits


- Supports exploration
  - Don't plan long sequence of actions: pick an action, try it, can change mind if want to do something else instead
- Provides immediate feedback
  - Can quickly see what outcome of actions are in manipulating the world
  - Easy to compare desired state of the world to actual state of the world

# Drawbacks

- Only a small Number of Objects on screen at once
- It can be physically demanding on the user
- Can be relatively slow
  - If the user needs to perform a large number of actions, it may be impractical
- Repetitive tasks are not well supported
  - e.g. can be better for novices to learn, but harder to experts to exploit
- Some gestures can be error prone

# Example - Kayak





**Advice: BUY** [Learn more](#) ⓘ

Create a price alert

**Stops** [Show all](#)

nonstop \$127

1 stop \$145

2+ stops \$303

---

**Times** [Show all](#)

Take-off Washington (DCA)  
Fri 2:41p – 10:30p

Take-off Chicago (CHI)  
Mon 5:30a – 10:00p

Show landing times ▾

**Airports** [Show all](#)

Depart/Return same

Washington

DCA: Reagan-Nati... \$127

BWI: Baltimore/Wa... \$207

DCA ↔ CHI 108 of 1115 flights

Dec 16 Friday ↔ Dec 19 Monday

Economy cabin 1 traveler

[Change](#)

Sort by: **Price** [Recommended](#) [Duration](#) [More](#) ▾

[Round-trip](#) | [Flight-by-flight](#)

\$207

JustFly, Experience world-class service

Click "View Deal" to find our cheapest flights

justfly.com

\$207 nonstop [View Deal](#)

www.justfly.com Ad

\$227

American Airlines 📶 🗑️ 📺

8:12p DCA → 9:26p ORD 2h 14m nonstop

3:25p ORD → 6:12p DCA 1h 47m nonstop

[View Deal](#)

Show details Economy

\$227

American Airlines 📶 🗑️ 📺

8:12p DCA → 9:26p ORD 2h 14m nonstop

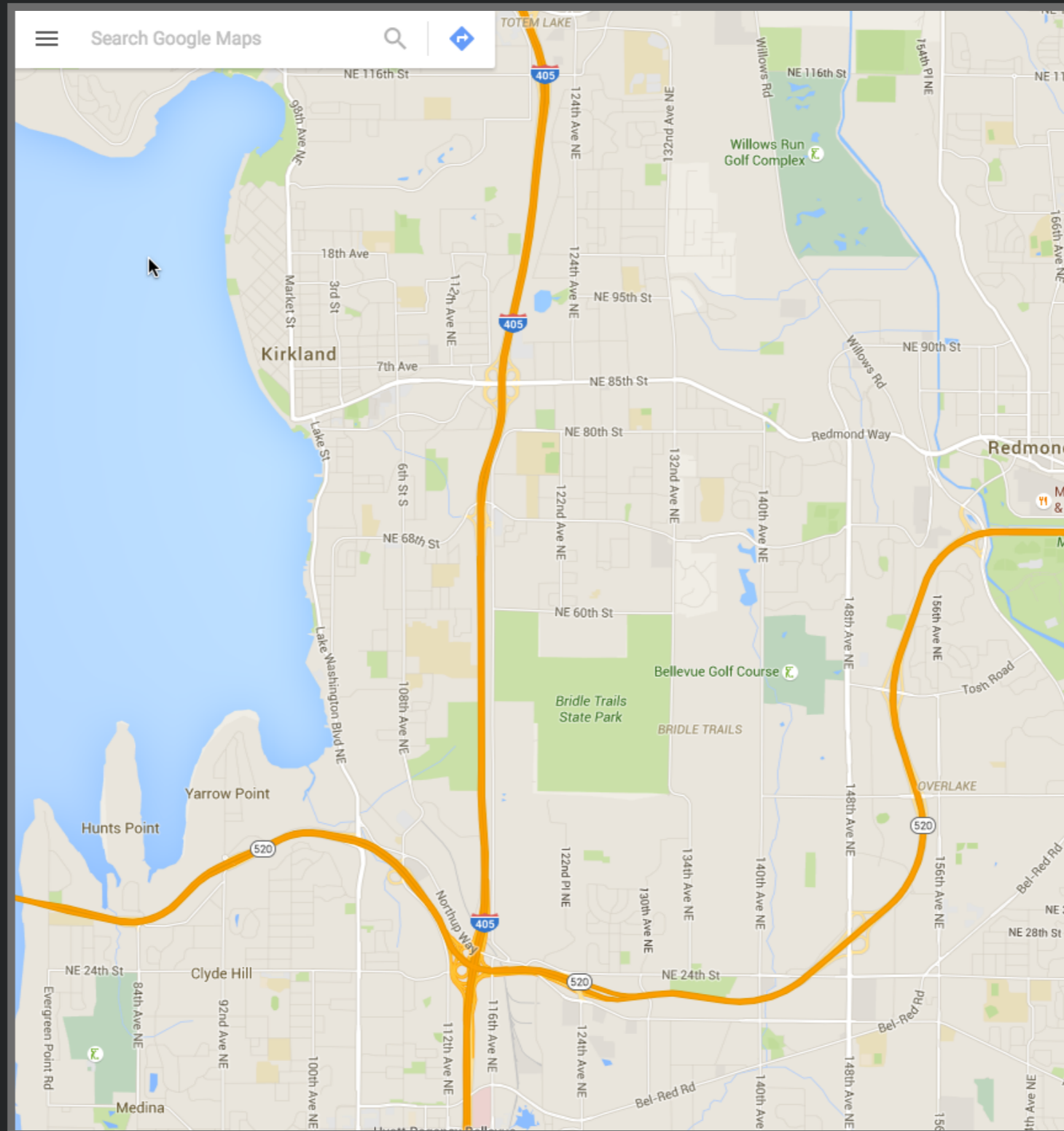
11:55a ORD → 2:42p DCA 1h 47m nonstop

[View Deal](#)

Show details Economy



# Example - Google Maps



# Example - GUI Builder



The screenshot displays the Qt Creator GUI Builder interface for a file named 'validators.ui'. The interface is divided into several panels:

- Left Panel (Toolbox):** Contains various widget and layout classes categorized into 'Layouts', 'Spacers', 'Buttons', 'Item View...I-Based', and 'Item Wi...-Based'. The 'Buttons' category is currently selected, showing options like Push Button, Tool Button, Radio Button, Check Box, Comman...Button, and Button Box.
- Center Panel (Design View):** Shows a visual representation of the GUI form. It features two validation widgets: a `QIntValidator` with a minimum value of 0 and a maximum of 1000, and a `QDoubleValidator` with a minimum of 0.00, a maximum of 1000.00, and 2 decimal places. A 'Quit' button is located at the bottom right of the form.
- Right Panel (Object Inspector):** Displays a hierarchical tree of the GUI object. The root is `ValidatorsForm` (class `QWidget`). It contains several sub-objects, including `QHBoxLayout`, `Spacer`, `Local...ector`, `QHBoxLayout`, `Spacer`, `QPushButton`, `Spacer`, `QGroupBox`, `QHBoxLayout`, `QGridLayout`, and `QLabel`.
- Bottom Panel (Signals & Slots Editor):** Shows a table of connections between the GUI elements. The connection is as follows:

Sender	Signal	Receiver	Slot
pushButton	clicked()	Valid...sForm	close()

# Example - Spreadsheets



FlyCalc - WIG2004.XLS

File Edit View Insert Format Tools ?

Formula: [ ]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
1		788	355	564	399	413	897	444	523	413							
2		800	923	233	307	864	355	90	877	864							
3		657	788	755	444	455	478	432	405	455							
4		599	866	233	201	413	361	455	233	413	Sep 2005	Oct 2005	Nov 2005	Dec 2005	Jan 2006	Feb 2006	
5		899	755	673	311	780	400	614	754	780							
6			334	953	888	214	644	789	361	978							
7		233	644	766	446	231	977	577	453	847	455	507	690	700		788	800
8		577	533	968	897	541	977	475	358	975	355	478	361	400		355	923
9											742	267	599	700		564	233
10	Bryant Park	965	365	233	708	564	344	78	359	997	352	215	836		399	307	
11	Keokuk	670	607	233	846	980	544	613	523	877	405	233	754		413	864	
12	Westport	855	732	908	556	352	315	635	413	864	455	413	780		897	355	
13	Temple	607	244	641	908	561	555	314	467	900	378	723	382		444	90	
14	Lockhart	222	645	999	182	388	905	814	444	190	432	455	614		523	877	
15	Stonington	344	756	600	481	339	489	144	399	307	444	201	311		413	864	
16																	
17	Subtotal	5455	4380	5088	5002	4521	4866	4084	4342	5687	3718	3890	5477		4796	5313	
18																	
19	U.K. Factories																
20																	
21	Clacton	855	315	908	556	352	556	635	413	864	455	413	780		980	966	
22	Penge	506	605	860	222	459	222	521	897	355	478	361	400		670	800	
23	Runcom	670	544	233	846	980	846	613	523	877	405	233	754		2242	1543	
24	Worcester Park	344	489	600	481	339	481	144	399	307	444	201	311		899	900	
25	Wapping	855	315	908	556	352	556	635	413	864	455	413	780		600	650	
26	Tooting Bec	506	605	860	222	459	222	521	897	355	478	361	400		600	670	
27	Belham	222	905	999	182	388	182	814	444	90	432	455	614		797	668	
28	Wigan	670	544	233	846	980	846	613	523	877	405	233	754		800	796	
29	Ashby de la Zouche	855	315	908	556	352	556	635	413	864	455	413	780		413	780	
30	Bude	607	555	641	908	561	908	314	467	900	378	723	382		361	400	
31	Loce	344	489	600	481	339	481	144	399	307	444	201	311		455	614	
32	Scunthorpe	674	677	790	650	666	679	677	566	756	567	685	433		900	780	
33																	
34	Subtotal	5073	4761	5982	5078	4750	5078	4433	4478	5441	3696	3233	5086		7167	7021	
35																	
36	Canadian Factories																
37																	
38	Deception Bay	344	489	600	600	481	339	521	897	355	478	361	233		846	613	
39	Mirissauqua	855	315	908	600	481	339	481	855	315	908	556	352		481	144	
40	WIG																

FlyCalc 1.1 - Copyright Natium 2003-2006



# Example: Live Programming

The screenshot shows a live programming environment with a code editor on the left and an output/debug console on the right. The code editor has tabs for JS, HTML, and CSS. The JS tab is active, showing the following code:

```
1  
2 helloWorld(-1);  
3  
4 function helloWorld(x) {  
5     if (x > 0)  
6         console.log('hello world');  
7  
8     var y = x + 1;  
9  
10  
11 }
```

The output/debug console shows the following results:

Line	Value
2	[ -1][ undefined]
5	[ false][ -1]
8	[ -1][ 0]

The screenshot shows a browser window with three panes of code and a rendered output. The top pane shows HTML code:

```
<html>  
<head>  
</head>  
<body>  
<h1>Hello</h1>  
</body>  
</html>
```

The middle pane shows CSS code:

```
h1{  
    color : gray;  
}
```

The right pane shows JavaScript code:

```
function hello(){  
    alert("Hello")  
}
```

The bottom pane shows the rendered output: **Hello**



# Example: Live Programming

The screenshot shows a live programming environment with a code editor on the left and an output/debug console on the right. The code editor has tabs for JS, HTML, and CSS. The JS tab is active, showing the following code:

```
1  
2 helloWorld(-1);  
3  
4 function helloWorld(x) {  
5     if (x > 0)  
6         console.log('hello world');  
7  
8     var y = x + 1;  
9  
10  
11 }
```

The output/debug console shows the following results:

Line	Value
2	[ -1][ undefined]
5	[ false][ -1]
8	[ -1][ 0]

The screenshot shows a browser window with three panes of code and a rendered output. The top pane shows HTML code:

```
<html>  
<head>  
</head>  
<body>  
<h1>Hello</h1>  
</body>  
</html>
```

The middle pane shows CSS code:

```
h1{  
    color : gray;  
}
```

The right pane shows JavaScript code:

```
function hello(){  
    alert("Hello")  
}
```

The bottom pane shows the rendered output: **Hello**



# Example: Edit Constants by Editing Output

The screenshot shows the sketch-n-sketch v0.4.2 IDE. On the left, a code editor displays the following code:

```
1  
2 (def [x0 y0 w h sep n]  
3   [81 124 20 90 65! 3])  
4  
5 (def boxi (\i  
6   (let xi (+ x0 (* i sep ))  
7     (rect 'lightblue' xi y0 w h))))  
8  
9 (svg (map boxi (zeroTo n))))
```

On the right, a visual output shows three light blue vertical rectangles. The first two are on the left, and the third is on the right with a small black crosshair over it, indicating it is the active element. The IDE interface includes several control buttons: [Heuristics] Fair, [Orientation] Vertical, 3 Boxes PLDI, Edit Code, Undo, Redo, [Zones] Hidden, and [Widgets] Shown. At the bottom right, a status bar shows: rect3 Interior (ACTIVE) 124{y0} 81{x0}.

Chugh et al. [PLDI '16]

# 7 Minute Break



# SWE 632 - Design & Development of User Interfaces

---



George Mason  
University

---

Instructor:  
Dr. Kevin Moran

Teaching Assistant:  
David Gonzalez Samudio

Class will start in:  
**07:00**

Welcome to Class!

# SWE 632 - Design & Development of User Interfaces

---



George Mason  
University

---

Instructor:  
Dr. Kevin Moran

Teaching Assistant:  
David Gonzalez Samudio

Class will start in:

07:00

Welcome to Class!

# In-Class Activity



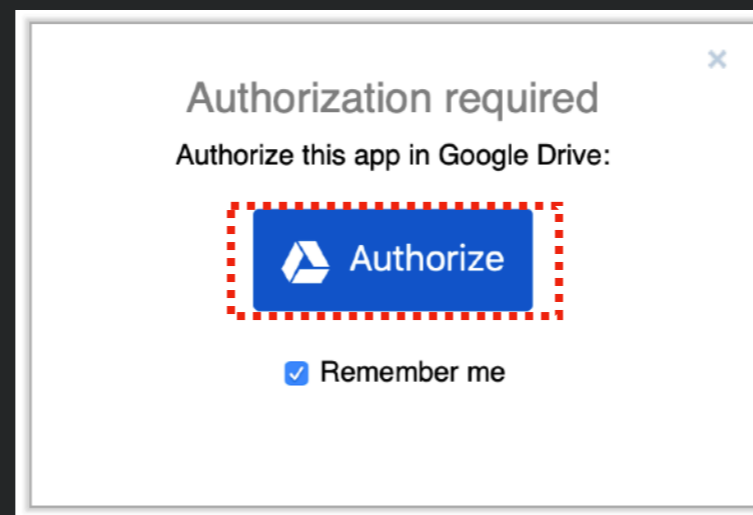
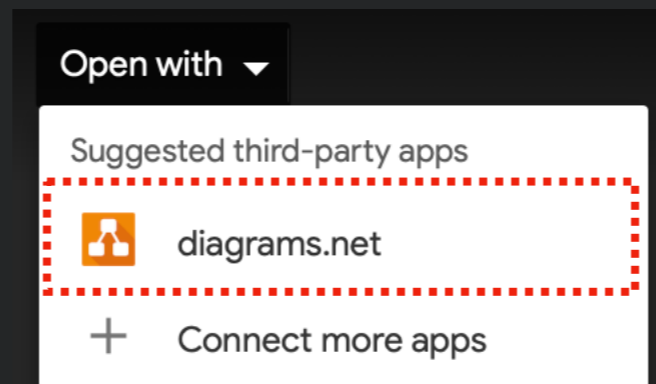
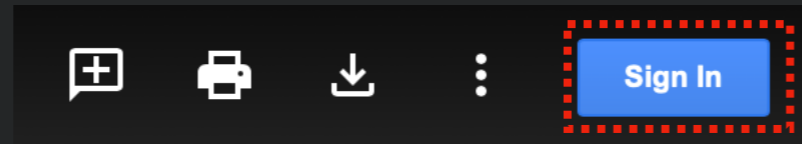
# In Class Activity: Direct Manipulation Programming



- In groups of 2
  - Design a system for writing code through direct manipulation
    - Create sketches showing key screens
    - Should support
      - Standard programming language features (variables, conditionals, loops, functions)
      - Should make it faster and easier to make code changes
      - Should make it easier to get feedback on if program exhibits intended behavior

***Deliverable:*** Sketches in the Google Drive Link

# In Class Activity: Direct Manipulation Programming Interactions



# In Class Activity: Direct Manipulation Programming



- In groups of 2
  - Design a system for writing code through direct manipulation
    - Create sketches showing key screens
    - Should support
      - Standard programming language features (variables, conditionals, loops, functions)
      - Should make it faster and easier to make code changes
      - Should make it easier to get feedback on if program exhibits intended behavior

***Deliverable:*** Sketches in the Google Drive Link





# Acknowledgements

- Slides adapted from Dr. Thomas Latoza's SWE 632 course