Com S 417
Software Testing

Fall 2017 – Week 9, Lecture 16

Announcements

- Research Project.
- Exam 2 will be Nov. 2 in class.
- Lab 4 is available and due Oct. 31.
- We *will* have 5 labs.

Topics

- Issuing a request from a junit test.
- Template languages
- Hello World in JSP
- In-container vs. Out-of-Container tests.
- Spring and alternative deployments.
- Combinatorial Testing
- Exam Schedule.

JSP (Java Server Pages)

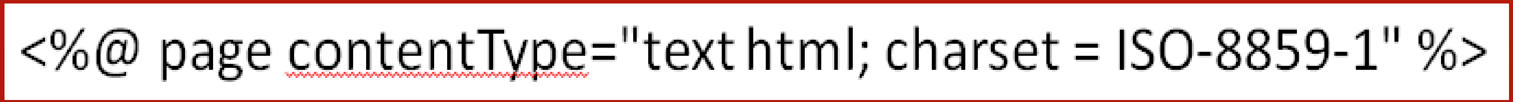
- The following slides (with white background) were prepared by S. Mitra.

```
<%@ page contentType="text html; charset = ISO-8859-1" %>


<HTML>  <HEAD>   <META HTTP-EQUIV="Content-Type"
CONTENT="text/html; charset=ISO-8859-1">
<TITLE>CS417 Hello JSP World</TITLE>
</HEAD>
<p><font color=red>Here we print what we got from original form +
from servlet1   + from servlet2</font></p>
<%  out.println(request.getParameter("original"));
     out.println(request.getAttribute("sv1_message"));
     out.println(request.getAttribute("sv2_message"));
%>
</HTML>
```

Best Practices

- Know what character set your editor uses so that you can code the charset in correctly. It should describe the charset used to create/edit the jsp file.



```
<%@ page contentType="text html; charset = ISO-8859-1" %>
```

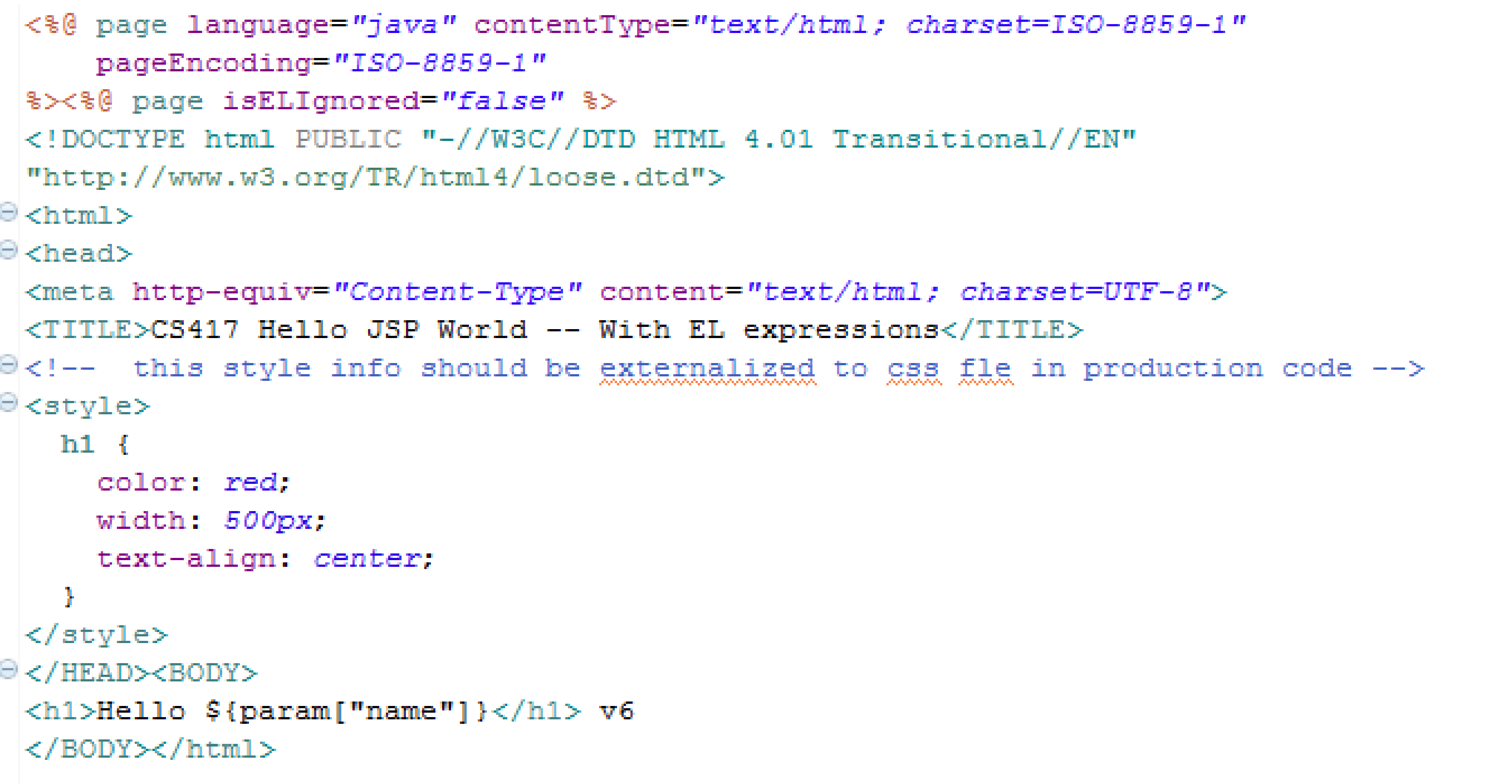
- Include a valid strict-mode doc-type header
- <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

  See https://www.quirksmode.org/css/quirksmode.html and https://www.w3.org/QA/2002/04/valid-dtd-list.html
- Use UTF-8 in the generated HTML: <meta charset="UTF-8">

JSP – further exploration

- JSP details
  - https://www.tutorialspoint.com/jsp/index.htm
- JSTL (standard tag library)
  - https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm
- EL (expression language)
  more convenient access to certain pre-defined container objects. See Implicit Objects in
  - https://www.tutorialspoint.com/jsp/jsp_expression_language.htm

Hello World with EL



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"
%><%@ page isELIgnored="false" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<TITLE>CS417 Hello JSP World -- With EL expressions</TITLE>
<!--  this style info should be externalized to css fle in production code -->
<style>
  h1 {
    color: red;
    width: 500px;
    text-align: center;
  }
</style>
</HEAD><BODY>
<h1>Hello ${param["name"]}</h1> v6
</BODY></html>
```

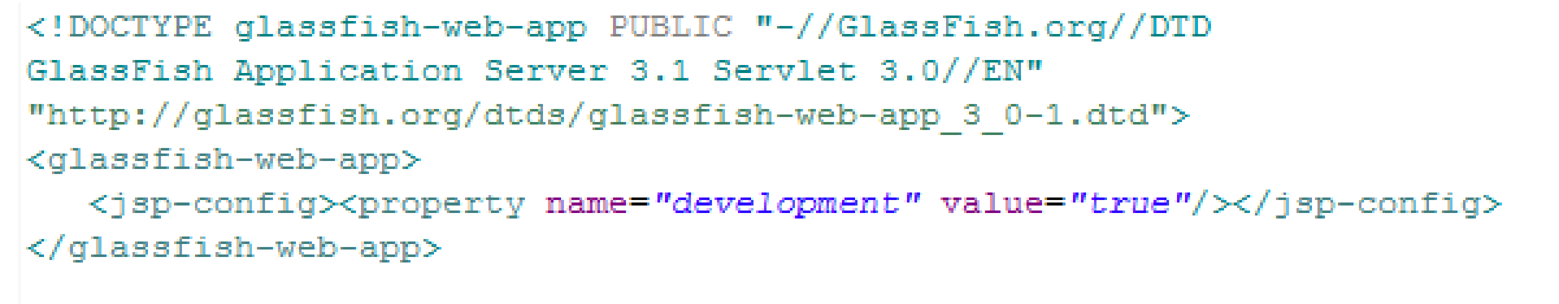
Enabling EL in glassfish (per page)

- Add the page directive:

```
<%@ page isELIgnored="false" %>
```

Enabling Drop in JSP in Glassfish

- Enable development mode in glassfish-web.xml. (It goes in same directory with web.xml)



```
<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD
GlassFish Application Server 3.1 Servlet 3.0//EN"
"http://glassfish.org/dtds/glassfish-web-app_3_0-1.dtd">
<glassfish-web-app>
   <jsp-config><property name="development" value="true"/></jsp-config>
</glassfish-web-app>
```

- Restart the container.
- Now if you copy-paste the edited .jsp file to

  glassfish/domains/domain1/applications/<contextroot>/

  the container will detect the modified file and recompile the jsp next time it is requested.

Design Considerations

- *Very* handy to have unique identifier on each HTTP page so that you can do a quick test such as

  content.asString().contains(“<body id=‘main’”)

  to determine if you’ve successfully navigated to a particular page.
- It is important that you separate generation of the presentation (html, etc), from maintenance of the server-side model. For example:
  - Receive and process the post against the database,
  - Then forward to a filter servlet to generate the HTML based on the new database contents.

In Container vs. Out of Container

- ‘In Container’ means that the SUT is executing in a container.
  - Test code generally runs in a different execution environment, complicating deployment, coordination, instrumentation, result collection, etc.
- ‘Out of Container’ means that the SUT and test code can be running in the same execution environment (even the same JVM).
  - When the core business logic is properly separated from network and presentation concerns, and assembled in a separate “construction” phase, then it doesn’t need to be in the container to be tested.
  - Spring and Spring Integration-like gateways and endpoints simplify this approach.

Front Door Controller

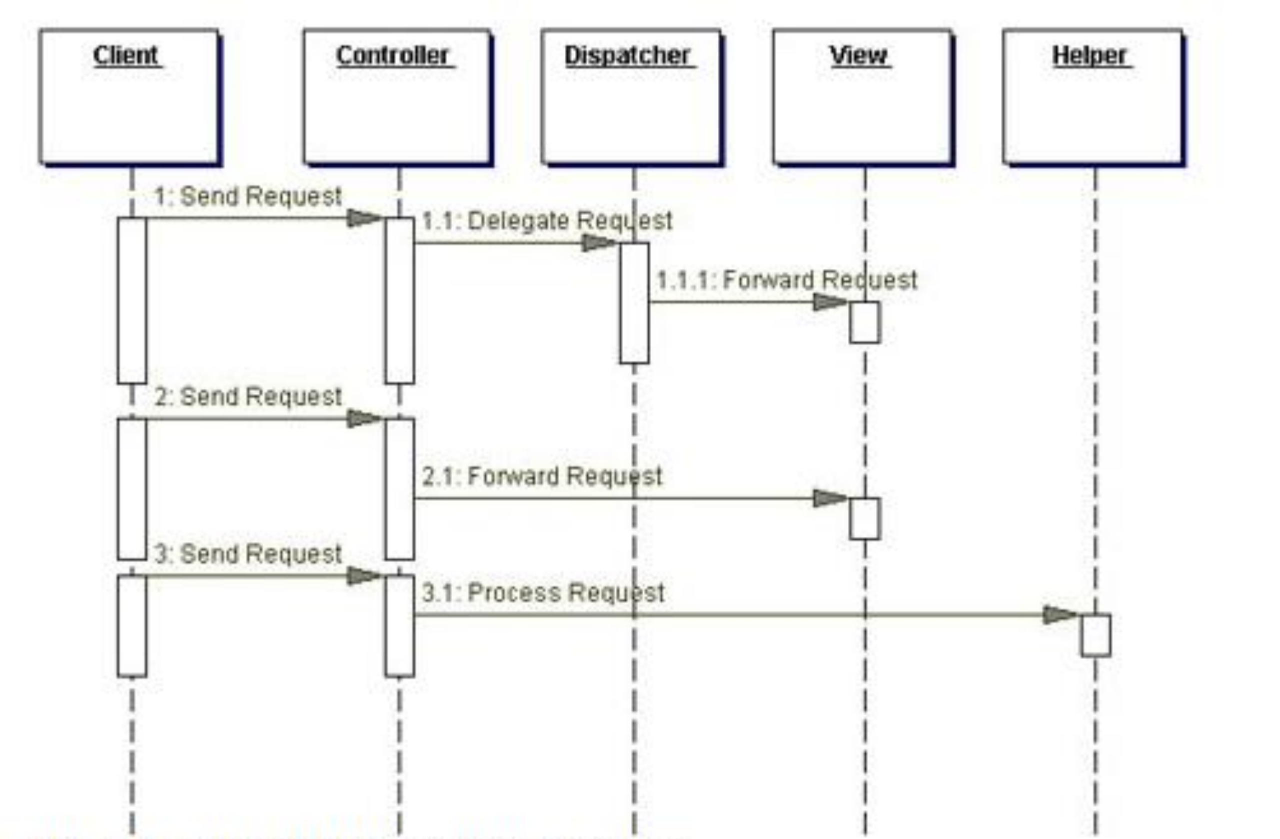
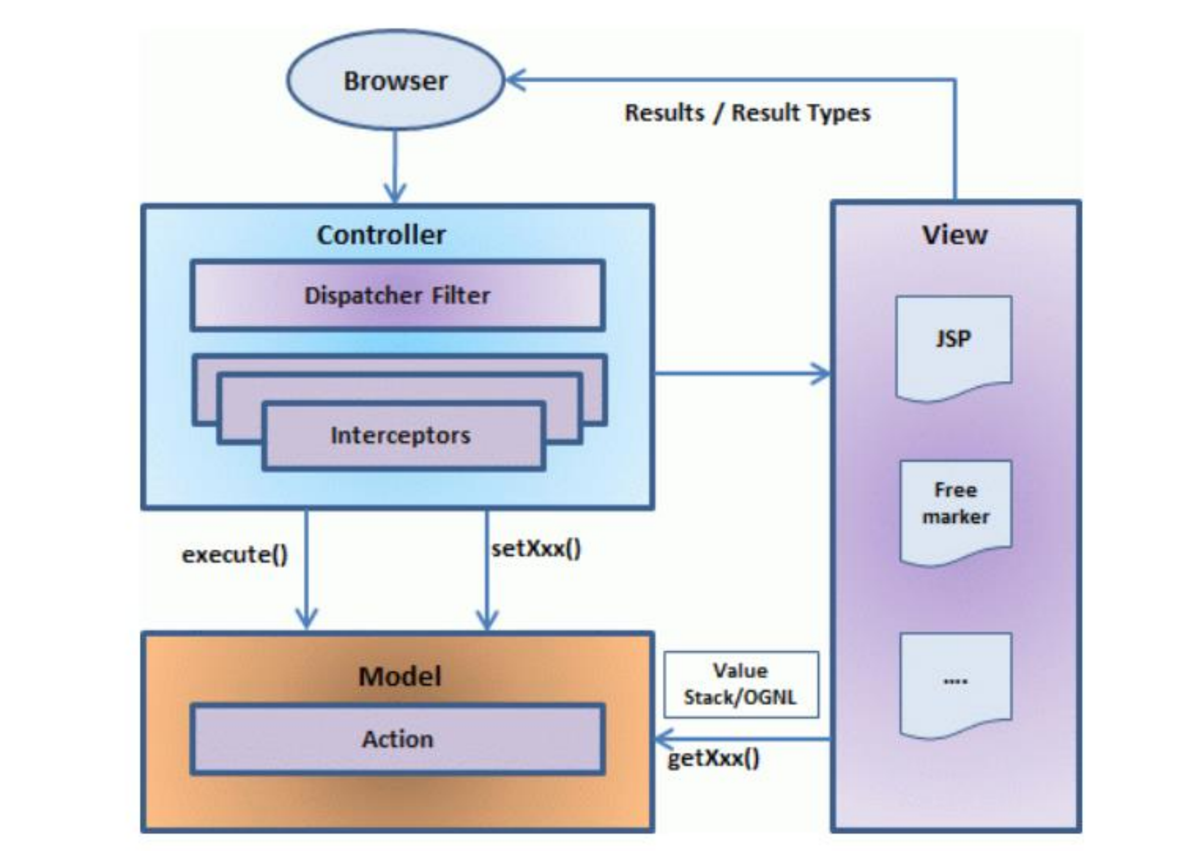


Figure 7.8 Front Controller sequence diagram

from: http://www.oracle.com/technetwork/java/frontcontroller-135648.html

Struts Architecture



from: https://www.tutorialspoint.com/struts_2/struts_architecture.htm

Servlet Engines for our Labs
Tomcat

- Tomcat is the Servlet Engine than handles servlet requests for Apache (a generic network server)
  - Tomcat is a "helper application" for Apache
- Apache can handle many types of web services
  - Apache can be installed without Tomcat
  - Tomcat can be installed without Apache
    - It is easier to install Tomcat standalone than as part of Apache
    - By itself, Tomcat can handle web pages, servlets, and JSP
- Apache and Tomcat are open source (and therefore free)

Adapted from slides by S Mitra

Servlet Engines for our Labs

GlassFish

- GlassFish is Sun/Oracle’s reference implementation for the Java EE (enterprise) 6 specification.
  - GlassFish is open source.
  - Because GlassFish was created “from scratch” to support the extensions in Java EE, the relationship of GlassFish artifacts to the specification is a little more natural.
- By default, Eclipse does *not* support integrated control of GlassFish.
  - A good thing, because Eclipse prefer’s to use the embedded version of Tomcat, which can cause significant confusion for beginners.

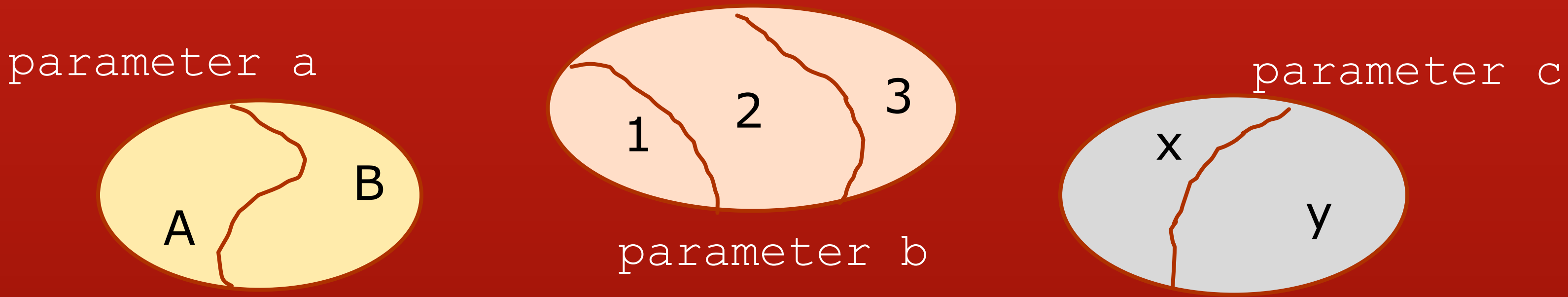
Adapted from slides by S Mitra

Combinatorial Testing & Interaction Faults

- Pairwise combinations are a form of combinatorial testing.
- Earlier we focused on pairwise combinations of representative values from all equivalence classes in two or three inputs.
- Pairwise is popular because it is relatively effective at reaching interaction faults involving two conditions (2-way interactions).

Each Choice Example

For “Each choice” a value chosen from each block must appear in at least one test:



parameter a

A

B

1

2

3

parameter b

parameter c

x

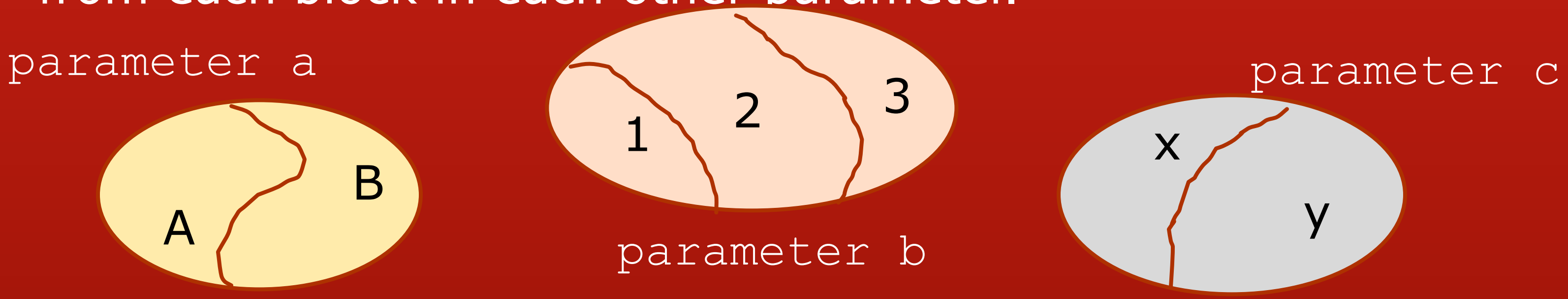
y

Each Choice
(A, 1, x) (B, 2, y)
(A, 3, x)

Many fewer tests, but many more opportunities for a bug to go undiscovered.

Pair-Wise Example (worked out)

For “Pair wise” a value for each block in each parameter partition (characteristic) must be combined with a value from each block in each other parameter.



Pairs to Include

(A,1)	(B,1)	(1,x)
(A,2)	(B,2)	(1,y)
(A,3)	(B,3)	(2,x)
(A,x)	(B,x)	(2,y)
(A,y)	(B,y)	(3,x)
		(3,y)

Pairwise Tests

(A, 1, x) ~~(B, 1, x)~~
~~(A, 1, y)~~ (B, 1, y)
(A, 2, x) ~~(B, 2, x)~~
~~(A, 2, y)~~ (B, 2, y)
(A, 3, x) ~~(B, 3, x)~~
(A, ~~3~~, y) (B, ~~3~~, y)

Many fewer tests, but many more opportunities for a bug to go undiscovered.

Mathur introduces pairwise design in sections 4.6 and 4.7

Presentation by Richard Kuhn NIST

https://csrc.nist.gov/CSRC/media/Presentations/Introduction-to-Combinatorial-Testing-Presentation/images-media/kuhn-intro-mse-nist.pdf

Software Failure Internals

- How does an interaction fault manifest itself in code?

Example: pressure < 10 & volume > 300 (2-way interaction)

```
if (pressure < 10) {
    // do something
    if (volume > 300)  { faulty code!  BOOM! }
    else { good code, no problem}
} else {
    // do something else
}
```

A test that included pressure = 5 and volume = 400 would trigger this failure

Pairwise testing is popular, but is it enough?

- Pairwise testing commonly applied to software
- Intuition: some problems only occur as the result of an interaction between parameters/components
- Tests all pairs (2-way combinations) of variable values
- Pairwise testing finds about 50% to 90% of flaws

Finding 90% of flaws is pretty good, right?

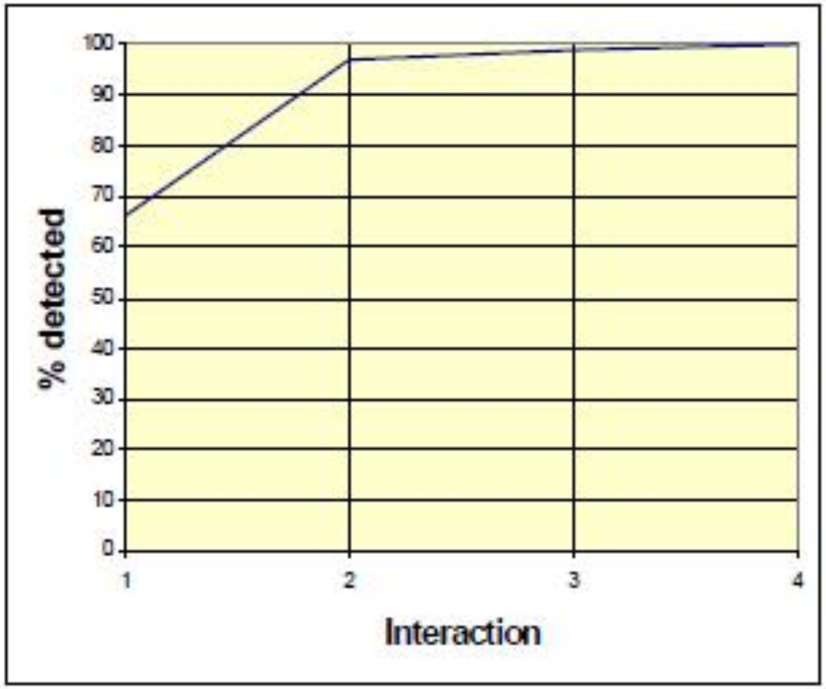
"Relax, our engineers found 90 percent of the flaws."

NIST
National Institute of Standards and Technology

How about hard-to-find flaws?

- Interactions e.g., failure occurs if
- pressure < 10 (1-way interaction)
- pressure < 10 & volume > 300 (2-way interaction)
- pressure < 10 & volume > 300 & velocity = 5 (3-way interaction)
- The most complex failure reported required 4-way interaction to trigger

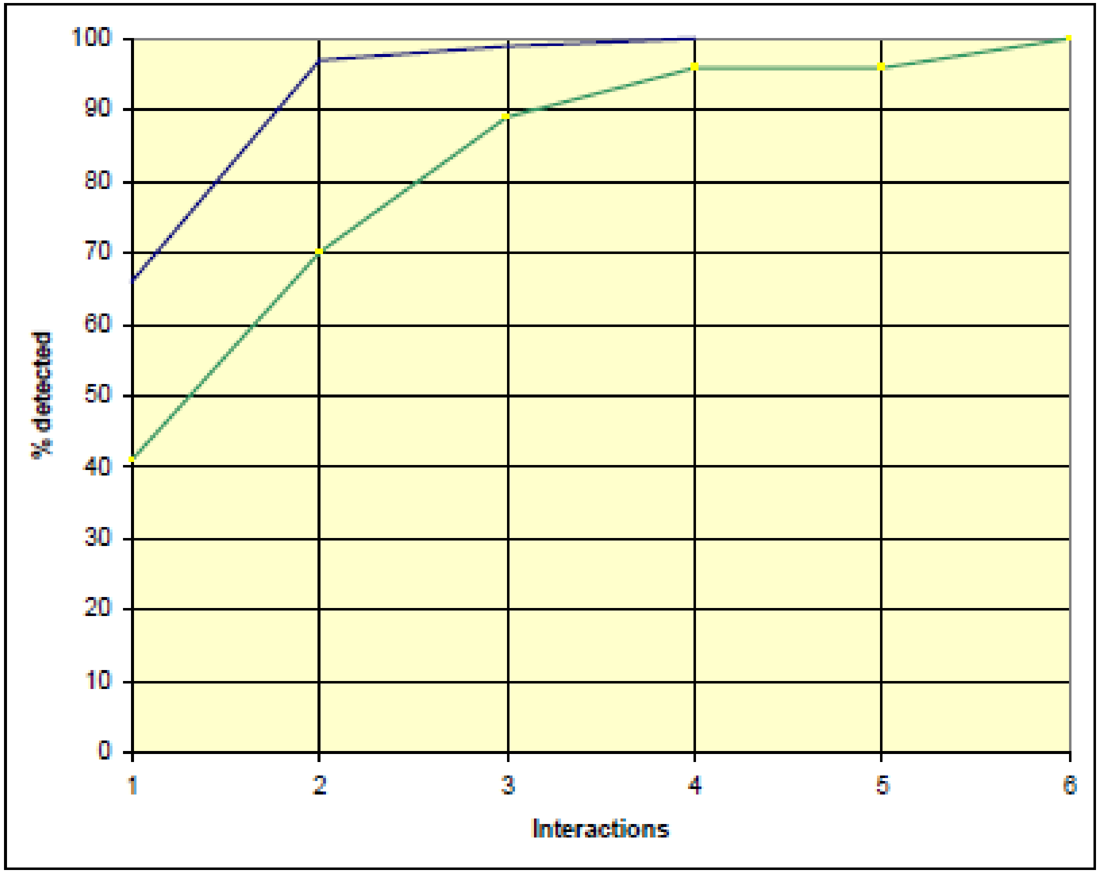
NIST study of 15 years of FDA medical device recall data



Interesting, but that's just one kind of application.

How about other applications?

Browser (green)



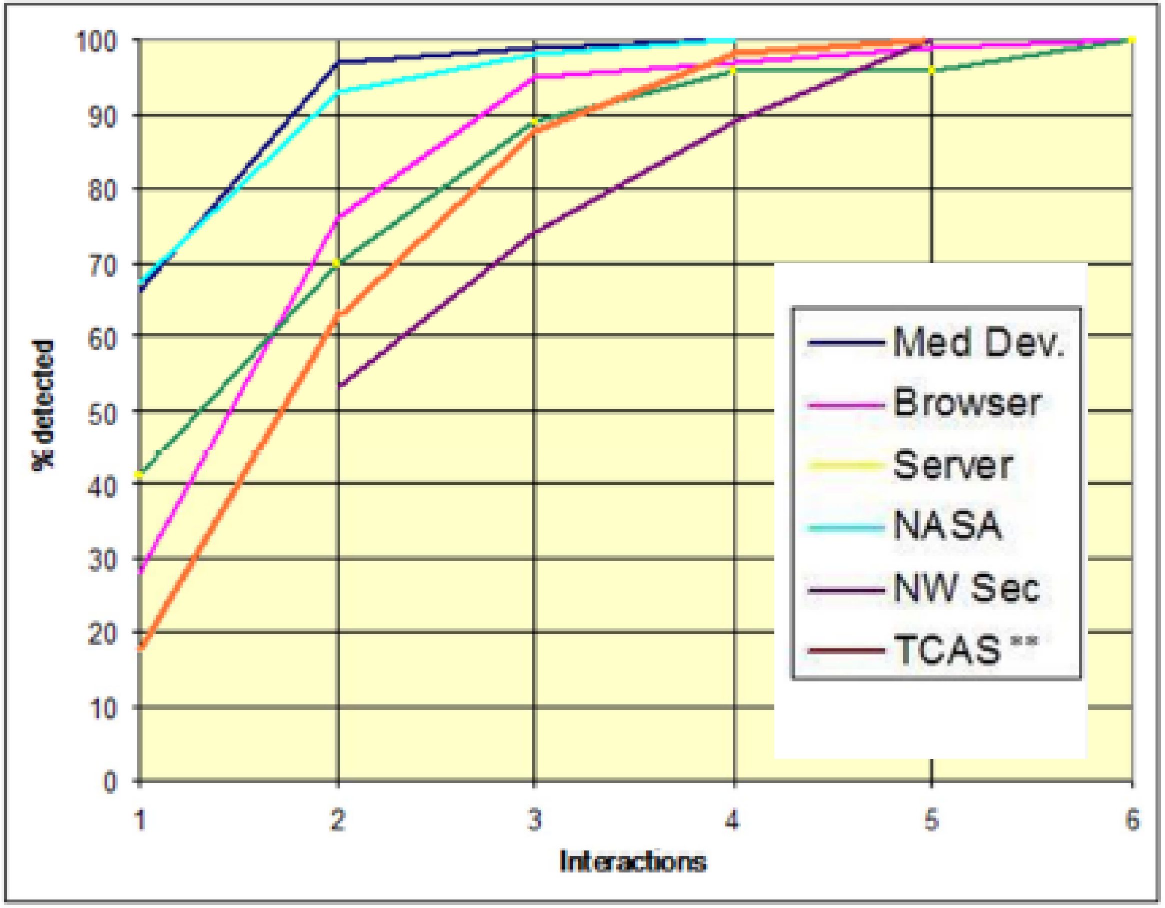
These faults more complex than medical device software!!

Why?

Finally

Network security (Bell, 2006)

(orange)



Curves appear to be similar across a variety of application domains.

Why this distribution?

So, how many parameters are involved in really tricky faults?

- Maximum interactions for fault triggering for these applications was 6
- Much more empirical work needed
- Reasonable evidence that maximum interaction strength for fault triggering is relatively small

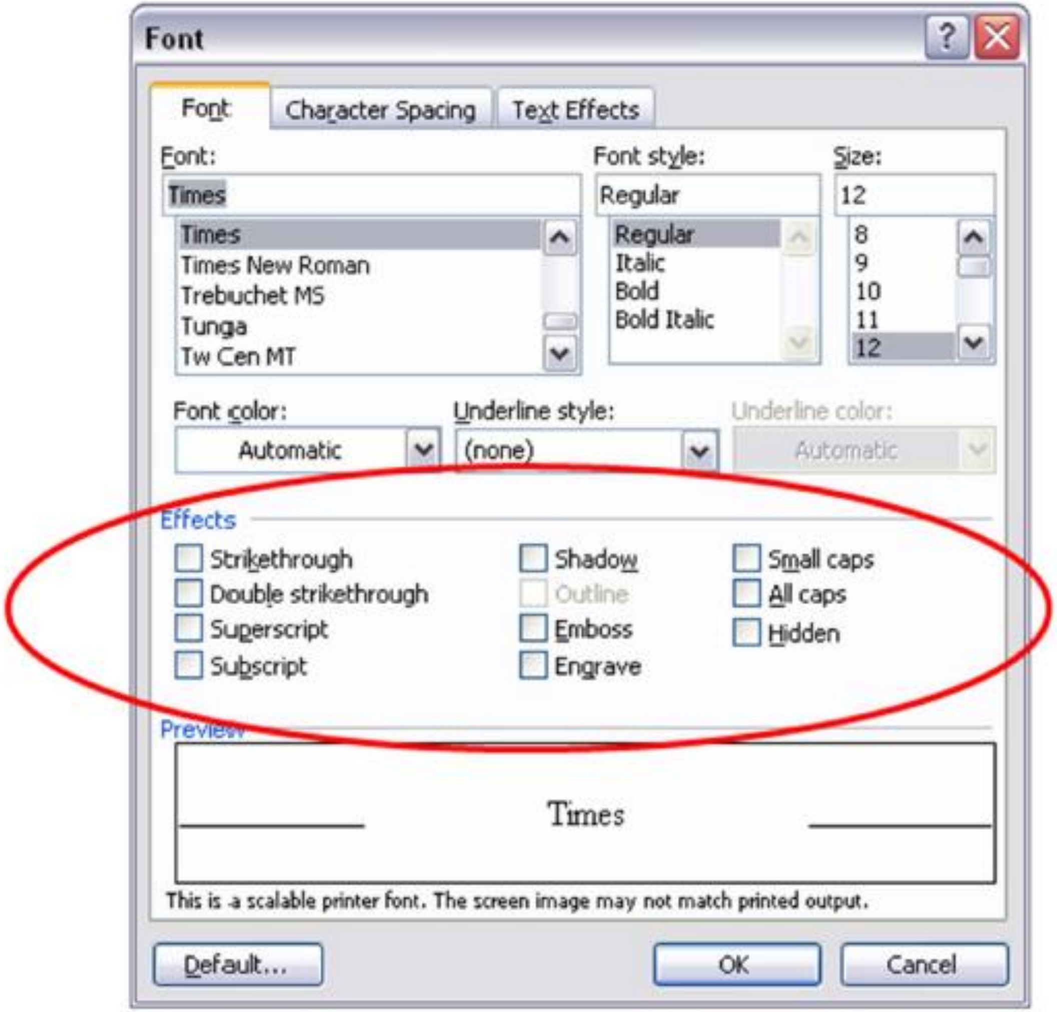
How does this knowledge help?

Biologists have a “central dogma”, and so do we:

If all faults are triggered by the interaction of *t* or fewer variables, then testing all *t*-way combinations can provide strong assurance

(taking into account: value propagation issues, equivalence partitioning, timing issues, more complex interactions, . . .)

What is combinatorial testing? A simple example



Font: Times

Font style: Regular

Size: 12

Font color: Automatic

Underline style: (none)

Underline color: Automatic

Effects

☐ Strikethrough
☐ Double strikethrough
☐ Superscript
☐ Subscript
☐ Shadow
☐ Outline
☐ Emboss
☐ Engrave
☐ Small caps
☐ All caps
☐ Hidden

Preview

Times

This is a scalable printer font. The screen image may not match printed output.

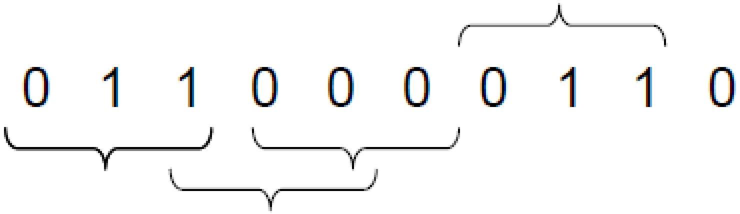
How Many Tests Would It Take?

- There are 10 effects, each can be on or off
- All combinations is 2^{10} = 1,024 tests
- What if our budget is too limited for these tests?
- Instead, let’s look at all 3-way interactions …

NIST
National Institute of
Standards and Technology

Now How Many Would It Take?

- There are $\binom{10}{3}$ = 120 3-way interactions.
- Naively 120 x 2^3 = 960 tests.
- Since we can pack 3 triples into each test, we need no more than 320 tests.
- Each test exercises many triples:



0 1 1 0 0 0 0 1 1 0

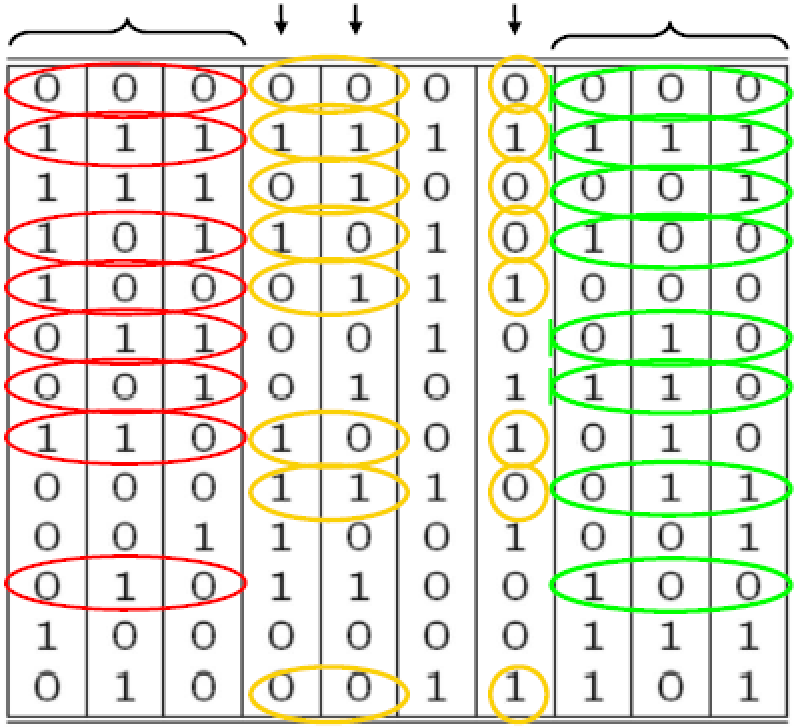
We can pack a lot into one test, so what's the smallest number of tests we need?

A covering array

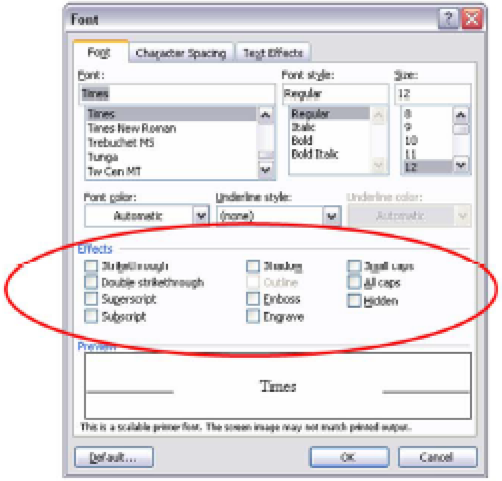
All triples in only 13 tests, covering $\binom{10}{3}2^3 = 960$ combinations

Each row is a test:

Each column is a parameter:



0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1



Font

Font | Character Spacing | Text Effects

Font: Times
Times
Times New Roman
Trebuchet MS
Tunga
Tw Cen MT

Font style: Regular
Regular
Italic
Bold
Bold Italic

Size: 12
8
9
10
11
12

Font color: Automatic

Underline style: (none)

Underline color: Automatic

Effects

□ Strikethrough
□ Double strikethrough
□ Superscript
□ Subscript
□ Shadow
□ Outline
□ Emboss
□ Engrave
□ Small caps
□ All caps
□ Hidden

Preview

Times

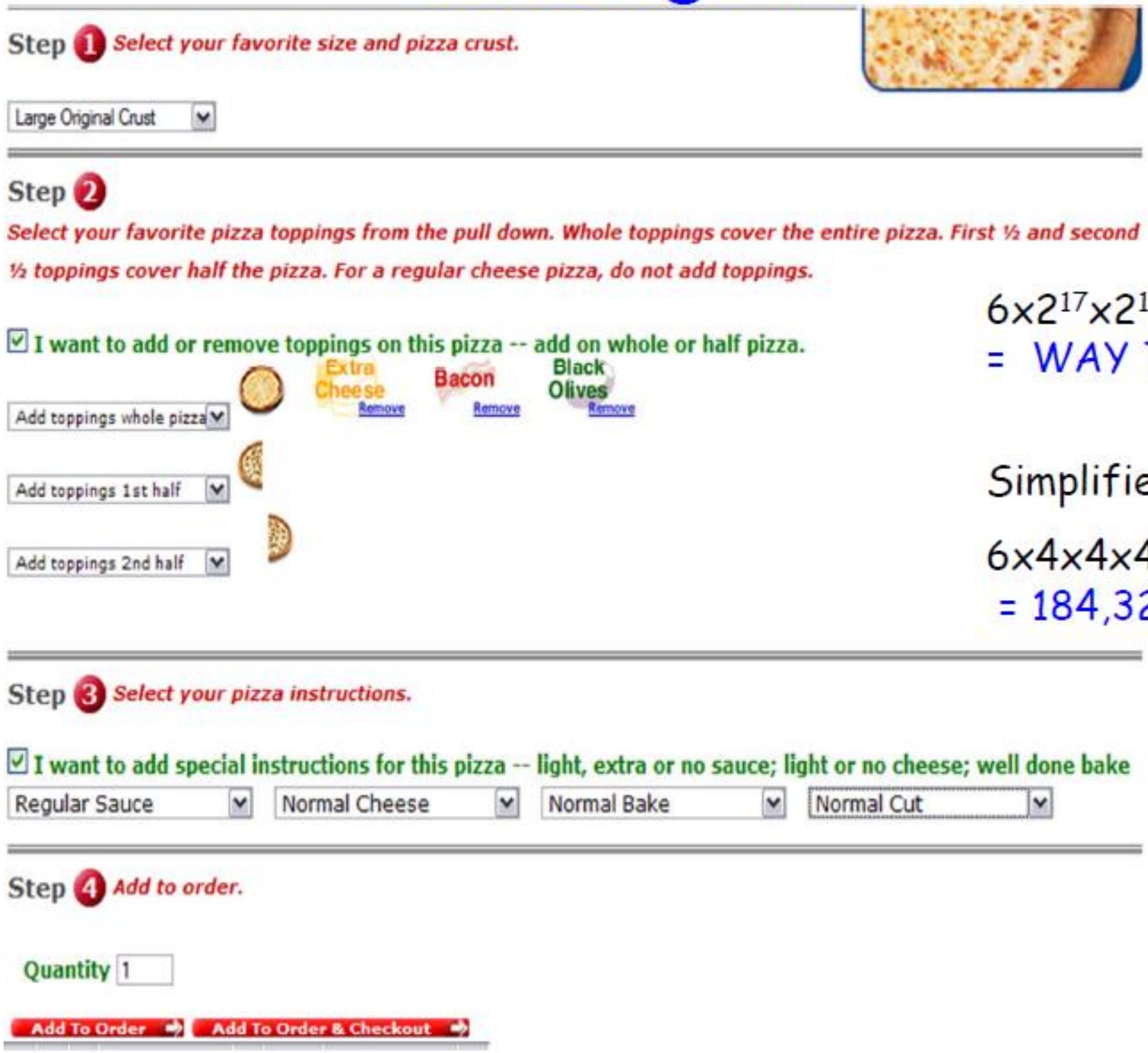
This is a scalable printer font. The screen image may not match printed output.

Each test covers $\binom{10}{3}$ = 120 3-way combinations

<u>Finding covering arrays is NP hard</u>

Ordering Pizza

National Institute of Standards and Technology



Step 1 *Select your favorite size and pizza crust.*

Large Original Crust

Step 2

Select your favorite pizza toppings from the pull down. Whole toppings cover the entire pizza. First ½ and second ½ toppings cover half the pizza. For a regular cheese pizza, do not add toppings.

☑ I want to add or remove toppings on this pizza -- add on whole or half pizza.

Add toppings whole pizza

Add toppings 1st half

Add toppings 2nd half

Step 3 *Select your pizza instructions.*

☑ I want to add special instructions for this pizza -- light, extra or no sauce; light or no cheese; well done bake

Regular Sauce | Normal Cheese | Normal Bake | Normal Cut

Step 4 *Add to order.*

Quantity 1

$6 \times 2^{17} \times 2^{17} \times 2^{17} \times 4 \times 3 \times 2 \times 2 \times 5 \times 2$

= WAY TOO MUCH TO TEST

Simplified pizza ordering:

$6 \times 4 \times 4 \times 4 \times 4 \times 3 \times 2 \times 2 \times 5 \times 2$

= 184,320 possibilities

Ordering Pizza Combinatorially

Simplified pizza ordering:

6x4x4x4x4x3x2x2x5x2
= 184,320 possibilities

2-way tests:	32
3-way tests:	150
4-way tests:	570
5-way tests:	2,413
6-way tests:	8,330

If all failures involve 5 or fewer parameters, then we can have confidence after running all 5-way tests.

How do we test this?

- 34 switches = 2^{34} = 1.7 x 10^{10} possible inputs = 1.7 x 10^{10} tests

What if we knew no failure involves more than 3 switch settings interacting?

- 34 switches = 2^{34} = 1.7 x 10^{10} possible inputs = **1.7 x 10^{10}** tests
- If only 3-way interactions, need only **33** tests
- For 4-way interactions, need only **85** tests

NIST
National Institute of Standards and Technology

New algorithms

- Smaller test sets faster, with a more advanced user interface
- First parallelized covering array algorithm
- More information per test

T-Way	IPOG		ITCH (IBM)		Jenny (Open Source)		TConfig (U. of Ottawa)		TVG (Open Source)	
	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
2	100	0.8	120	0.73	108	0.001	108	>1 hour	101	2.75
3	400	0.36	2388	1020	413	0.71	472	>12 hour	9158	3.07
4	1363	3.05	1484	5400	1536	3.54	1476	>21 hour	64696	127
5	4226	18s	NA	>1 day	4580	43.54	NA	>1 day	313056	1549
6	10941	65.03	NA	>1 day	11625	470	NA	>1 day	1070048	12600

Traffic Collision Avoidance System (TCAS): $2^7 3^2 4^1 10^2$

Times in seconds

Unlike diet plans,
results ARE typical!

Cost and Volume of Tests

- Number of tests: proportional to $v^t \log n$

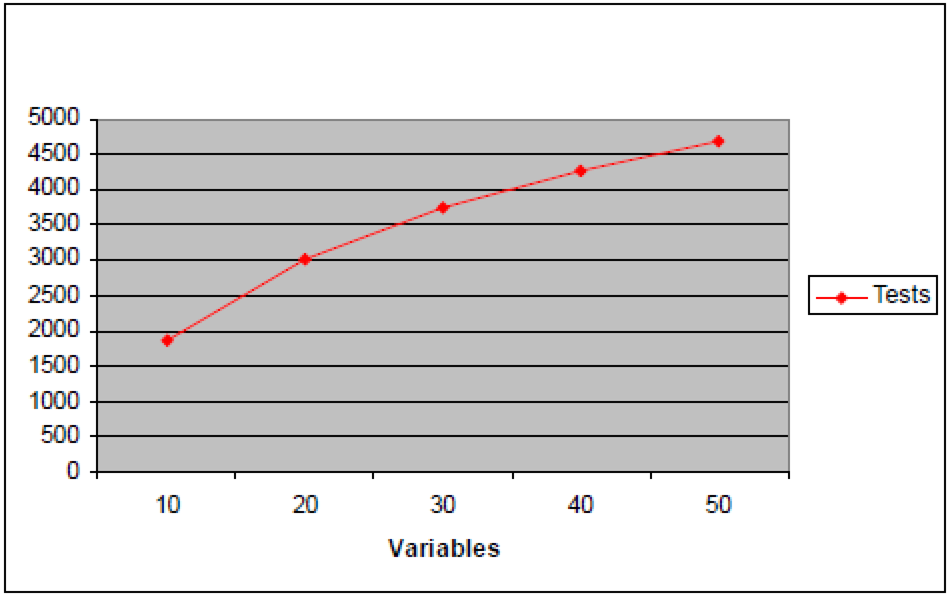
  for v values, n variables, t-way interactions

- *Thus:*

  •Tests increase exponentially with interaction strength *t* : BAD, but unavoidable

  •But only logarithmically with the number of parameters : GOOD!

- Example: suppose we want all 4-way combinations of *n* parameters, 5 values each:



National Institute of Standards and Technology

EXAMPLE 2: Document Object Model Events

- DOM is a World Wide Web Consortium standard incorporated into web browsers

- NIST Systems and Software division develops tests for standards such as DOM

- DOM testing problem:
  - large number of events handled by separate functions
  - functions have 3 to 15 parameters
  - parameters have many, often continuous, values
  - verification requires human interaction (viewing screen)
  - testing takes a *long* time

NIST
National Institute of
Standards and Technology

DOM Functions

Event Name	Param.	Tests
Abort	3	12
Blur	5	24
Click	15	4352
Change	3	12
dblClick	15	4352
DOMActivate	5	24
DOMAttrModified	8	16
DOMCharacterDataModified	8	64
DOMElementNameChanged	6	8
DOMFocusIn	5	24
DOMFocusOut	5	24
DOMNodeInserted	8	128
DOMNodeInsertedIntoDocument	8	128
DOMNodeRemoved	8	128
DOMNodeRemovedFromDocument	8	128
DOMSubTreeModified	8	64
Error	3	12
Focus	5	24
KeyDown	1	17
KeyUp	1	17
Load	3	24
MouseDown	15	4352
MouseMove	15	4352
MouseOut	15	4352
MouseOver	15	4352
MouseUp	15	4352
MouseWheel	14	1024
Reset	3	12
Resize	5	48
Scroll	5	48
Select	3	12
Submit	3	12
TextInput	5	8
Unload	3	24
Wheel	15	4096
Total Tests		36626

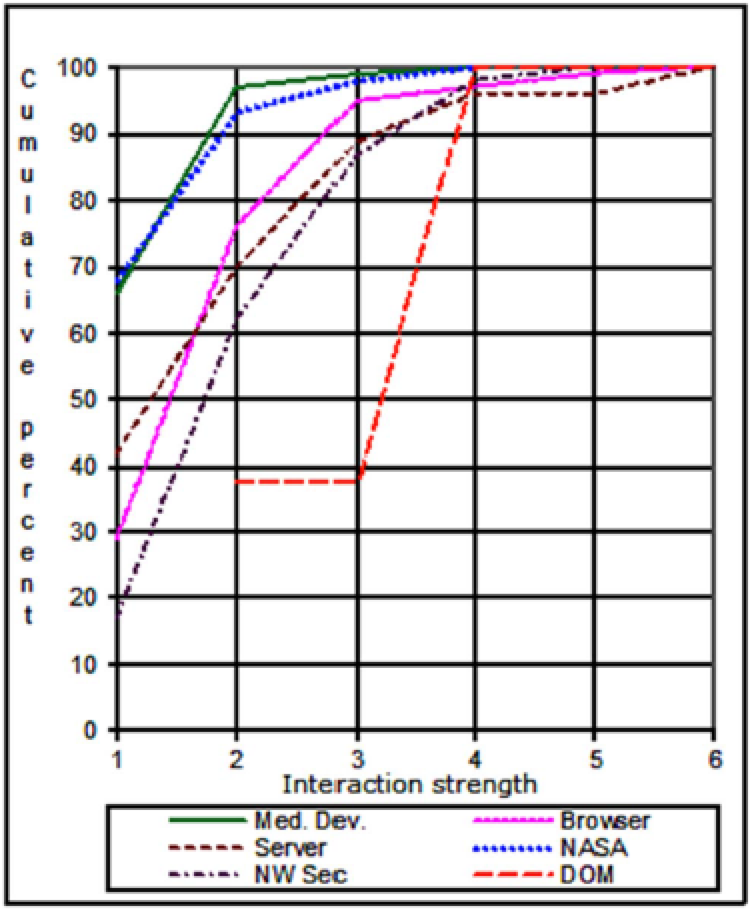
Exhaustive testing of equivalence class values

NIST
National Institute of Standards and Technology

World Wide Web Consortium
Document Object Model Events

t	Tests	% of Orig.	Test Results		
			Pass	Fail	Not Run
2	702	1.92%	202	27	473
3	1342	3.67%	786	27	529
4	1818	4.96%	437	72	1309
5	2742	7.49%	908	72	1762
6	4227	11.54 %	1803	72	2352

All failures found using < 5% of original exhaustive discretized test set



SUMMARY

- Combinatorial testing is now a practical approach that produces high quality testing at lower cost

- Good algorithms and user-friendly tools are available – no cost tools from NIST, Microsoft, others

- Basic combinatorial testing can be used in two ways:
  - combinations of configuration values
  - combinations of input values
  - these can be used separately or at the same time

- Case studies are beginning to appear

- All tools and materials available at NIST web site csrc.nist.gov/acts

Research Project

- Teams of 4 to 5 (slots for 16 to 20 teams).
- Written report.
- 15 minute Presentation (all team members have speaking role).
- Attendance is required for everyone during team presentations: Nov 16, 28, 30, Dec 5, and Dec 7.
- Teams and topics must be submitted ASAP. Your proposal must be approved Oct 31. Submit to Rumesh.
- First team to propose a topic gets it, so act quickly.
- Emphasis is on new or advanced tools and on recent research.

Suggested topics

- New and or widely used tools for
  - Generating combinatorial testing covering arrays.
  - Mutation testing.
  - Performance testing (application scope, e.g. TPTP)
  - Performance testing (web scope)
  - Test and Defect management (especially test to requirements traceability).
  - Test Instrumentation (especially non-interferring)
    - mediation systems
  - Test automation (especially GUI-related)
    - guitar
    - selenium and products built on-top of selenium web-driver
    - capture/playback products.

Suggested Topics (tools cont’d)

- Slicing tools for testing and debugging
- Reliability testing
- Usability testing
- Integration testing (spring integration?)
- Real-time and Concurrency Testing (GroboUtils and related)
- Advanced static analysis tools.
- Behavior Driven Testing (BDT tools)
- Mobile Application Testing
- Security Testing
- Product line testing

Suggested topics (cont’d)

- Recent research
  - on combinatorial testing.
  - on mutation testing.
  - on algorithmic program debugging.
  - on slicing
  - on usability testing
  - on security testing
  - on test standardization
  - empirical evaluation of agile test quality
  - on testing big-data applications

For Ideas:

- Antonia Bertolino, “Software Testing Research: Achievements, Challenges, Dreams”
- query “software ______ testing tools” on wikipedia.
- query “glenford myers” on google scholar and then limit results to those newer than 2016.
  - looks for recent papers that reference a seminal testing work.
- query “Offutt” +”software testing” on google scholar and then limit results to those newer than 2016.
  - Similar queries based on authors who have contributed to a field that interests you. (Check the bibliographic notes at the end of a related chapter in Ammann & Offutt or in Mathur.

Background Reading

HTTP: the web protocol

- Basics of HTTP messages:
  - https://www.tutorialspoint.com/http/http_messages.htm
  - https://www.tutorialspoint.com/http/http_requests.htm
  - https://www.tutorialspoint.com/http/http_responses.htm
  - https://www.tutorialspoint.com/http/http_methods.htm
  - https://www.tutorialspoint.com/http/http_status_codes.htm
  - https://www.tutorialspoint.com/http/http_url_encoding.htm
- Examples & network level tools
  - https://wiki.wireshark.org/Hyper_Text_Transfer_Protocol
  - https://tools.ietf.org/html/rfc2616