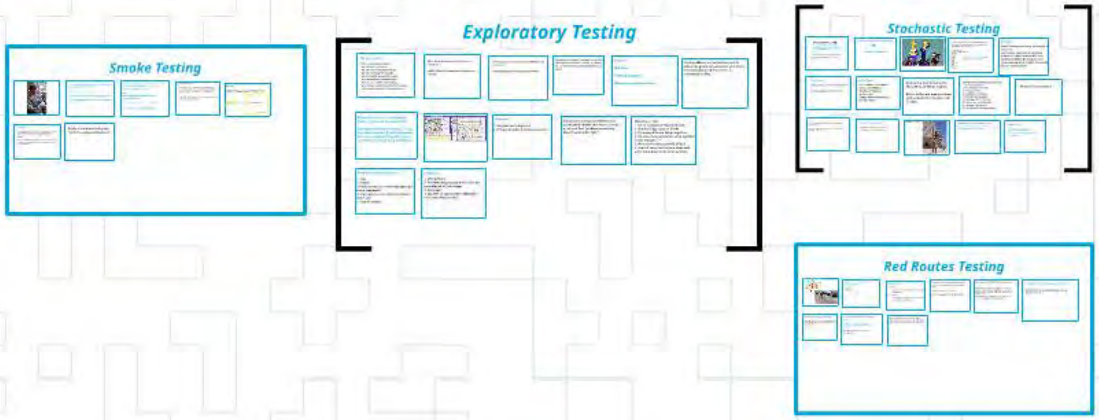
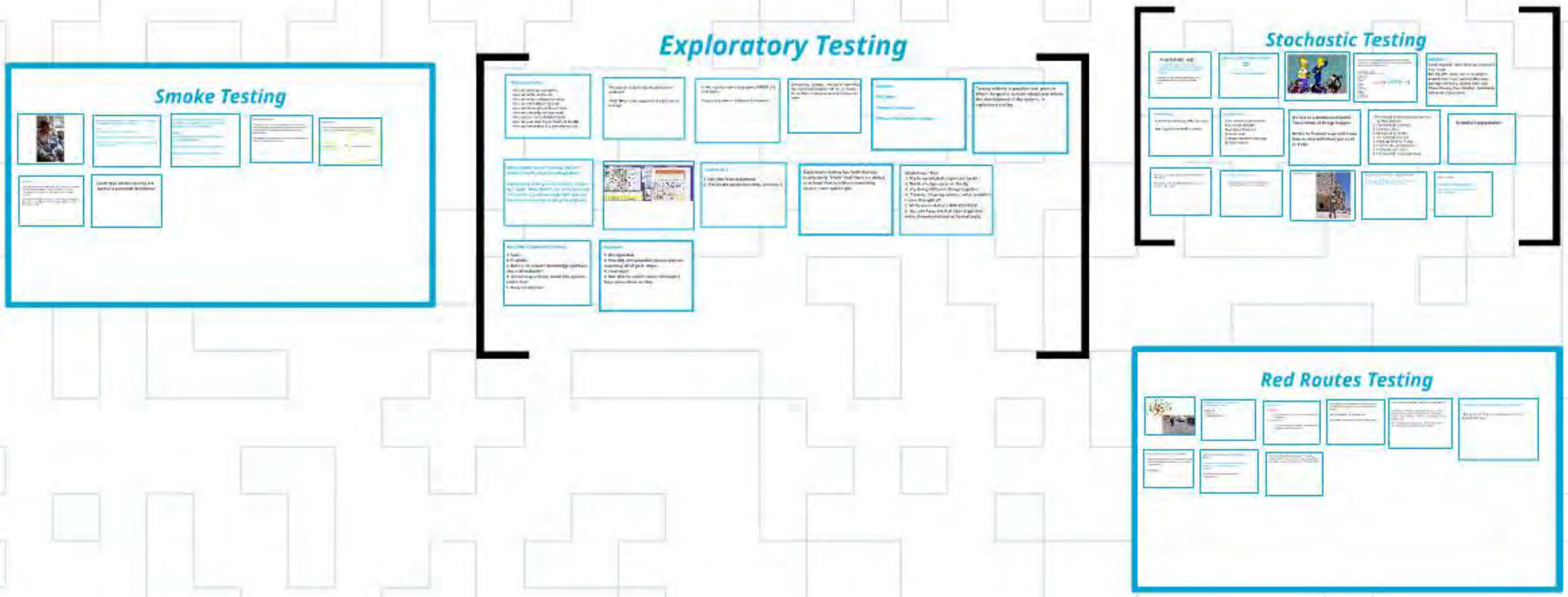
CS1699: Lecture 19 - Let's Get Informal: Exploratory, Smoke, Stochastic and Red Route Testing



CS1699: Lecture 19 - Let's Get Informal: Exploratory, Smoke, Stochastic and Red Route Testing



Exploratory Testing

The Story So Far...

We can develop test plans.
We can write unit tests.
We can write integration tests.
We can formally verify code.
We can check properties of code.
We can statically analyze code.
We can use combinatorial tests.
We can test over the network or locally.
We can determine ECs, boundaries, etc.

What do all these kinds of tests have in common?

HINT: What is the supposed *sine qua non* of testing?

A: We need to know the outcome BEFORE the test starts!

Expected behavior vs Observed Behavior

Sometimes, though... we don't know what the expected behavior will be, or should be, or there may be subjective factors at work.

Examples:

IRIX 4Dwm

CoMotion workspace

XMonad tiled window manager

Testing without a specified test plan, in which the goal is to learn about and inform the development of the system, is *exploratory testing*.

Often called "ad hoc" testing, but this implies carelessness and sloppy work.

Exploratory testing is not careless, it's just less rigid. More faith is put in the hands of the tester to go down alleys that may not have been seen before using the software.

How to do it

1. Use your best judgment.
2. If in doubt about next step, see step 1.

Exploratory testing has faith that you instinctively "know" that there's a defect, or at least that you know something doesn't seem quite right.

Guidelines / Tips:
1. Try to accomplish important tasks
2. Think of edge cases on the fly
3. Try doing different things together
4. If I were the programmer, what wouldn't I have thought of?
5. Write down defects IMMEDIATELY!
6. You can keep track of your steps and write them down later as formal tests.

Benefits of Exploratory Testing

1. Fast.
2. Flexible.
3. Relies on testers' knowledge (perhaps also a drawback?)
4. Great way to learn about the system under test
5. Easy-to-update!

Drawbacks

1. Unregulated.
2. Possibly unrepeatable (unless you are recording all of your steps).
3. Coverage?
4. Not able to automatized (although I have some ideas on this).

The Story So Far...

We can develop test plans.
We can write unit tests.
We can write integration tests.
We can formally verify code.
We can check properties of code.
We can statically analyze code.
We can use combinatorial tests.
We can test over the network or locally.
We can determine ECs, boundaries, etc.

What do all these kinds of tests have in common?

HINT: What is the supposed *sine qua non* of testing?

A: We need to know the outcome BEFORE the test starts!

Expected behavior vs Observed Behavior

Sometimes, though... we don't know what the expected behavior will be, or should be, or there may be subjective factors at work.

Examples:

IRIX 4Dwm

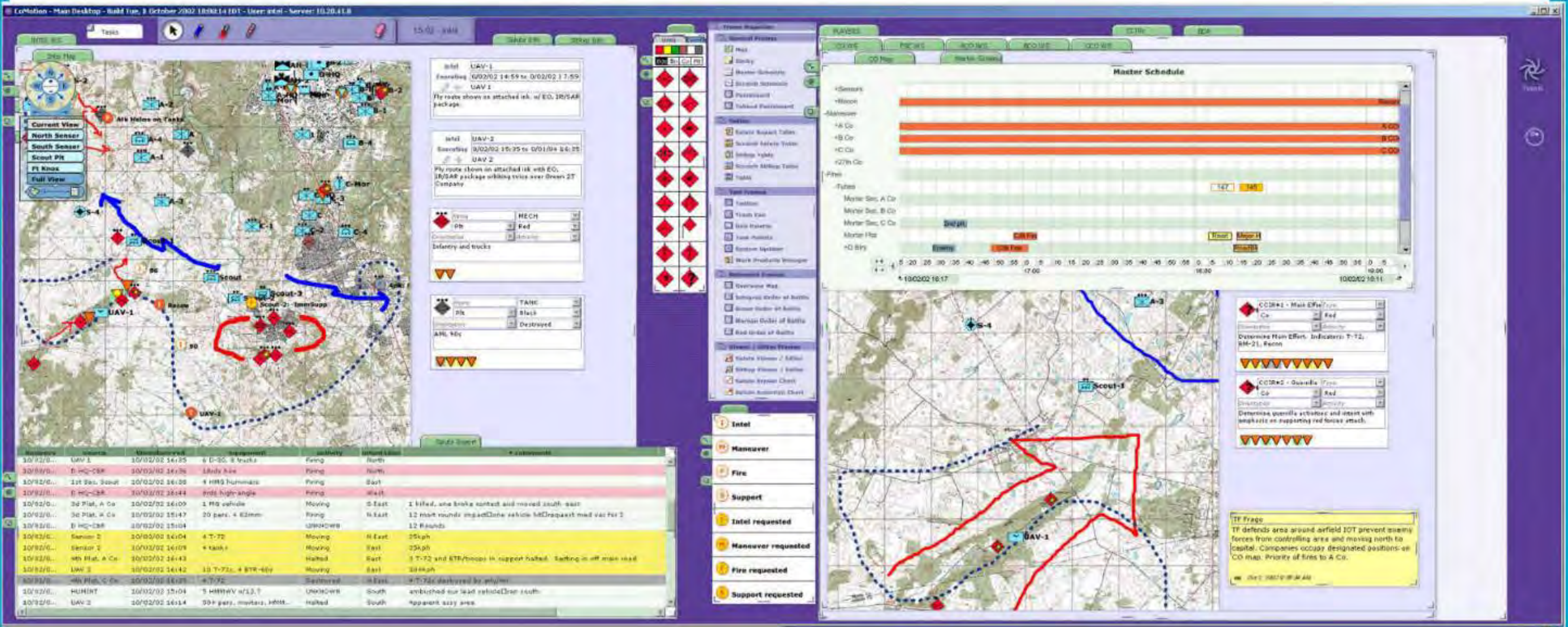
CoMotion workspace

XMonad tiled window manager

Testing without a specified test plan, in which the goal is to learn about and inform the development of the system, is *exploratory testing*.

Often called "ad hoc" testing, but this implies carelessness and sloppy work.

Exploratory testing is not careless, it's just less rigid. More faith is put in the hands of the tester to go down alleys that may not have been seen before using the software.



CoMotion - Main Desktop - Build Tue, 1 October 2002 18:58:14 EDT - User: intel - Server: 10.20.41.8

[illegible]	source	timeObserved	equipment	activity	[illegible]	comments
10/02/0...	UAV 1	10/02/02 16:35	6 D-30, 8 trucks	Firing	North	
10/02/0...	B HQ-CBR	10/02/02 16:36	18rds [illegible]	Firing	North	
10/02/0...	1st Sec. Scout	10/02/02 16:38	4 HMG hummers	Firing	East	
10/02/0...	B HQ-CBR	10/02/02 16:44	6rds high-angle	Firing	West	
10/02/0...	3d Plat. A Co	10/02/02 16:09	1 PS vehicle	Moving	S East	1 killed, one broke contact and moved south east
10/02/0...	3d Plat. A Co	10/02/02 15:47	20 pers. 4 82mm	Firing	N East	12 mort rounds impact□one vehicle hit□request med vac for 2
10/02/0...	B HQ-CBR	10/02/02 15:04		UNKNOWN		12 Rounds
10/02/0...	Sensor 2	10/02/02 16:04	4 T-72	Moving	N East	25kph
10/02/0...	Sensor 1	10/02/02 16:09	4 tanks	Moving	East	25kph
10/02/0...	4th Plat. A Co	10/02/02 16:43		Halted	East	3 T-72 and BTR/troops in support halted. Setting in off main road
10/02/0...	UAV 2	10/02/02 16:42	10 T-72s, 4 BTR-60s	Moving	East	30kph
10/02/0...	4th Plat. C Co	10/02/02 16:25	4 T-72	Destroyed	N East	4 T-72s destroyed by [illegible]
10/02/0...	HUMINT	10/02/02 15:04	5 HMMWV w/12.7	UNKNOWN	South	ambushed our lead vehicle□on south
10/02/0...	UAV 2	10/02/02 16:14	50+ pers. mortars, HMM...	Halted	South	Apparent assy area

Master Schedule

TF Frago
TF defends area around airfield IOT prevent enemy forces from controlling area and moving north to capital. Companies occupy designated positions on CO map. Priority of fires to A Co.

How to do it

1. Use your best judgment.
2. If in doubt about next step, see step 1.

Exploratory testing has faith that you instinctively "know" that there's a defect, or at least that you know something doesn't seem quite right.

Guidelines / Tips:

1. Try to accomplish important tasks
2. Think of edge cases on the fly
3. Try doing different things together
4. If I were the programmer, what wouldn't I have thought of?
5. Write down defects IMMEDIATELY!
6. You can keep track of your steps and write them down later as formal tests.

Benefits of Exploratory Testing

1. Fast.
2. Flexible.
3. Relies on testers' knowledge (perhaps also a drawback?)
4. Great way to learn about the system under test
5. Easy-to-update!

Drawbacks

1. **Unregulated.**
2. **Possibly unrepeatable (unless you are recording all of your steps).**
3. **Coverage?**
4. **Not able to automatized (although I have some ideas on this).**

Smoke Testing

Smoke testing (PLUMBING): send smoke down the pipes to find leaks BEFORE sending water or other fluids

WHY?
Much easier to clean up smoke than water.

Won't waste effort - hooking up to a water main is non-trivial

Won't cause further damage (high-pressure water going through a hole -> bigger hole)

Smoke testing (software): Do some minimal testing to ensure that the system is, in fact, testable, or ready to be released

WHY?
No need to test system that can't perform minimal functionality.

Setting up test harnesses etc. is non-trivial.

May waste time going down blind alleys.

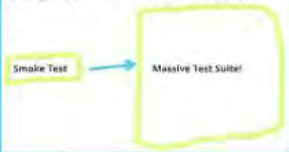
Smoke testing can be:

1. Scripted : There are a few small but important test cases (taking an hour or two to execute, at most) which are run prior to the software being released to the greater team.

2. Unscripted: An experienced tester does exploratory or ad hoc testing for an hour or so.

Keep in mind:

Smoke testing is an ADDITION to traditional software testing. It is a GATEWAY to further testing or release.



Sanity Testing

A really, really basic smoke test - e.g., can the CD be read (media check)? Will the program install? Can the program be executed? Are all expected files on the server?

NB: Some texts say that "smoke" and "sanity" testing are synonymous, but these are the ways in which I've used them in industry.

Sanity and smoke testing are "*part* of a complete breakfast."

NYPD

Smoke testing (PLUMBING): send smoke down the pipes to find leaks BEFORE sending water or other fluids.

WHY?
Much easier to clean up smoke than water.

Won't waste effort - hooking up to a water main is non-trivial.

Won't cause further damage (high-pressure water going through a hole -> bigger hole).

Smoke testing (software): Do some minimal testing to ensure that the system is, in fact, testable, or ready to be released.

WHY?
No need to test system that can't perform minimal functionality.

Setting up test harnesses etc. is non-trivial.

May waste time going down blind alleys.

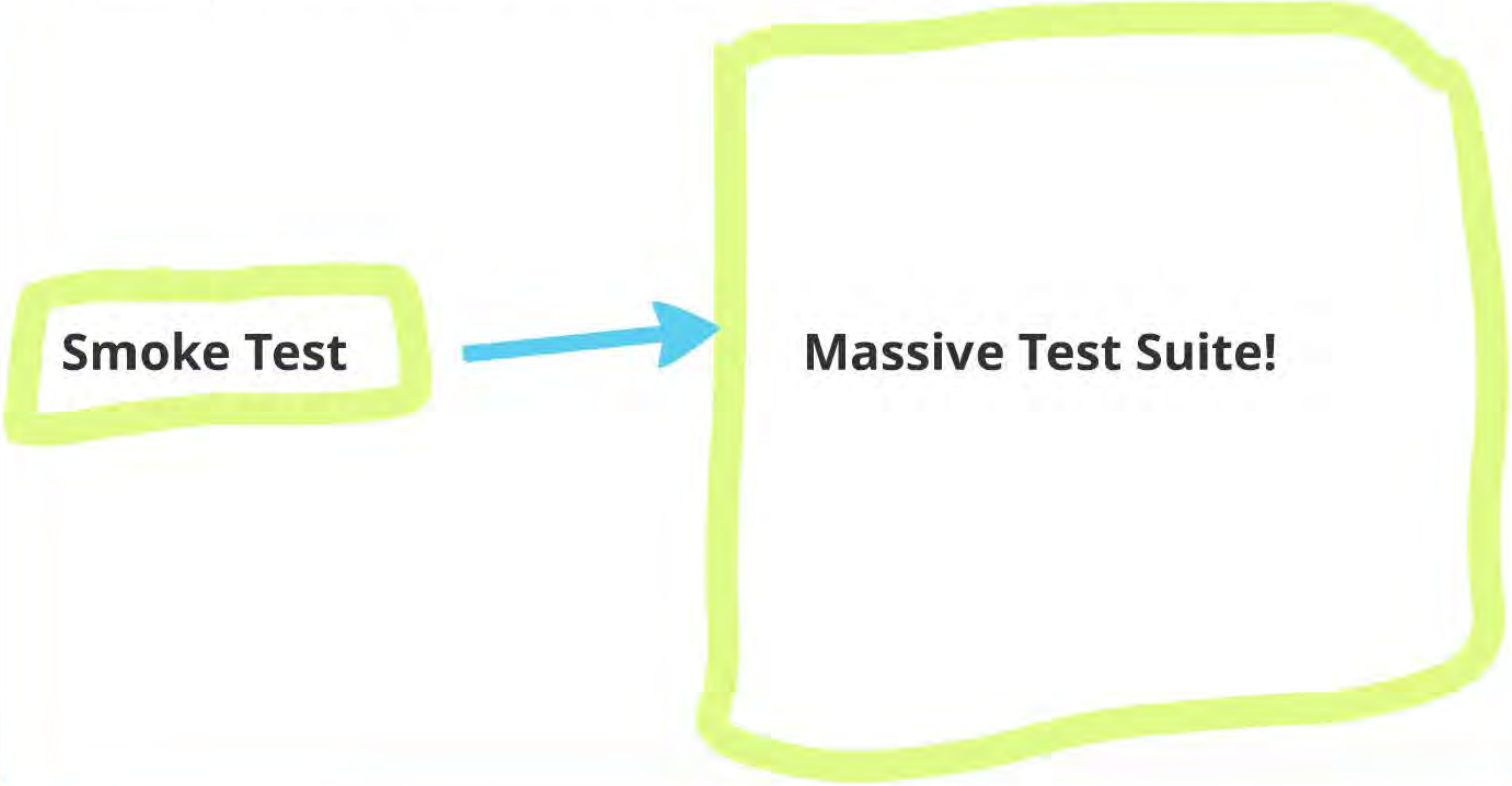
Smoke testing can be:

1. Scripted : There are a few small but important test cases (taking an hour or two to execute, at most) which are run prior to the software being released to the greater team.

2. Unscripted: An experienced tester does exploratory or ad hoc testing for an hour or so.

Keep in mind:

Smoke testing is an ADDITION to traditional software testing. It is a GATEWAY to further testing or release.



Sanity Testing

A really, really basic smoke test - e.g., can the CD be read (media check)? Will the program install? Can the program be executed? Are all expected files on the server?

NB: Some texts say that "smoke" and "sanity" testing are synonymous, but these are the ways in which I've used them in industry.

Sanity and smoke testing are "*part* of a complete breakfast."

Stochastic Testing

stochastic, *adj.*

randomly determined; having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely

mid 17th cent.: from Greek stokhastikos, from stokhazesthai 'aim at, guess,' from stokhos 'aim.'

Infinite monkey + Infinite typewriters

=

the works of Shakespeare

Think of stochastic testing as property based testing, with few or no limits on input, and the only invariant is "the system keeps running!"

```
hurewhjwehjkfwnjk
[illegible]
mehewhiu13
fewjwekr3
ew9993
{"json":}....
[illegible]
fehiee
[illegible]
njkfnuh9t3
nioewh&r3
e=++)@5
N(#HIFHEUJE
```

$\longrightarrow$ *SYSTEM* =)

Variants:

Smart Monkey - does what you expect a user to do

Evil Monkey - Goes out of its way to provide bad input (executable code, strange numbers, binary data, etc)

Chaos Monkey (from Netflix) - Randomly kill servers/processes

Chaos Monkey

Randomly terminates AWS instances

Run regularly on Netflix servers

Related Tools

Cut network connectivity
Reduce bandwidth
Modify permissions
Remove user
Change network topology
Induce latency

We live in a distributed world. These kinds of things happen.

Better to find out now and know how to deal with than get a call at 3 AM.

"The Fallacies of Distributed Computing"
-by Peter Deutsch

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

Graceful Degradation

"The best way to avoid failure is to fail constantly." - Jeff Atwood

"If you want to make a difficult task easier, do it all the time." -Bill Laboon

The more you deal with a problem:

1. The more you know HOW to deal with it
2. The easier it is to automate it away

Example: Generators

Computers have moved from being *pets* to *cattle*.

Testing large, distributed systems means treating your systems as fungible "units of computation".

Example: Google

http://googletesting.blogspot.com/

"How Google Tests Software" by Whittaker, Arbon, and Carollo

stochastic, *adj.*

randomly determined; having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely.

mid 17th cent.: from Greek stokhastikos, from stokhazesthai 'aim at, guess,' from stokhos 'aim.'

Infinite monkey + Infinite typewriters

=

the works of Shakespeare

Think of stochastic testing as property based testing, with few or no limits on input, and the only invariant is "the system keeps running!"

hurewhjwehjkfwehjk
fesnjfsenjlkfei9832y80qewnkfew
ewhewhiut3
fewjwe8r3
ew9983
("json") ....
{}{#U(d
fehioe
U()HIOFDNkn
njkfniuh9t3
nioewh0r3
e=++)@$
N((#HIFHEUIE

$\longrightarrow$ ***SYSTEM*** **=)**

Variants:

Smart Monkey **- does what you expect a user to do**

Evil Monkey **- Goes out of its way to provide bad input (executable code, strange numbers, binary data, etc)**

Chaos Monkey (from Netflix) **- Randomly kill servers/processes**

Chaos Monkey

Randomly terminates AWS instances

Run regularly on Netflix servers

Related Tools

Cut network connectivity

Reduce bandwidth

Modify permissions

Remove user

Change network topology

Induce latency

We live in a distributed world. These kinds of things happen.

Better to find out now and know how to deal with than get a call at 3 AM.

"The Fallacies of Distributed Computing"
-by Peter Deutsch

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

Graceful Degradation

"The best way to avoid failure is to fail constantly." - Jeff Atwood

"If you want to make a difficult task easier, do it all the time." -Bill Laboon

The more you deal with a problem:

1. **The more you know HOW to deal with it**
2. **The easier it is to automate it away**

Example: Generators

Computers have moved from being *pets* to *cattle*.

Testing large, distributed systems means treating your systems as fungible "units of computation".

Example: Google

http://googletesting.blogspot.com/

"How Google Tests Software" by Whittaker, Arbon, and Carollo

Red Routes Testing

Red Routes -> The important paths of functionality in a system

Should be:
- CRITICAL
- FREQUENTLY USED

Example: Amazon

Red Route:

Log in, select book, check out -> book should be delivered

Not a Red Route:

Log in and log out 900 times -> determine if shopping cart icon looks OK

Comes from usability testing, but often used for "traditional" testing to focus on areas greatest import

Not as specific as a classic test plan

Puts a little leeway into the hands of the tester

Red routes are COMPLETE ACTIVITIES, not specific tasks.

"Select book" or "log in" are important, but not red routes. There should be a key business or customer objective embedded in the path (e.g., "buy book", "write review", etc.).

For a web app, for example, you will usually need to visit several pages (this is just a heuristic!).

Red Routes have an objective accomplishment

"Receive book" is an accomplishment, "site is easy to use" is not.

Red routes focus on GOALS, not ACTIONS.

Bad: Type "JoeUser123" in username box, type "AwesomePassword" in password box, click "login" button.

Good: Log in.

Red routes should be portable to competing systems

You should be able to log in, buy a book, write a review, etc. on barnesandnoble.com, for example

Note that this doesn't have to be a 1:1 correspondence

Red route testing lets you ensure that the key functionality of a site or system is working without exhaustive testing of every aspect of functionality.

Red Routes -> The important paths of functionality in a system

Should be:

- **CRITICAL**
- **FREQUENTLY USED**

Example: Amazon

Red Route:

Log in, select book, check out -> book should be delivered

Not a Red Route:

Log in and log out 900 times -> determine if shopping cart icon looks OK

Comes from usability testing, but often used for "traditional" testing to focus on areas greatest import

Not as specific as a classic test plan

Puts a little leeway into the hands of the tester

Red routes are COMPLETE ACTIVITIES, not specific tasks.

"Select book" or "log in" are important, but not red routes. There should be a key business or customer objective embedded in the path (e.g., "buy book", "write review", etc.).

For a web app, for example, you will usually need to visit several pages (this is just a heuristic!).

Red Routes have an objective accomplishment

"Receive book" is an accomplishment, "site is easy to use" is not.

Red routes focus on GOALS, not ACTIONS.

Bad: Type "JoeUser123" in username box, type "AwesomePassword" in password box, click "login" button.

Good: Log in.

Red routes should be portable to competing systems

You should be able to log in, buy a book, write a review, etc. on barnesandnoble.com, for example.

Note that this doesn't have to be a 1:1 correspondence

Red route testing lets you ensure that the key functionality of a site or system is working without exhaustive testing of every aspect of functionality.

CS1699: Lecture 19 - Let's Get Informal: Exploratory, Smoke, Stochastic and Red Route Testing