Bypass Testing of Web Applications

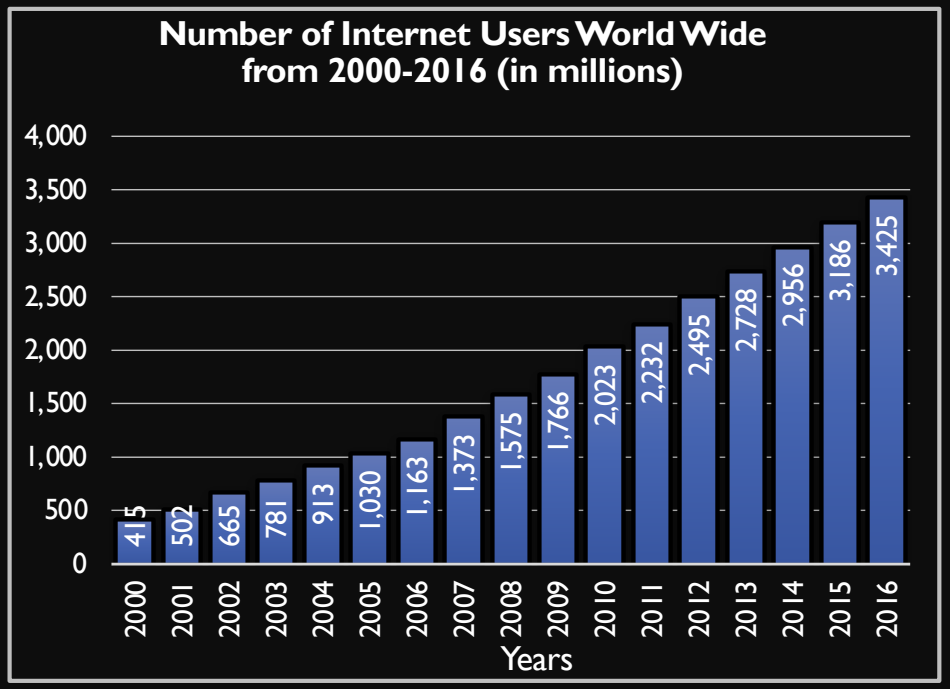
CS 3250 Software Testing

[Offutt, Papadimitrious, Praphamontripong, "A Case Study on Bypass Testing of Web Applications"]
[Based in part on SWE 642 (Software Engineering for the World Wide Web), by Offutt]
[Offutt, Wu, Du, and Huang, "Bypass Testing of Web Applications"]

Deploying Software

- Bundled – pre-installed on computer
- Shrink-wrap – bought and installed by end-users
- Contract – purchaser pays developer to develop and install
- Embedded – installed on a device
- Web – hosted on web servers, executed across the Internet through HTTP

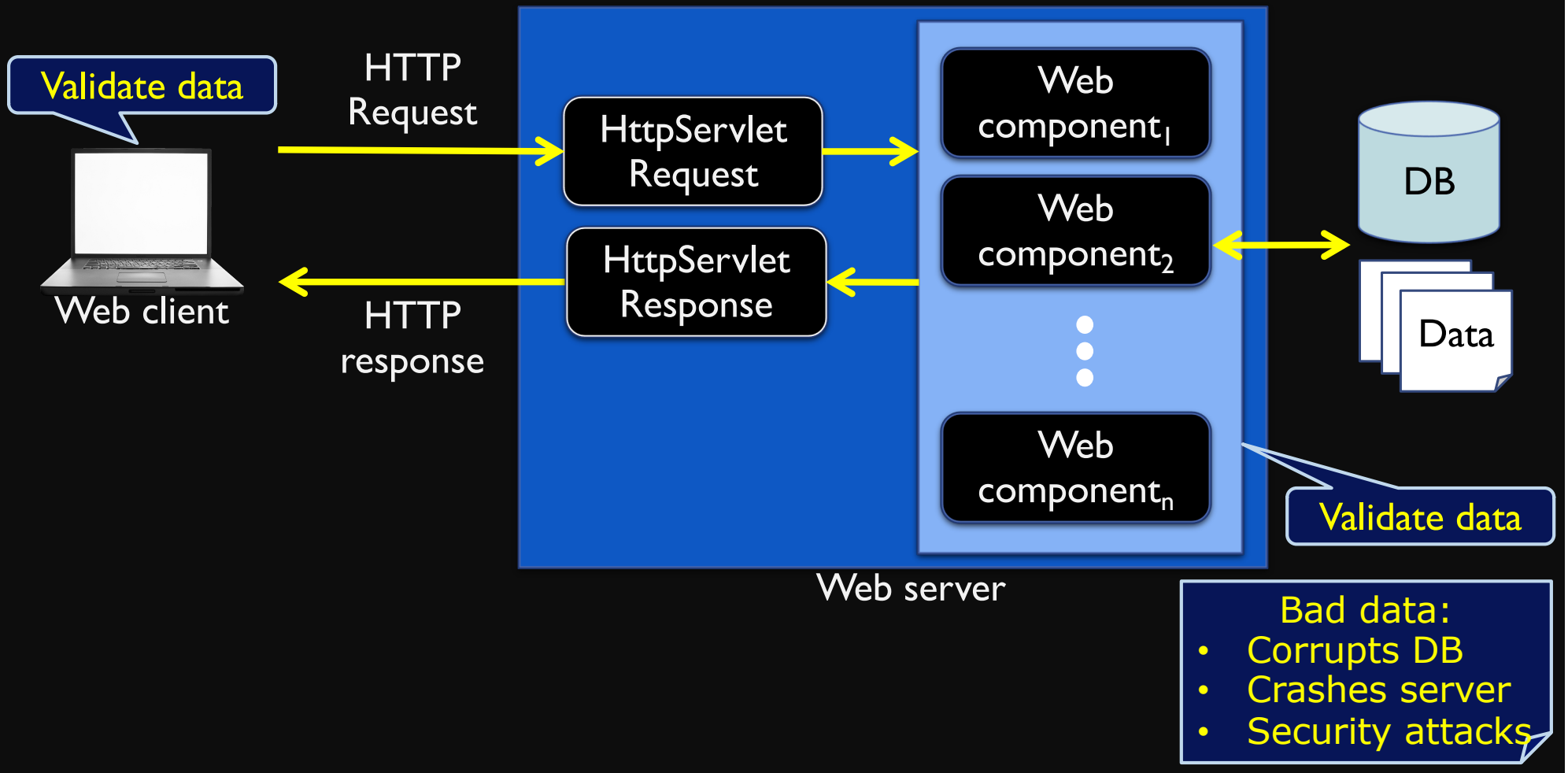
Web App Software Failures



- Apr 2016: Clicking the browser back button while using the TurboTax web software caused a user to lose information
- Feb 2015: Anthem suffered data breach
- Jan 2014: Dropbox outage caused customers to lose information
- Aug 2013: Amazon hosting service failure caused business to lose information and revenue
- Sep 2012: Cyberattack on Bank of America, JPMorgan Chase, Citigroup, US Bank, Wells Fargo, and PNC resulted in denial of services
- Jun 2012: Clicking the browser back button after successfully logging off Amazon website exposed payment information of the recent order
- Mar 2012: BodyShop website turned a buy-one-get-one 50% off to a half price each item
- Feb 2011: Over 120,000 users were unable to access email accounts due to Google mail service outage
- Sep 2011: Failures at Target delayed and canceled many customers' orders

[http://www.cs.virginia.edu/~up3f/cs3250/slides/UP_defense.pdf]

Web Applications



Development and deployment methods – extremely loosely coupled, browser-based clients, HTTP stateless, vulnerable to input manipulation

Bypass testing

- Users can easily "*bypass*" client-side constraint enforcement
- Bypass testing constructs tests to intentionally violate constraints:
  - Eases test automation
  - Validates input validation
  - Checks robustness
  - Evaluates security

Users can save and modify the HTML, and then run the modified HTML (to interact with the server)

[Offutt, Wu, Du, and Huang, "Bypass Testing of Web Applications"]

Applying Bypass Testing

- Analyze HTML to extract form elements
- Model constraints imposed by HTML and JavaScript
- Rules for test input generation:
  - From client-side constraints
  - Typical security violations
  - Common input mistakes

Proper behavior vs. Bypass behavior

Analyze HTML

to extract form elements

Activity: Identify Elements

Open the following web app via a web browser

https://www.cs.virginia.edu/~up3f/cs3250/inclass/bypass-testing/maintenance-request.php

1. Consider source code (open page source)
2. Identify elements that can be "bypassed"

Model constraints imposed by HTML and JavaScript

Types of Client-side Validation

- Performed by HTML form controls, their attributes, and client-side scripts that access DOM
- Two categories:

HTML validation	Uses elements in HTML to define input fields (controls) to limit user inputs
Scripting validation	Uses scripts to validate form data

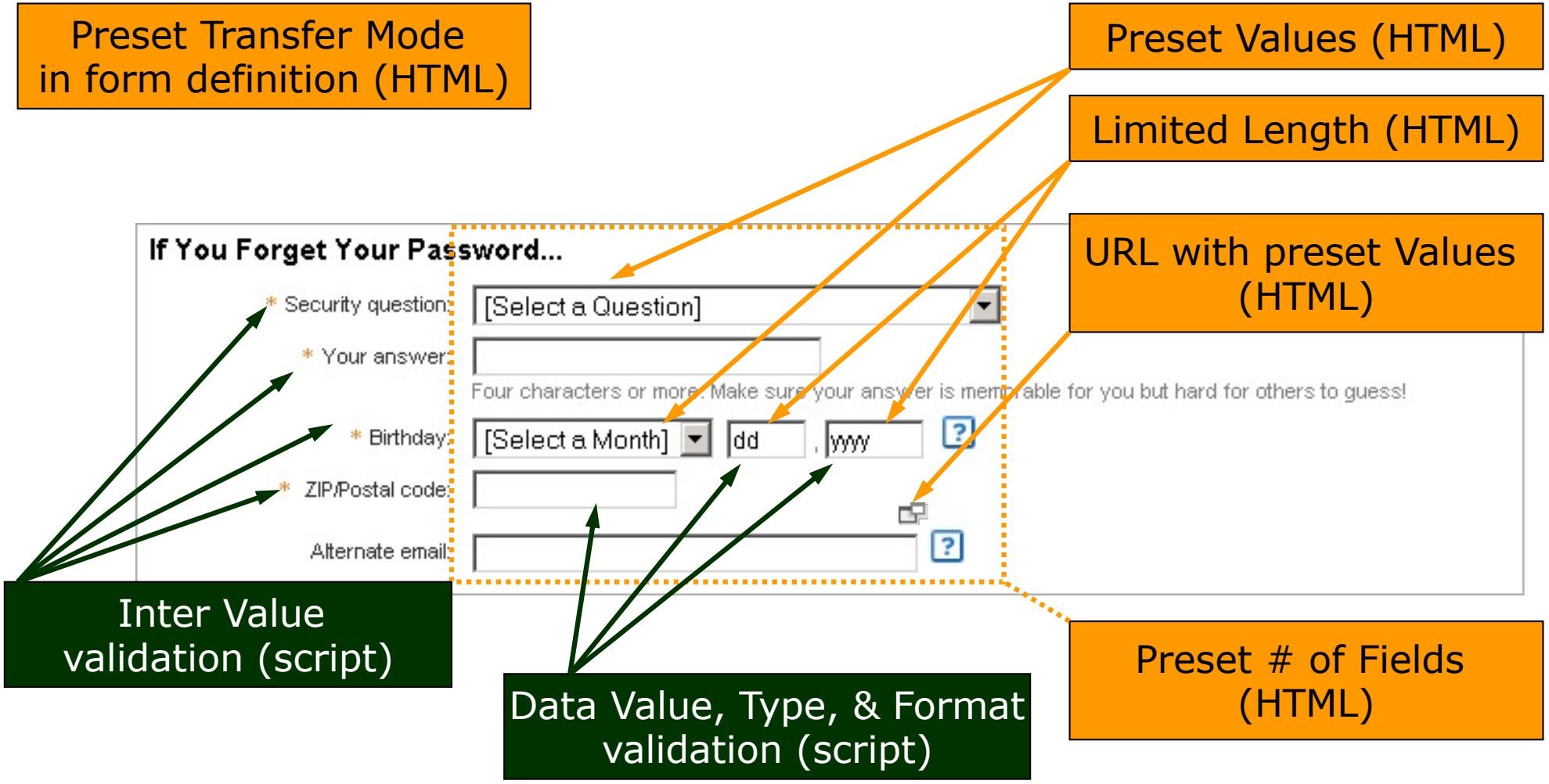
HTML Validation

Length constraints	Max number of characters allowed in a text field (*maxlength*)
Value constraints	Sets of specific values the user can submit for an input selection (*hidden inputs, menu controls, boolean controls, read-only inputs*)
Transfer mode constraints	How data are transmitted from the browser to the web server *(GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS, CONNECT, PATCH)*
Field element constraints	Data fields that are submitted to the app (*rendered and non-rendered)*
Target URL constraints	URLs the users are allowed to navigate from the current screen

Scripting Validation

Data type constraints	Types of data to be transmitted to the server (*integer check*)
Data format constraints	Format of data to be transmitted to the server *(phone format)*
Data value constraints	Data values when fields have semantic restriction (*age range*)
Inter-value constraints	Fields that must be submitted together *(credit # + expire date)*
Invalid characters constraints	Invalid strings that can cause reliability or security threats to the server *(<, >, ../, &)*

Example Interface: Yahoo Registration Form



[https://cs.gmu.edu/~offutt/documents/theses/vpapadim-thesispresent.ppt]

Activity: Model Constraints

Open the following web app via a web browser

https://www.cs.virginia.edu/~up3f/cs3250/inclass/bypass-testing/maintenance-request.php

1. Consider source code (open page source)
2. Identify client-side validation

Rules for test input generation

(some) Client-side Constraint Rules

- Violate size restrictions on strings
- Introduce values not included in static choices (e.g., radio buttons, checkboxes, select (drop-down) lists)
- Violate hard-coded values
- Use values that are considered as errors (domain-specific)
- Change HTTP transfer mode (GET, POST, …)
- Change destination URLs

(some) **Server-side Constraint Rules**

- Data type conversion
- Data format validation
- Inter-field constraint validation
- Inter-request data field (cookies, sessions, hidden inputs)

(some) **Security Violation Rules**

Potential illegal character	Symbol
Empty string	
Commas	,
Single and double quotes	‘ or “
Tag symbols	< and >
Directory paths	../
Strings starting with a forward slash	/
Strings starting with a period	.
Ampersands	&
Control character	Newline
Script symbols	<javascript> or <vbscript>

[Offutt, Bypass Testing, November 2020]

Activity: Construct Bypass Tests

1. Construct bypass tests for https://www.cs.virginia.edu/~up3f/cs3250/inclass/bypass-testing/maintenance-request.php
2. For each test you designed, encode your test using URL rewriting

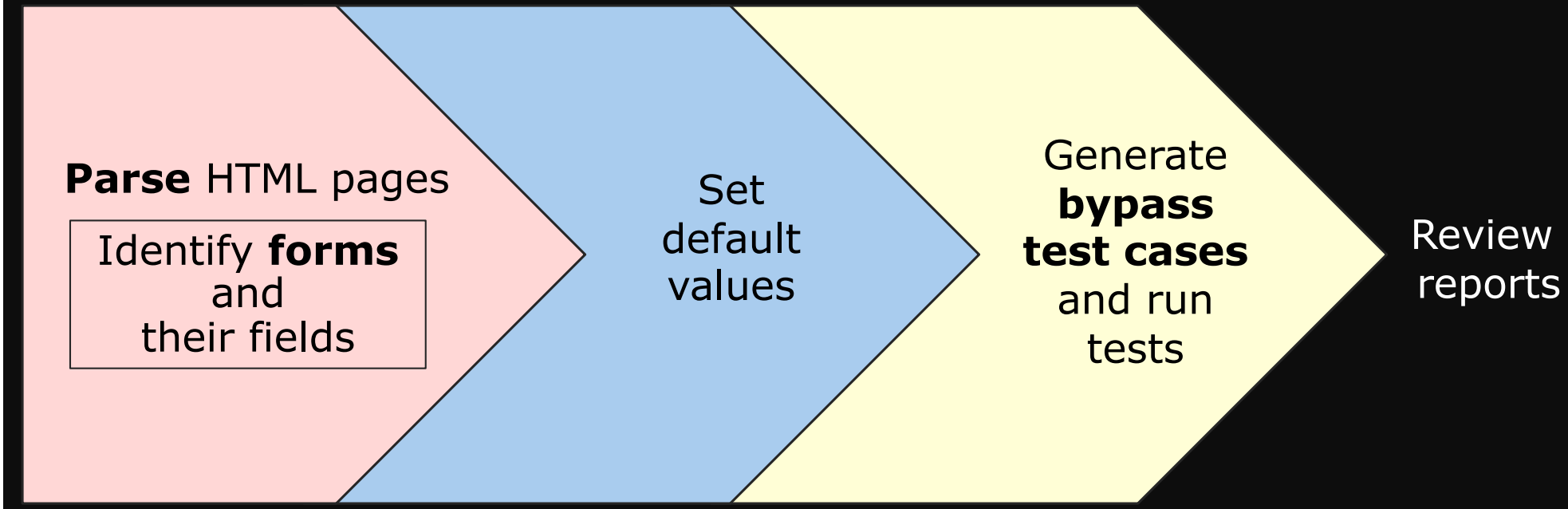
   For example, to create a valid test with 2 courses

```
requestedDate = 12/01/2025          (use %2F for /)
roomNo = Rice1800
requestedBy = someone,
requestDesc = fix light             (use + for a space)
priority_option = high
```

```
https://www.cs.virginia.edu/~up3f/cs3250/inclass/bypass-
testing/maintenance-
request.php?requestedDate=12%2F01%2F2025&roomNo=Rice1800&req
uestedBy=someone&requestDesc=fix+light&priority_option=high&
submitBtn=Submit
```

Research: AutoBypass Testing

- Autobypass: a web app that accepts a URL and generates input data for the HTML form fields
- Built on top of HttpUnit (extends Junit)

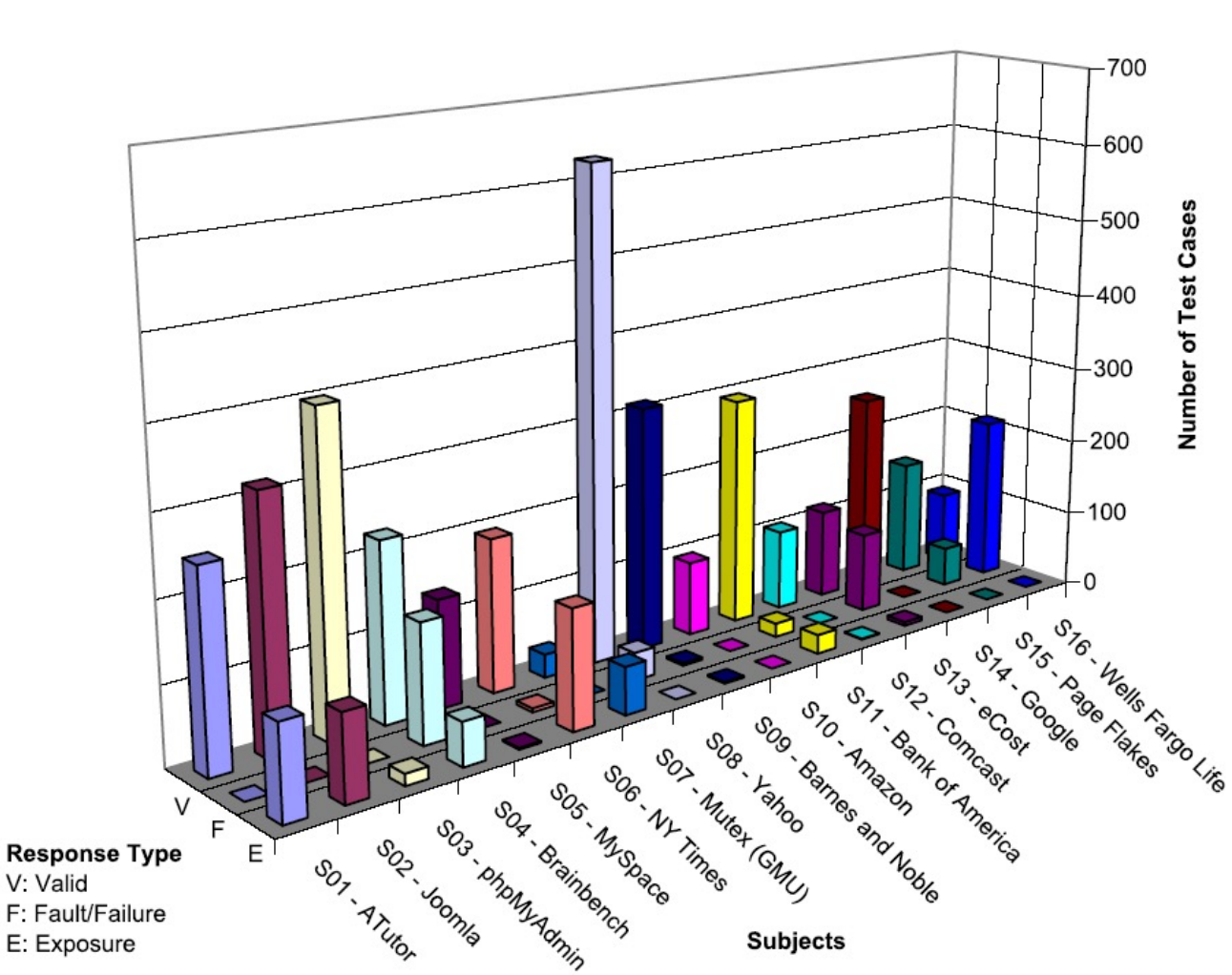


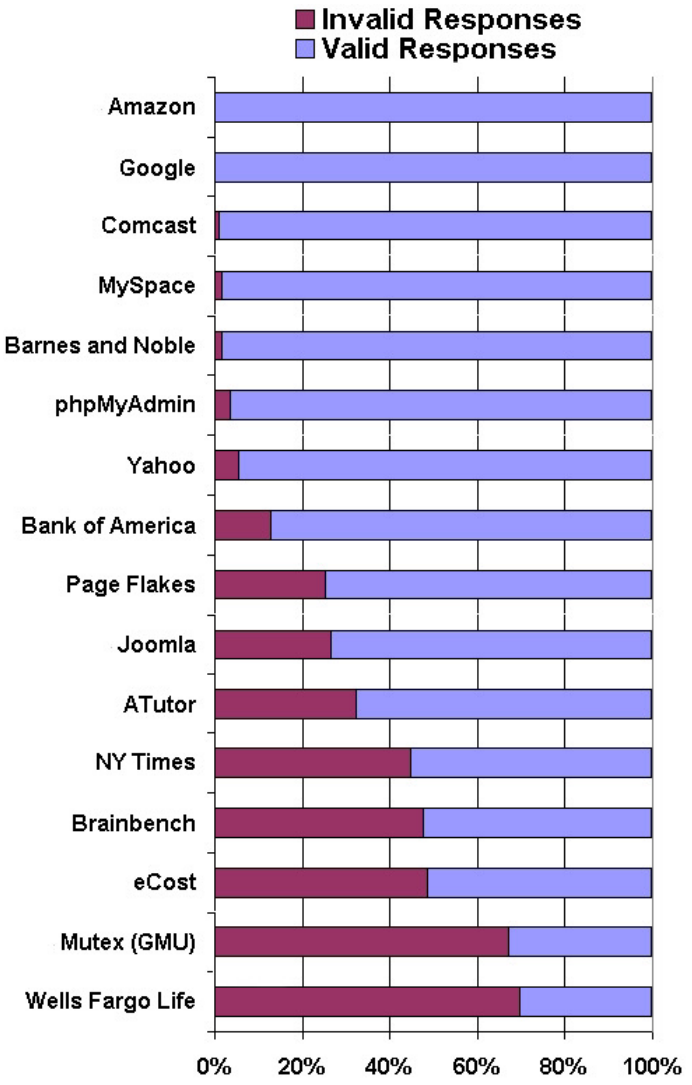
[https://cs.gmu.edu/~offutt/documents/theses/vpapadim-thesispresent.ppt]

Research: AutoBypass Testing

- Actual output ≠ expected output → failure
- Failure behavior
  - Exception reports that should not be shown to the user
  - Storing invalid data on the server
  - Security access violations
- Valid responses – 4 types:
  - V1: The app acknowledges the invalid request and provides an explicit message regarding the violation
  - V2: The server produces a generic error message
  - V3: The app apparently ignores the invalid request and produces an appropriate response
  - V4: The app apparently ignores the request

Result: AutoBypass Testing





[Offutt, Papadimitrious, Praphamontripong, "A Case Study on Bypass Testing of Web Applications"]

Wrap-Up

- Bypass testing can reveal errors in web apps
- Inexpensive to test web apps (resources and human labor)
- Efficient method creating limited test cases
- The ideas have been applied to the server-side validation

Don’t trust users!!!

Apply input validation to all inputs