

# Software Robustness Tutorial

## Testing of Complex Telecommunications Solutions

Vincent Sinclair/Abhaya Asthana

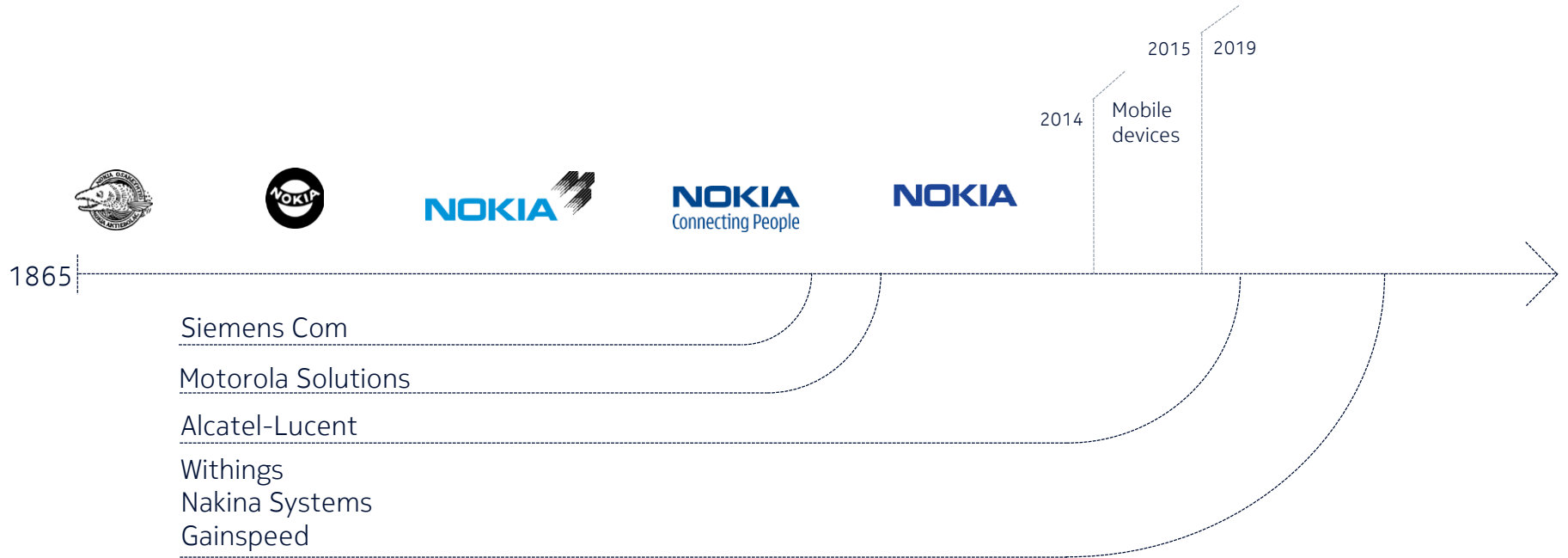
24-03-2019

# Topics

- 1. Reliability in Telecommunications Networks**
  - Overview of Nokia
  - Current & Future Telecommunications Networks
  - Reliability Requirements in Telecommunications Networks
- 2. Software Robustness**
  - Definition
  - Examples of Software Robustness Defects
  - Origin of Software Robustness Defects
- 3. Software Robustness Testing**
  - Role of Software Robustness Tester
  - Software Robustness Testing as a Distributed Activity
- 4. Building a Software Robustness Test Plan, including Fault Modelling**
  - Inputs to the software robustness test plan
  - Software Robustness Test Case Examples
  - Extending Typical Stability Run
  - Procedural Reliability
  - Challenges for testers
- 5. Conclusions**

# 1. Reliability in Telecommunications Networks

# Nokia: Long History of Successful Change



## At the forefront of every fundamental change in how we communicate and connect

### Telephony begins

### Analog revolution

### Digital revolution

### Mobile revolution

### The new connectivity

---

#### Long distance voice communication

#### Voice, data, and video communication

#### Wireless communication

#### Intelligent and seamless connectivity through the Cloud

Bell Telephone Laboratories formed in 1925

- Copper networks
- Circuit switches
- Amplifiers

- Laser
- Satellite communications
- UNIX
- DWDM
- 100Gbps optical transport
- 400G routers

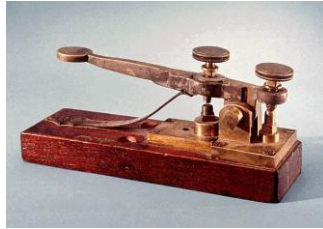
- First ever calls on GSM and LTE
- First car phone
- Commercialization of Small Cells
- MIMO

- 5G
- G.Fast: 1Gbps over copper
- Optical super channels
- Terabit IP routing
- Datacenter infrastructure and applications for the Cloud
- Smart sensors for the Internet of Things

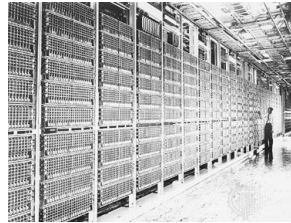
# The past

Breathtaking rate of innovation in communication devices and networks

Devices



Networks



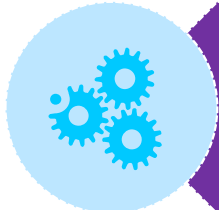
# BELL LABS IRELAND: RESEARCH STRATEGY



Redefining user experience



Future-proof flexible networks



Massively capable infrastructure


Infallible context awareness in real-time




“Drop-and-forget” small cells




Cognitive cloud: Zero-touch control




Energy-autonomous infrastructure



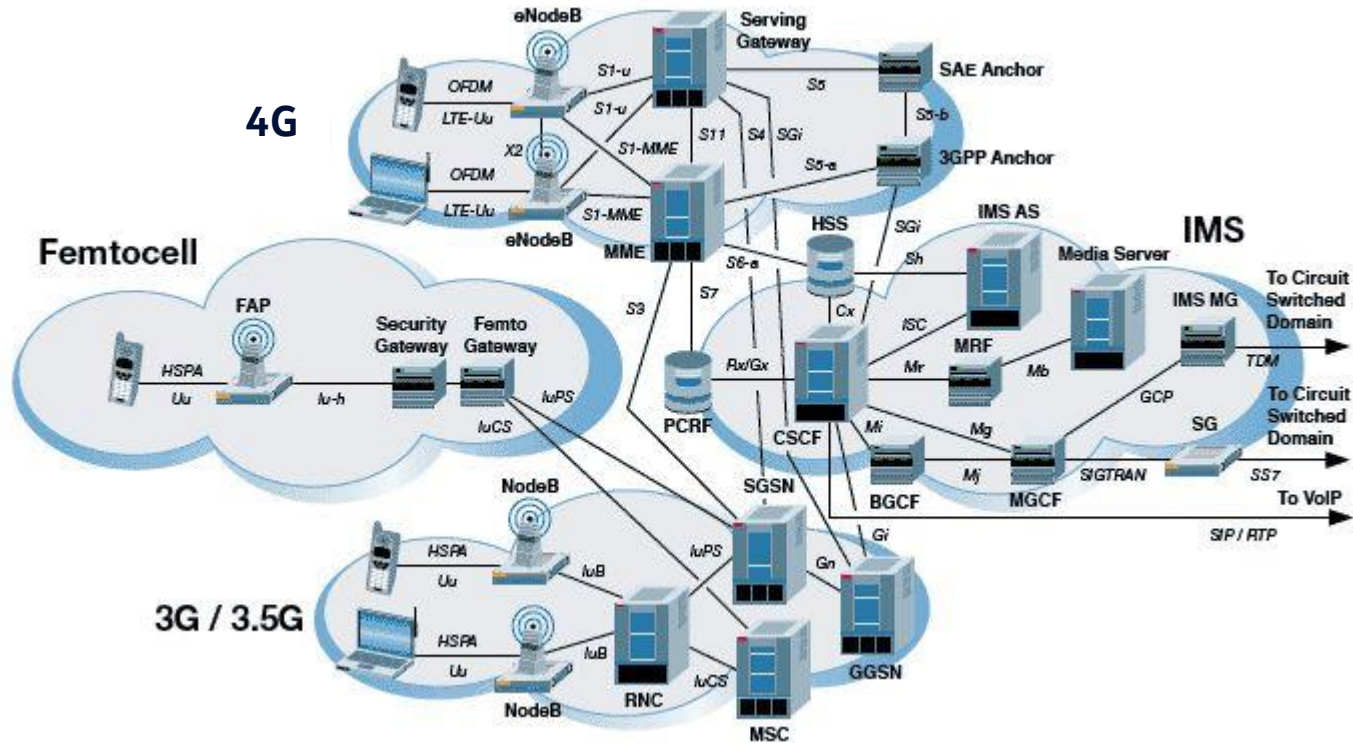
Zero-footprint thermal management



Massive MIMO: Squeezing the last from spectrum



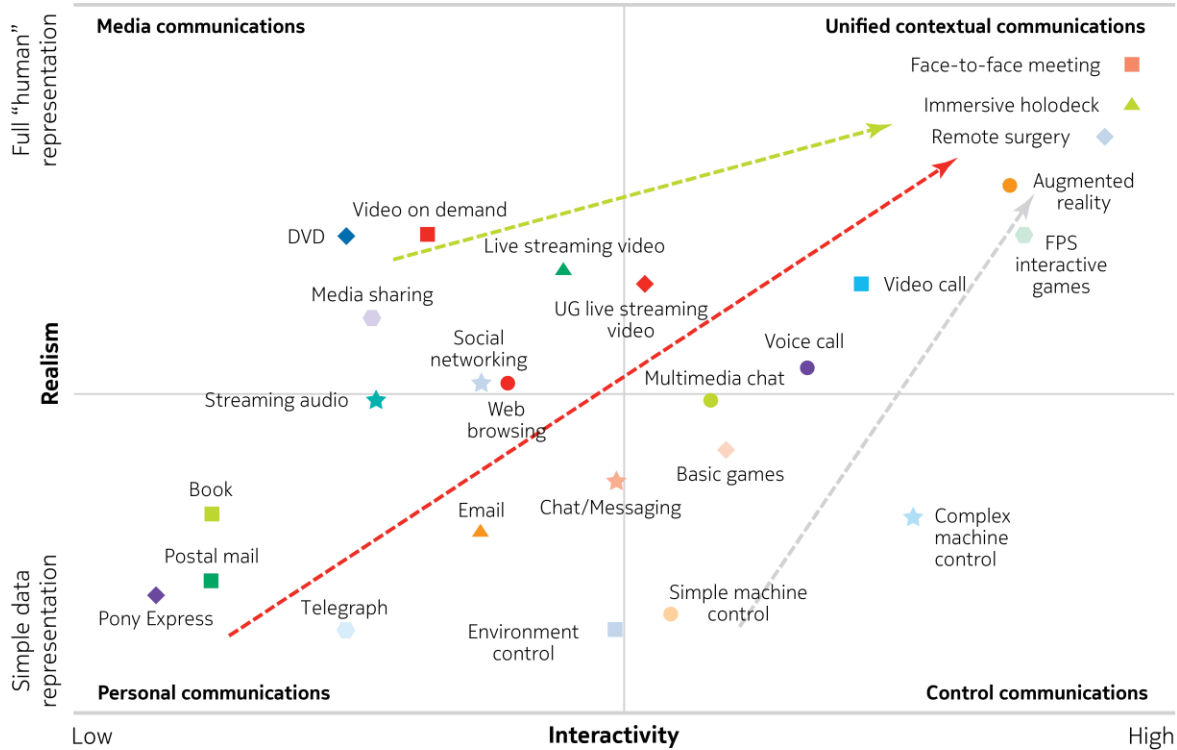
# Current Telecommunications Networks



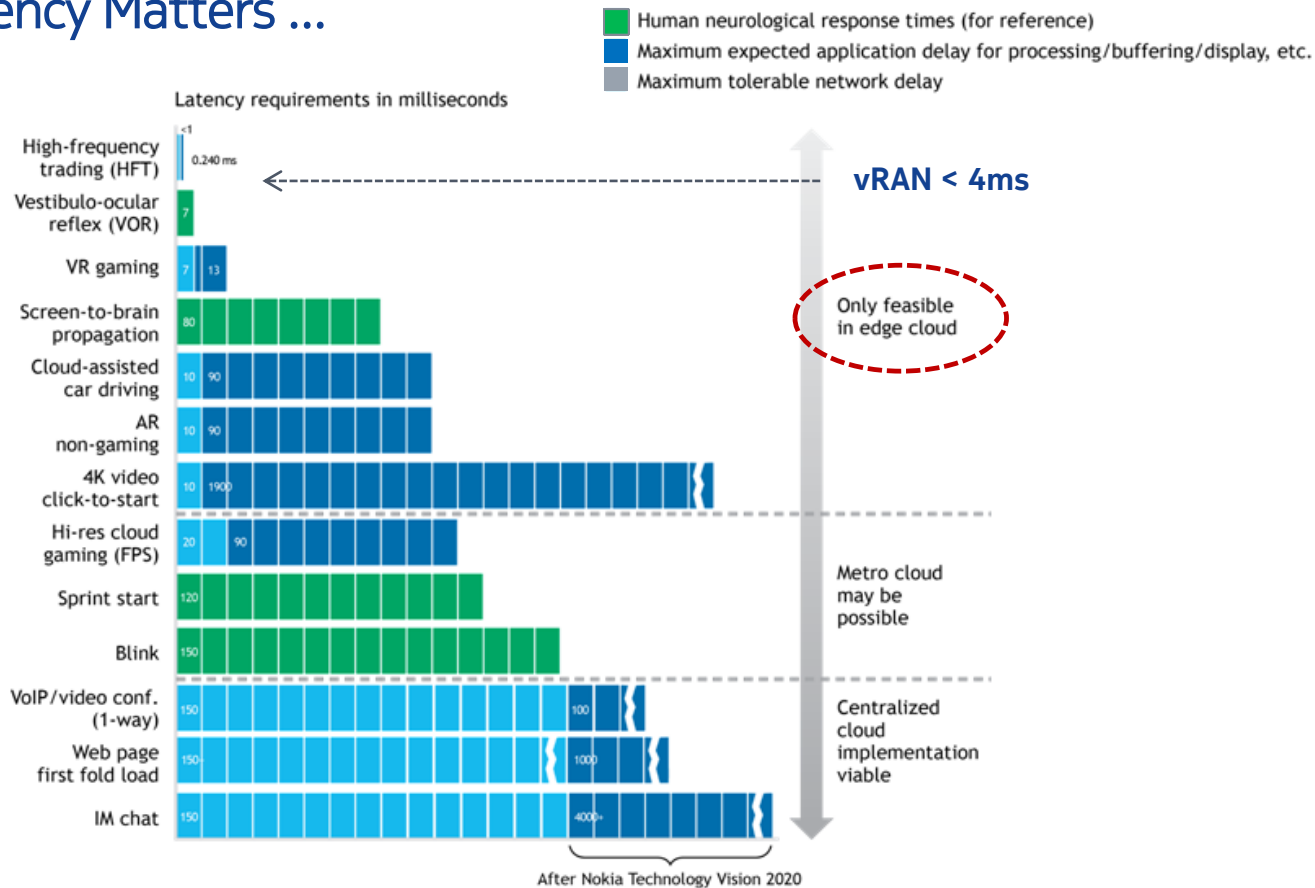


# The future of communications

## Rich, interactive, unified contextual communications



# Future Network: Latency Matters ...



# The Future is Different than the Past...”or we’re not in Kansas anymore”

	<b>Past</b>	<b>Future</b>
<b>Solutions</b>	Technology-driven	Human/Business need driven
<b>Driver</b>	Consumer (BW)	Industry (Latency & SLA)
<b>Architecture</b>	Heavily Centralized	Massively Distributed
<b>Partnership</b>	Limited APIs	Co-design & Open specs
<b>Standards</b>	Lead	Follow
<b>Investment</b>	Singular (Operator only)	Multiple & Cooperative (Many contributors/new players)
<b>Flexibility</b>	Limited (Provisioned)	Large (Software definable)
<b>Sharing</b>	Static and Limited (HW VPNs )	Dynamic and Infinite (SW Slices)
<b>Innovation Speed</b>	Per annum/decade (new services)	Per hour/day (new apps)

## 2. Software Robustness

## Robustness in Telecommunication Networks – Network Focus

- Measure is cell availability, excluding the blocked by user state (BLU).
- It gives the percentage of available time compared to the total time that cell should be available.
- The counter is incremented by 1 approximately every 10 seconds when "Cell Operational State is enabled".
- The counter is incremented with value 1 approximately every 10 seconds when cell "Administrative State is locked" or "Energy State is energySaving" or "Local State is blocked".

# Robustness in Telecommunication Networks – User Focus

Telco Operator view of reliability is changing, driven by increasing end user expectations:

Move from landline to mobile communications

Dependency on mobile communications

	Past	Future
Reliability	Network Element Focus	End User Services Focus
Quality	Number NE Defects/Outages	Number Users Affected
Measures	Five 9's	% successful services delivered

Today, many Telco companies have performance focused on KPIs such as:

Call set-up success rate, Call drop rate, Session set-up rate, Session retain rate.

Challenge – Deliver software to meet future reliability expectations amidst increasing complexity

# Software Robustness Outages – Non Telecom

- 2019: Change of time in spring to daylight time caused a software crash for parking payment system. 300 car parks could not charge for two days and had to provide free parking.
- 2015: Plane Crash caused by computer configuration files being accidentally wiped from three engines. The files needed to interpret the engine readings were deleted by mistake. This caused the affected propellers to spin too slowly
- 2014: Many RBS, NatWest and Ulster Bank customers locked out of their bank accounts. To prevent a repeat, an additional £450m was devoted specifically to "increasing the system's resilience"
- 2013: Airline traffic control system fails. The breakdown occurred when the National Air Traffic Service (NATS) computer system was making the switchover from the quieter night time mode to the busier daytime setup. It was unable to handle the normal volume of flights for a Saturday.
- 2013: Toyota firmware defect caused cars to accelerate unintentionally.

# Software Robustness Outages – Telecom

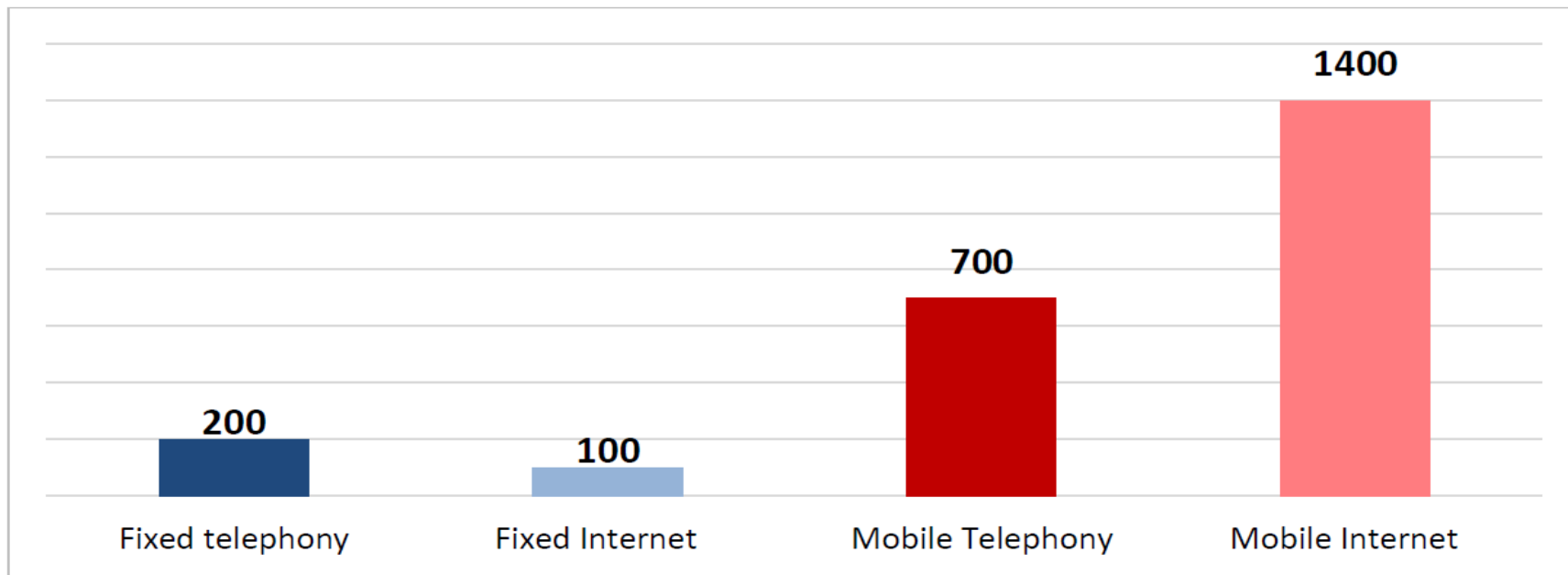
## 90 major incidents reported in EU (2013)

- 19 countries reported 90 significant incidents and 9 countries reported no significant incidents.
- Mobile networks most affected: Approximately half of the major incidents had an impact on mobile Internet and mobile telephony.
- Mobile network outages affect many users:
  - 1.4 million users affected for each outage (data)
  - 700 000 users affected for each outage (voice)
- Impact on emergency calls:
  - A fifth of the major incidents had an impact on the emergency calls (112 access –911 access in USA/Canada)
- Looking more in detail, the detailed causes affecting most user connections were
  - software misconfiguration
  - software bugs
  - power surges

Source: European Union Agency for Network and Information Security Annual Incident Reports 2013

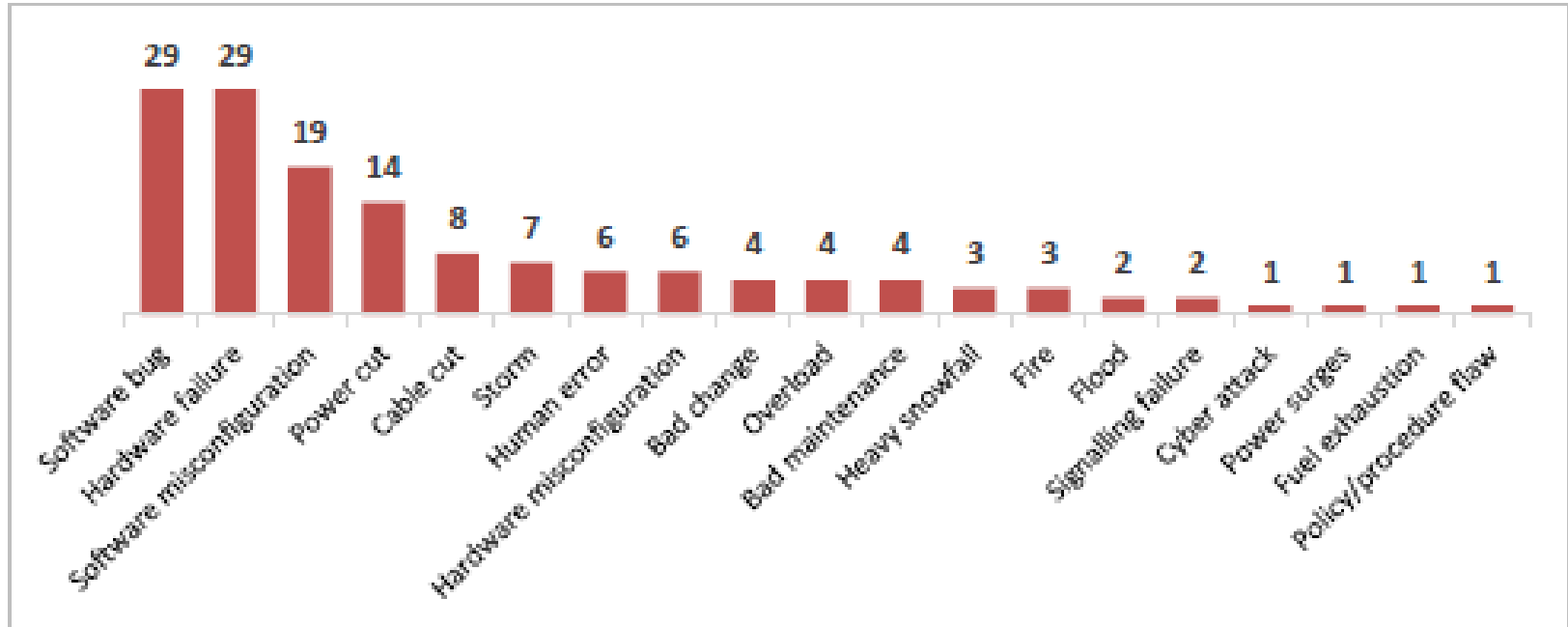


## Users Affected per Individual Outage (000'S)



Source: European Union Agency for Network and Information Security Annual Incident Reports 2013

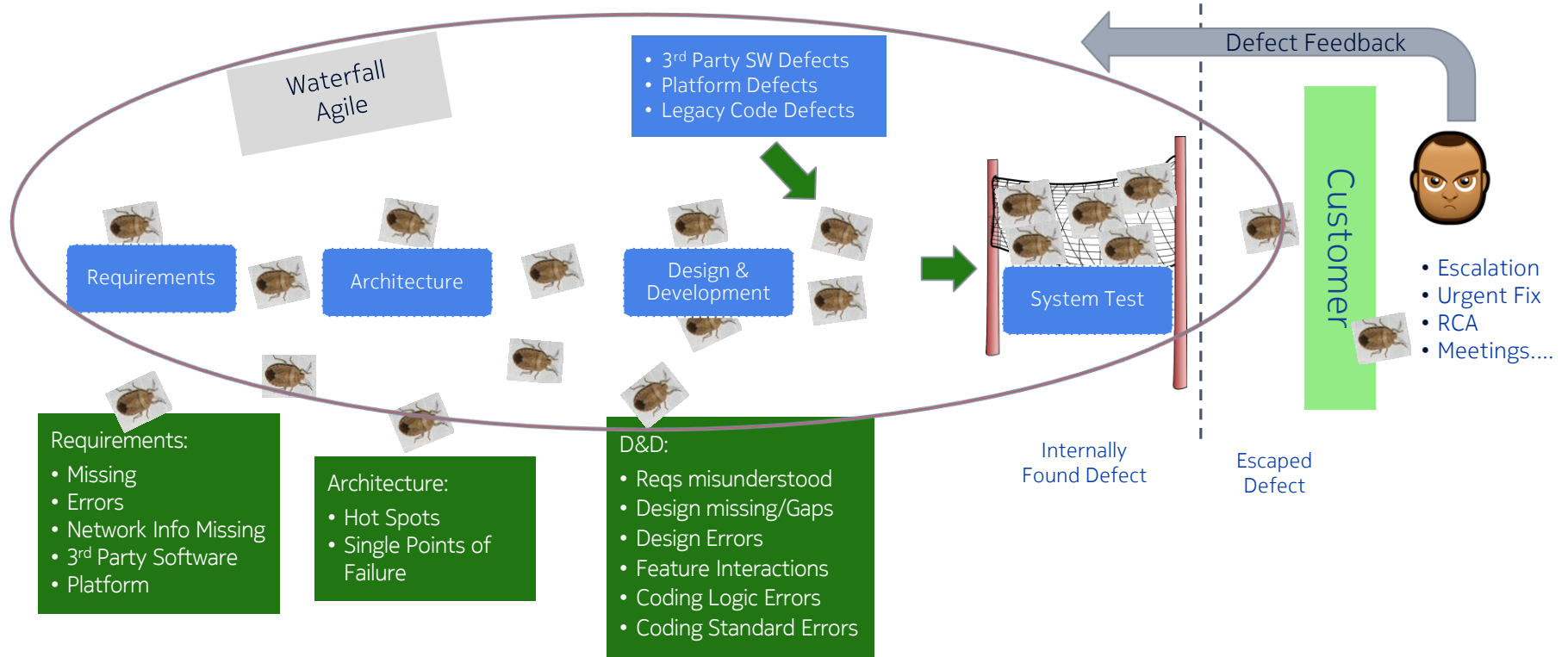
# Root Causes



Source: European Union Agency for Network and Information Security Annual Incident Reports 2013

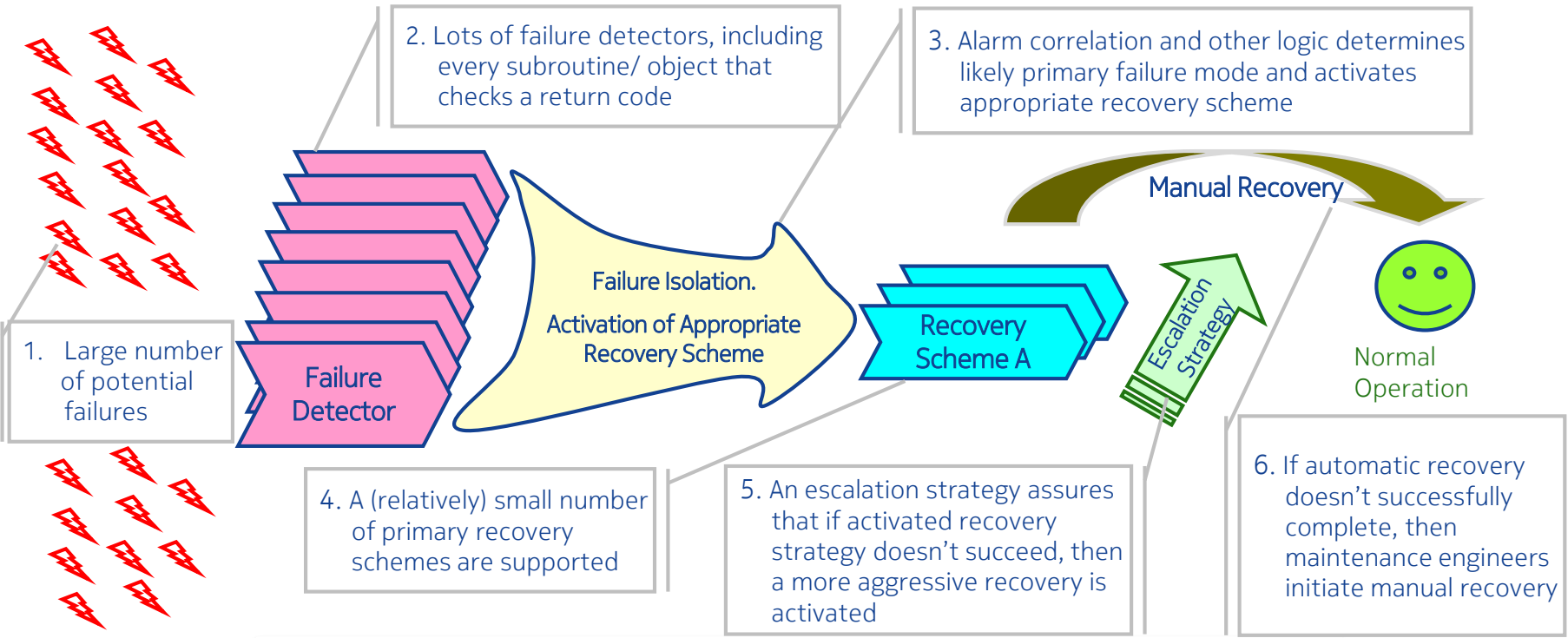
Where do these software defects originate ?

# Defect Life Cycle



What happens when an outage occurs ?

# Real World View of Failures and Recovery



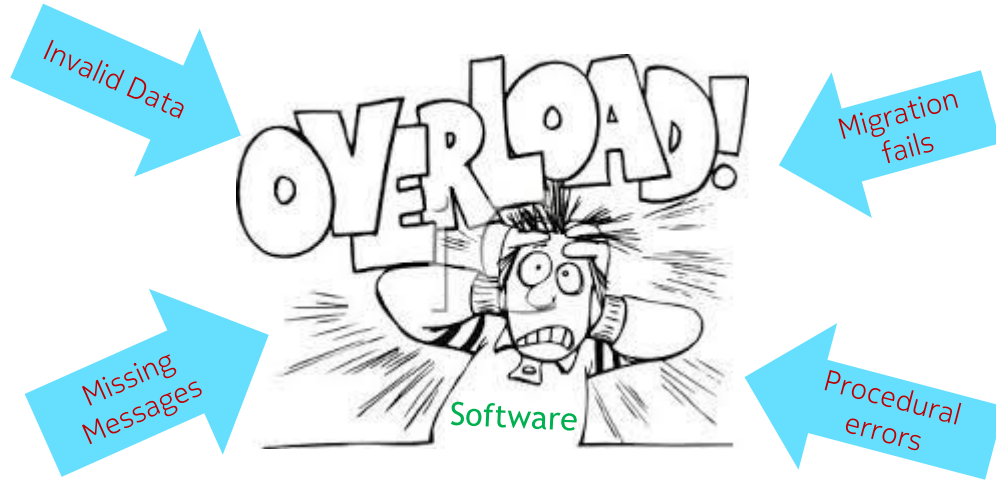
Challenge for tester: Build a test plan to test the most critical failures

# 3. Software Robustness Testing

# Why Software Robustness Testing?

- Customer perception:
  - End users are more and more dependent on having reliable telecommunications services
  - Telco companies are demanding higher and higher levels of reliability
- Traditional functional testing strives to minimize the number of residual defects in product
- Inevitably, latent defects 'leak' into the field, causing in-service failures
- Strategy: Confront running system with realistic fault events to verify that system automatically detects and recovers rapidly with minimal overall impact on service
- Failure acceleration:
  - Intentionally inserting fault to trigger the fault recovery can achieve more thorough testing in a controlled environment and within a reasonable time frame
  - Identify software robustness defects
  - Can identify design flaws and provide feedback to the design teams to improve fault detection, isolation and recovery

# Role of Software Robustness Tester



Fault Tolerant Mindset

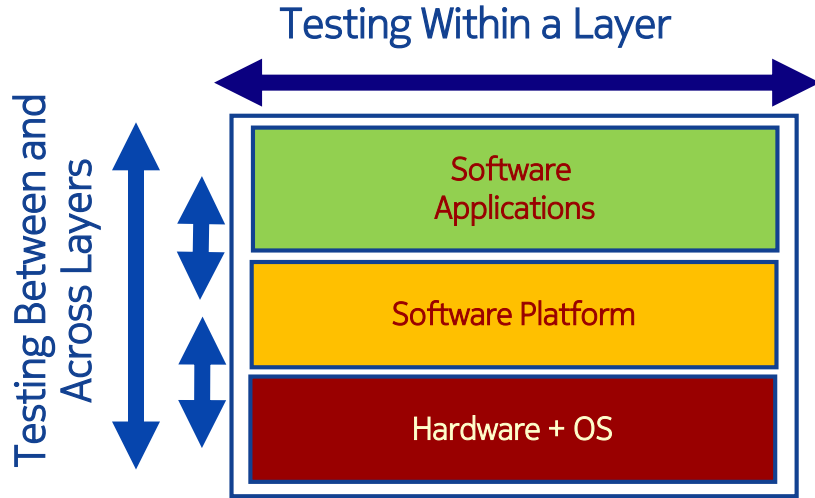
This is a different way of thinking

What are the critical failures that can realistically occur ?

How can I stress/break the software to trigger those failures ?

How can I test the detection, isolation and recovery from software failure ?

# Typical Software System Structure



Each Higher Layer is responsible for testing:

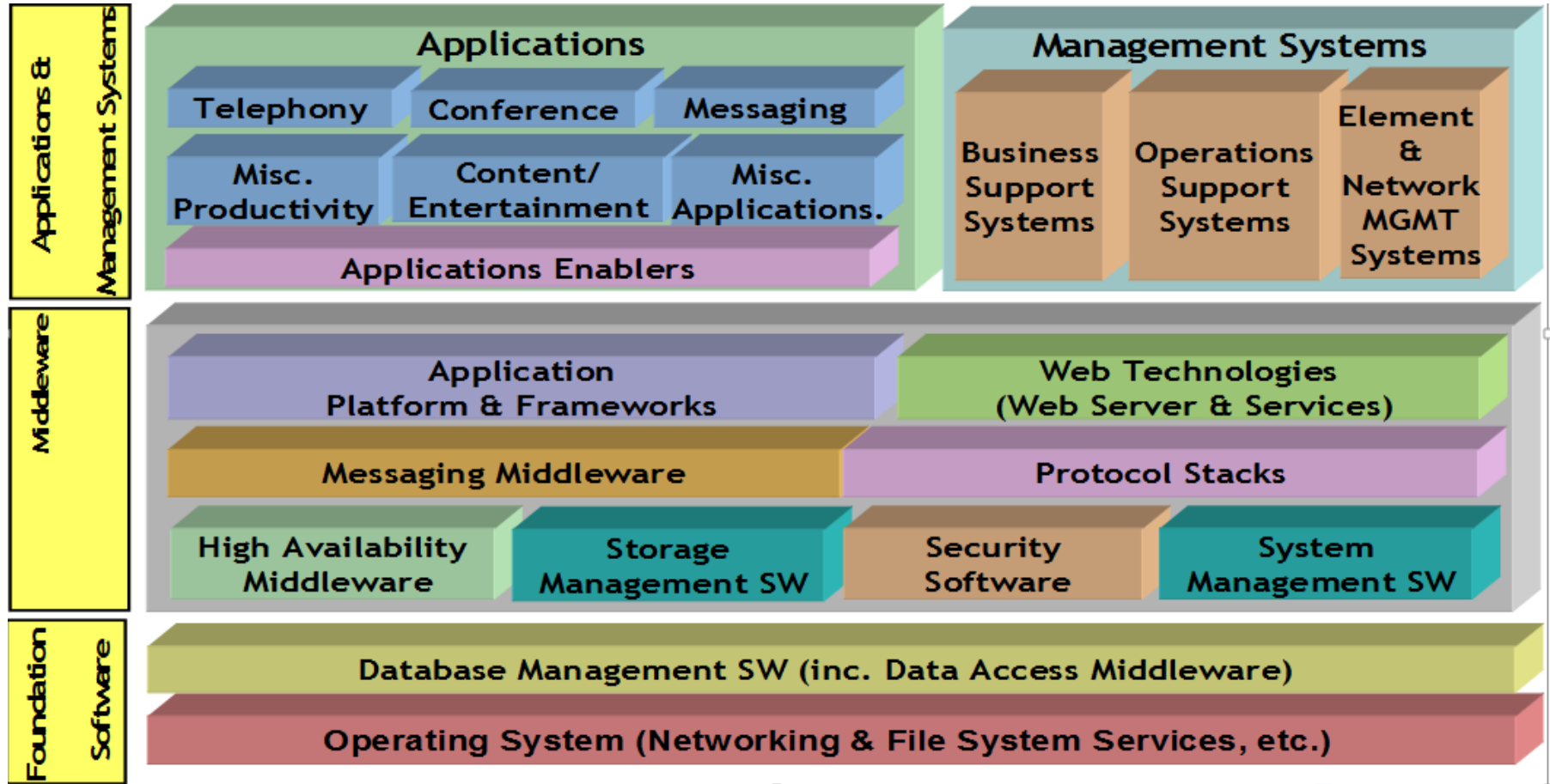
- within layers
- between layers
- across the lower layers

- Layers may be developed in separate organizations. (Platform, Middleware, Application)
- Some components may come from external software suppliers
- A failure in one layer may need to be detected and recovered from a different layer.
- We need to test that there is a solid connection between the failure detection and the failure recovery mechanism.
- For example, a failure in application layer may need a recovery action in the software platform layer.

So, where do we start ?



# Telecomm Software Stack

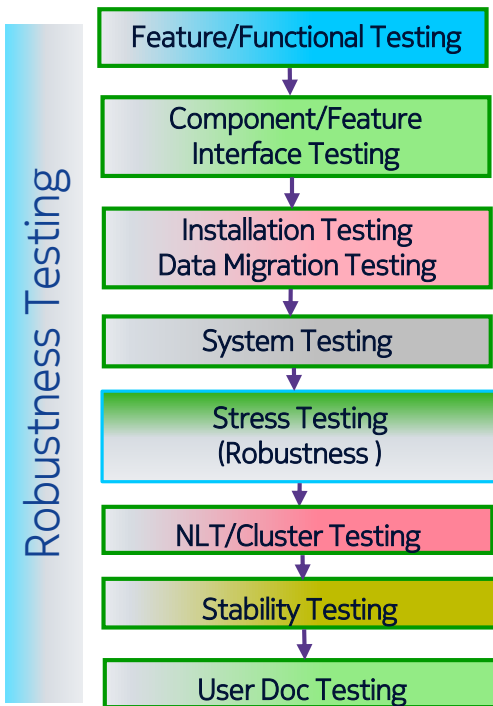


# Software Robustness Testing - Distributed Across Many Test Areas

Invalid inputs  
Duplication  
Stressful environment  
Network congestion  
Timing issues  
.....



Fault Insertion Testing  
Negative, Adversarial,  
Breakage, Chaos, Free



Testing features/functions with invalid inputs. Test under stress.

Testing component interfaces for robustness to invalid inputs, message errors, timing errors, missing/duplicate inputs.

Testing of the installation and configuration process. What could go wrong? What could interrupt data migration?

System functionality testing with invalid inputs or system under stress. What human errors can be made during OA&M procedures?

Stressing under high traffic load to trigger control mechanisms to Detect, Isolate, Recover. Testing with invalid inputs.

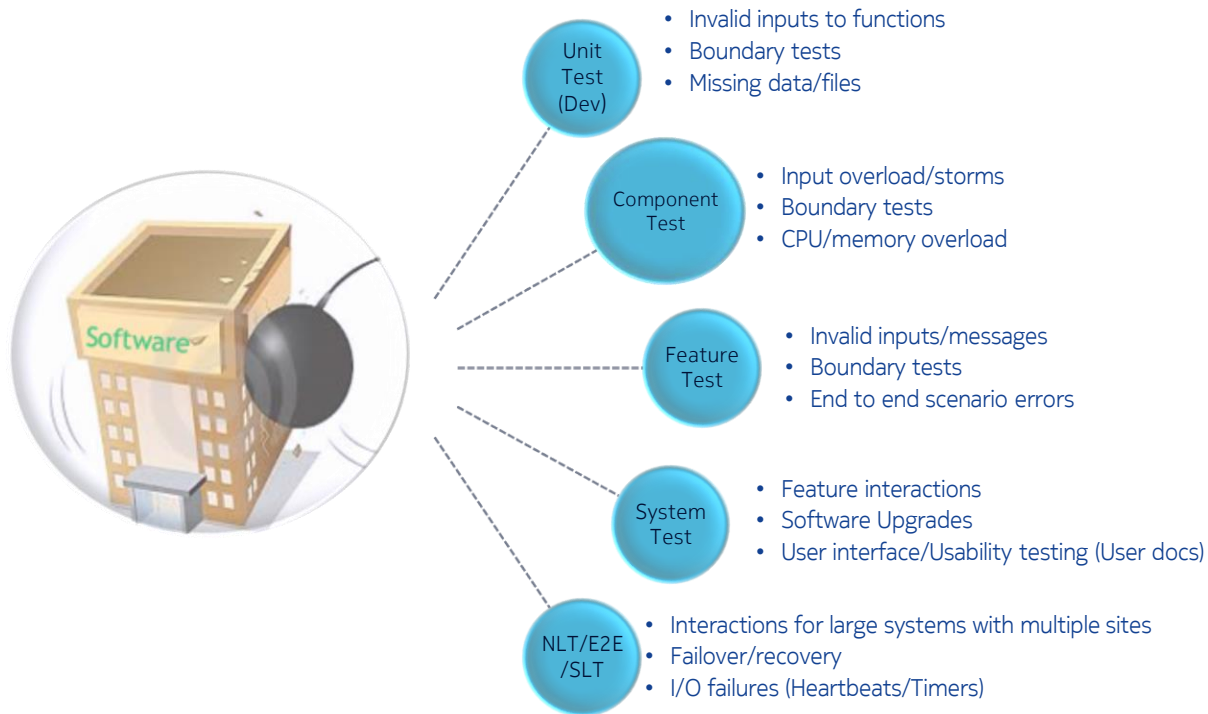
Communication and application inter-working (message errors, heartbeat failure, network congestion). Testing with multiple sites.

Mixed and varying traffic load, soak, growth and de-growth, upgrades, routine maintenance actions

Testing of user procedures to identify robustness gaps

Software robustness is tested as part of each individual test area  
Robustness testing by phase ...

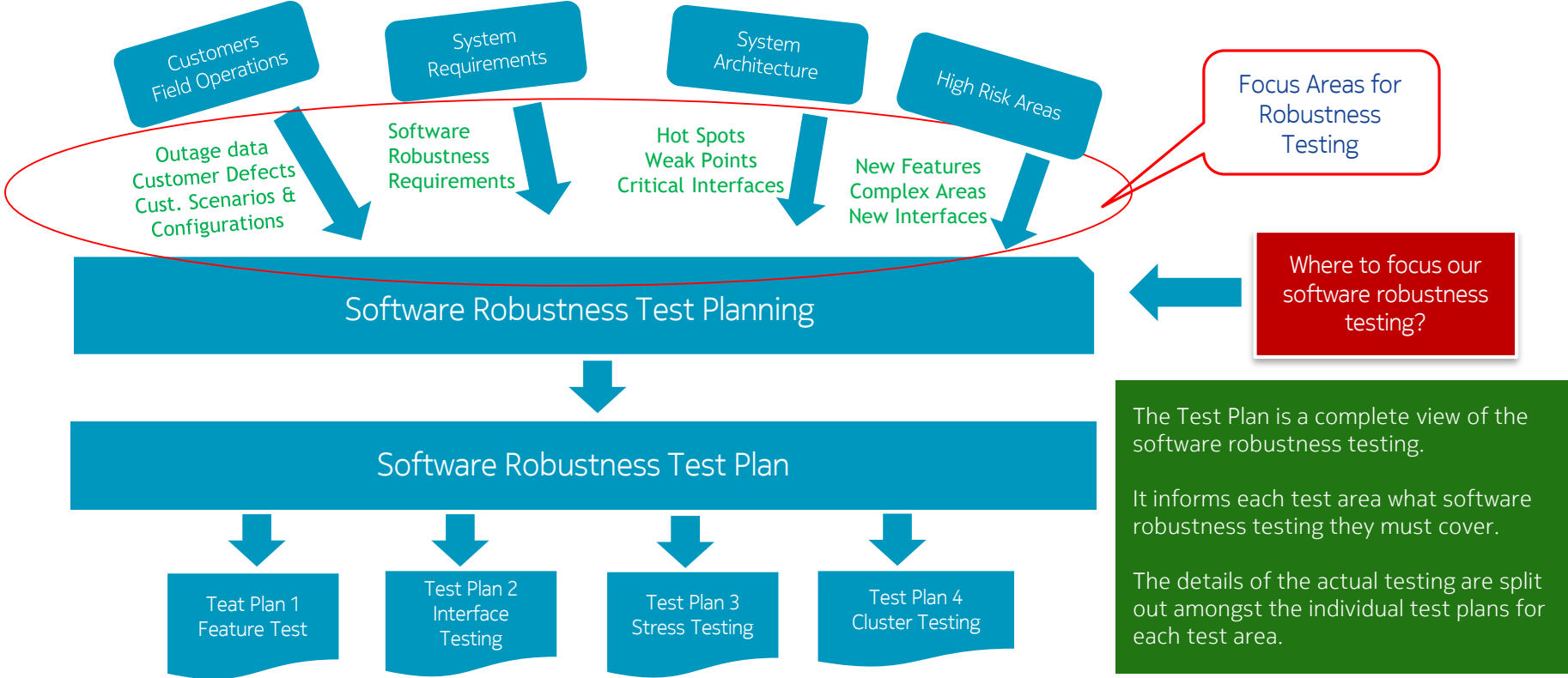
# Who Performs Robustness Testing?



Let's look at how to build a complete software robustness test plan

# 4. How to Build a Software Robustness Test Plan, including Fault Modelling

# Building a Complete Software Robustness Test Plan



This requires one person to co-ordinate software robustness testing across the different teams

# What Should Be Available Before Test Planning?

FUNCTION	SOFTWARE ROBUSTNESS ACTIVITIES	OUTPUTS AND RELATIONSHIP TO TESTING
SYSTEMS ENGINEERING	<p>Analysis of system interactions to identify what could go wrong.</p> <p>Analysis of customer issues to identify robustness issues.</p> <p>Develop requirements to prevent these failures.</p>	<p>List of system level software robustness requirements to address during development (high level).</p> <p>Customer scenarios, failure modes, error cases, corner cases, stress scenarios</p> <p>Informs testers what needs to be tested.</p>
ARCHITECTS	<p>Architect the software to meet the system level requirements.</p> <p>Analysis of the product architecture to identify potential robustness faults within the architecture, especially interface issues and resource management.</p> <p>Develop requirements to prevent these faults.</p>	<p>List of high risk components/features to focus testing (high impact on failure, complex components, problematic components).</p> <p>Interface documents/descriptions.</p> <p>List of hot spots and weak points.</p>
DESIGNERS/ CODERS	<p>Defensive programming and error checking:</p> <ul style="list-style-type: none"><li>* design of rainy-day cases, query failures, network errors</li><li>* design how to handle arguments out of range, null pointers, memory allocation errors, etc.</li></ul>	<p>Description of robustness elements of design (typically at feature level)</p>

- Testers ask for:
1. List of system level software robustness requirements
  2. List of customer scenarios to support, especially failure modes, error cases, corner cases
  3. List of high risk components/features
  4. Interface documents/descriptions
  5. List of your product's hot spots and weak points
  6. Description of data structures, especially shared data

# INPUT 1. Examples of Escaped Defects

Example	Description
1	We have lost connection from our xxx with all other nodes of the network.
2	Hardware failure. A disk controller lockup on Blade YY.
3	Provisioning issue due to defect trigger by mated pair switchover
4	XX overflow due to resource usage
5	4 XX in overload control following IP instability. A lot of latency in the connection between front end and back end
6	QoS problems (packet loss) resulting in repeatedly performing switchovers between the xxx sites. Each switchover caused work-orders in process to fail and require resubmission, and caused a backlog of work orders to be generated

Data gives insight to what failures occur in the field  
Lost connections, hardware fails, switchover, overflow, IP instability, packet loss..

## INPUT 2. Examples of Escaped Defects

Example	Description
1	switchover triggered xxx engine initialization to fail, resulting in yyy content stuck in queue.
2	Xxx and yyy initialization at the same time leaves xxx in bad state.
3	Xxx blades do not recover automatically after bond0 fails.
4	Xxx relocated sessions in pool. Original xxx retained sessions as sessions did not get cleaned up from original xxx.
5	We need a new upgrade procedure (for HA) which xxxxxxxx
6	Size of server.log on xxx increases dangerously when Repair Request outside Repair Window. Risk > disk full
7	userplane : Alarm "Bearer Input Missing; {Invalid input}"
8	not enough space available under xxxxxx Not enough hard disk space available.

Lost connections, hardware fails, switchover, overflow, IP instability, packet loss..



# Outages

- Questions to ask by analysing the outage data?
  - Which software components/sub-systems are most prone to fault?
  - Which software failure mechanisms are most common?
    - Timing/Latency Issues, Heartbeat failures, Invalid Input Data, Invalid Messages, Network Instability...

The answers to these questions gives insight to:

- Which sub-systems/components/features to focus on for robustness testing
- What type of testing to perform

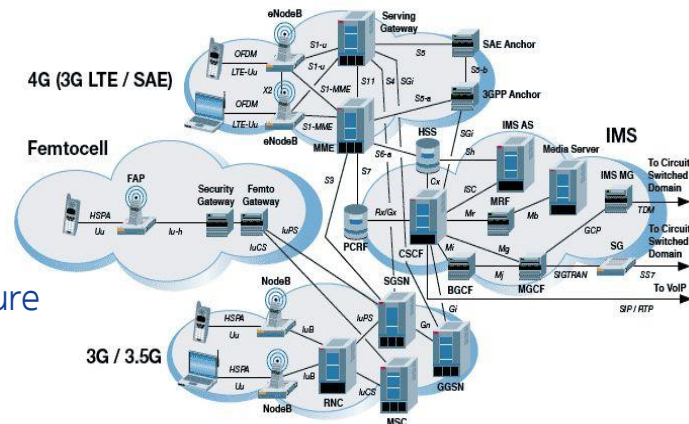
Testers must ask :

- What has already gone wrong? What other similar things could go wrong?
- How can I trigger such errors and test for detection, isolation, recovery?
- Each critical and realistic error needs a robustness test case

## 2. System Requirements

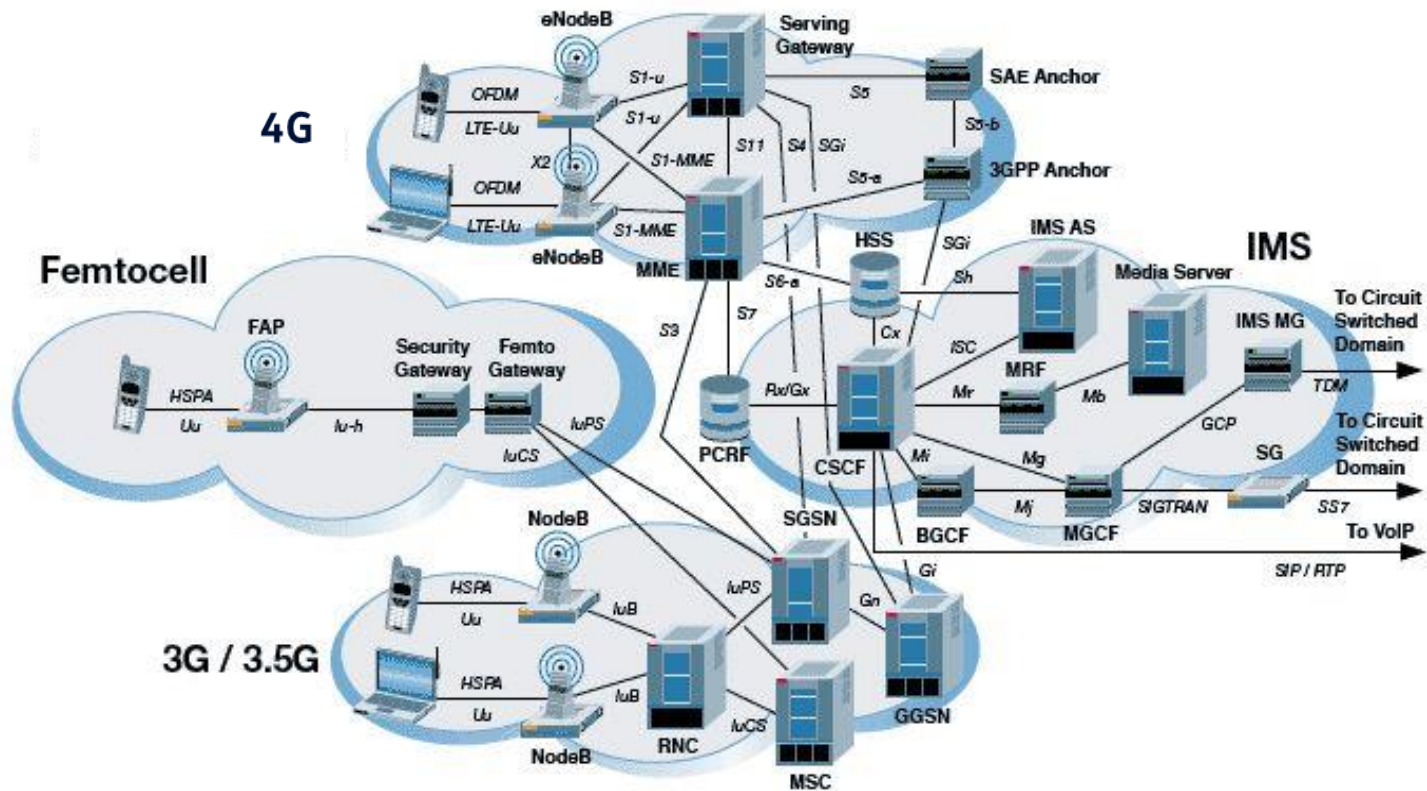
### Types of question to ask to your systems engineers and architects

- What are the inputs/outputs of the system ?
- What are the known end to end to end robustness requirements, failure modes, error cases, corner cases
- Are all the inputs specified, including their source, accuracy, range of values and frequency?
- Are all the outputs from the system specified, including their destination, accuracy, range of values, frequency, and format?
- Are all the external communication interfaces specified, including handshaking, error-checking, and communication protocols? (SGmb, SNMP, Diameter, M1, M3...)
- Is the data used in each task and the data resulting from each task specified?
- What external stressful conditions could arise that would stress the software
  - Hardware failures, Network failures, Timing issues, Data flood/overload



The answers to these questions gives insight to the system level interactions

# 3. Software Architecture

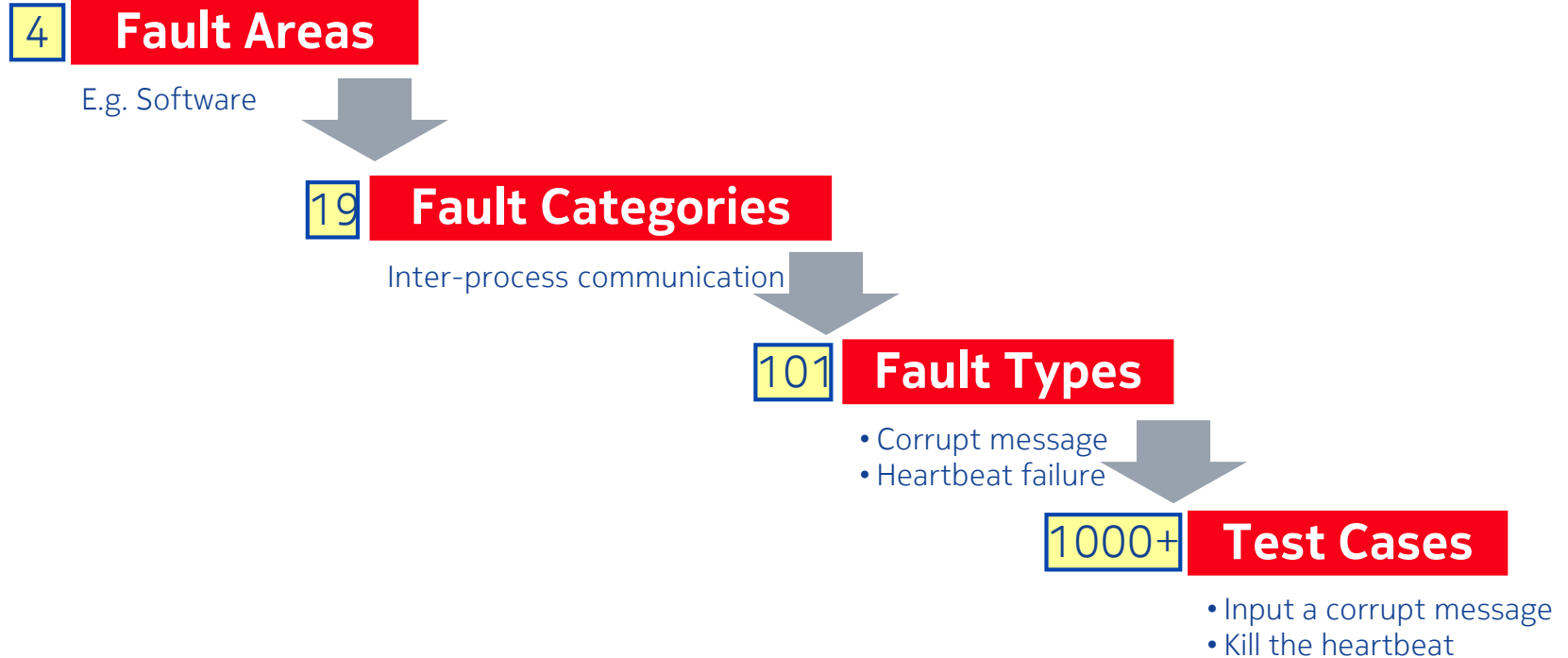


# Questions For Systems Engineers & Architects?

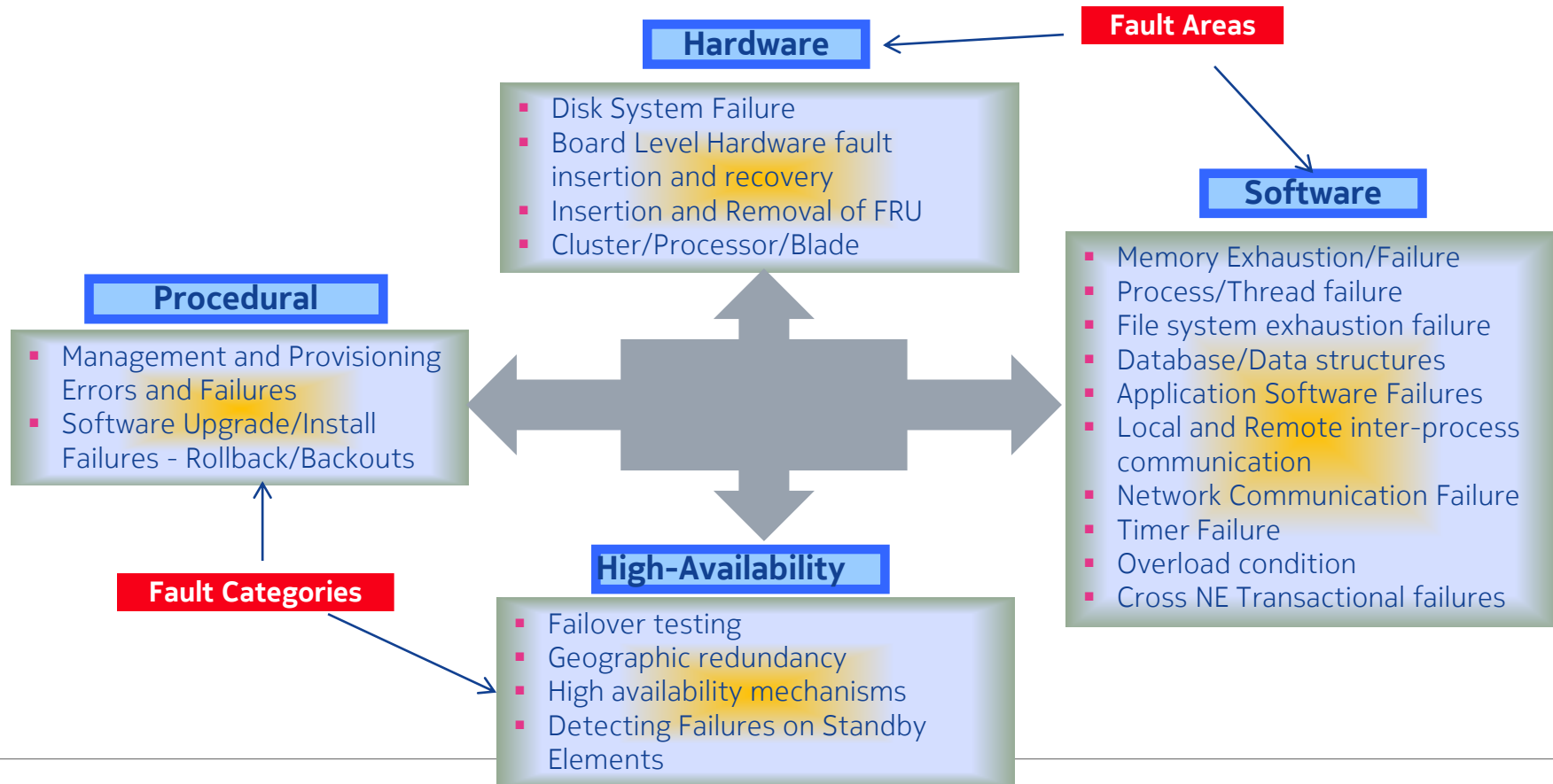
- What are the hot spot components/features with high activity?
- What are the weak spots with critical points of failure/single points of failure?
  - What are the critical interfaces internal to your product?
- Testers need good insight into the internal working of the software to build software robustness test cases
- Looking at the hot spots and weak points, how do we identify what could go wrong
  - Fault modelling and architecture exploring

Let's look at fault modeling

# Fault Taxonomies



# Fault Models (identifying potential failures)



# Developing Your Own Fault Categories

## Exploring Architecture Diagrams to identify fault categories

### If you can develop a Fault Model :

- You can test the model as you develop it
- You can draw implications from the model



### Examples of models for failure mode testing:

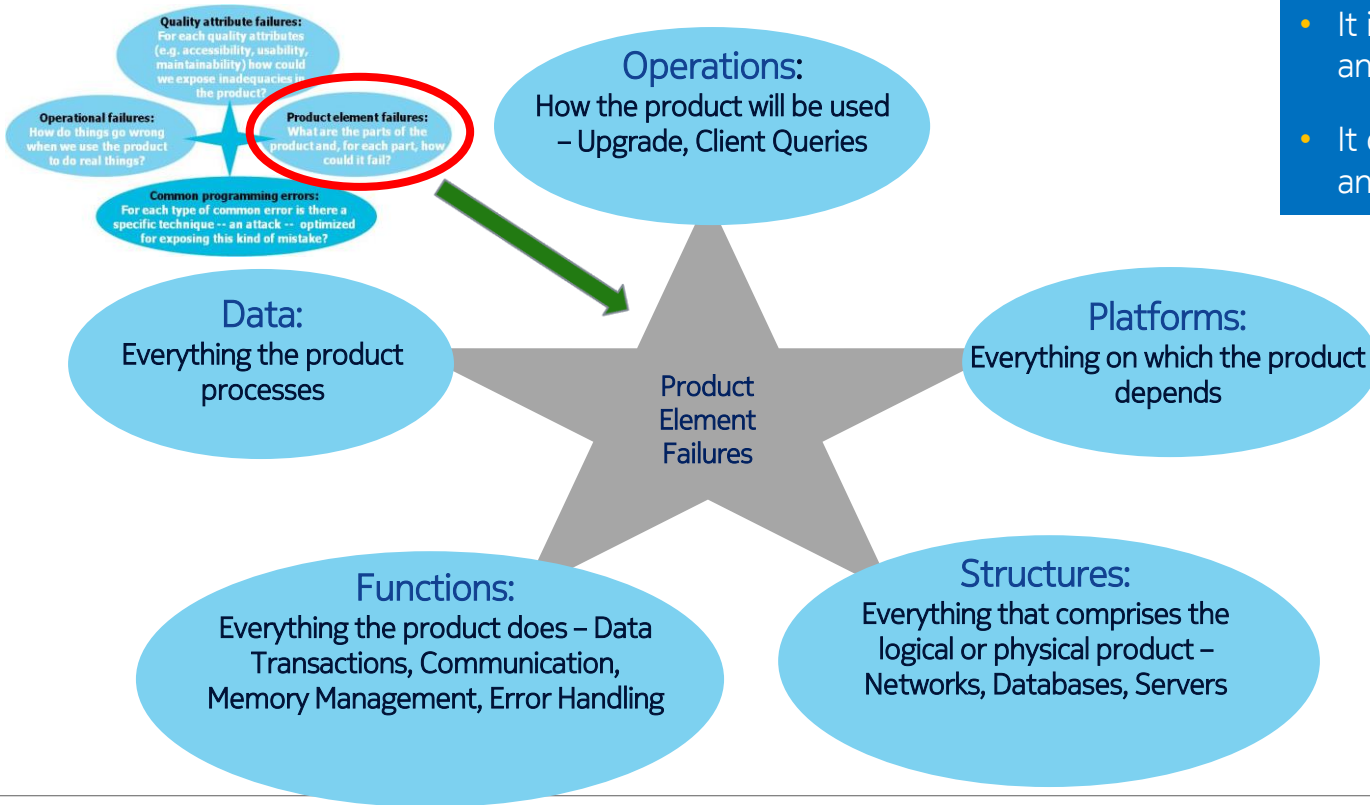
- architecture diagram
- state-based diagram
- dataflow

1. Work from a high level design (map) of the system
2. Pay primary attention to interfaces between components or groups of components. We're looking for cracks that things might have slipped through
3. What can we do to mess/break things up as we trace the flow of data or the progress of a task through the system?
4. Build the map in an architectural walkthrough
5. Invite several programmers and testers to a meeting. Present the programmers with use cases and have them draw a diagram showing the main components and the communication among them.
6. Ask "What if" questions to understand failure events and recovery responses. Initially, the diagram will change significantly with each example. After a few hours, it will stabilize.
7. Take a picture of the diagram, blow it up. Use markers to sketch your current focus. Share across different sites.
8. Plan tests from this diagram jointly with other testers who understand different parts of the system

Testers cannot do this without insight from systems engineers, architects and designers

# Developing Your Own Fault Categories/Fault Type

Imagine how the product can fail



- A software product is much more than code
- It involves a purpose, platform and user.
- It consists of hardware, software and user documentation



## 4: High Risk Areas

- The last input to the software robustness test plan is the areas that are high risk.
- High risk areas are typically:
  - new features
  - new components
  - new interfaces
  - components/features with high impact upon failure
  - very complex features or components
  - problematic features or components (high defects during design/development)
  - network configurations working at the extremes of their capability

We have seen how to identify what components/features/interfaces to focus on

We have seen how to identify fault categories and fault types to focus on

Together, these help us understand what to test and how to test

How do we build test cases for these fault categories/fault types ?

# EXAMPLES OF FAULT INJECTION (1)

Exploring the input domain	1	Input values that are out of bounds/range. Are they rejected?
	2	Apply inputs that force error messages to occur. Are error messages displayed
	3	Apply wrong inputs, that force the software to establish default values
	4	Explore range of allowable character sets and data types. Are disallowed characters rejected
	5	Overflow input buffers, too many characters for example. What happens to the data?
	6	Input parameter values that will cause a failure. Are they rejected?
	7	Repeat the same input or series of inputs numerous times.
Exploring stored data	8	Apply inputs using a variety of initial conditions
	9	Force a data structure to store too many or too few values
	10	Test with needed data/files missing
Exploring feature interaction	11	Pass invalid parameter values or invalid parameter types between features
	12	Send an invalid IP address to a feature or component.
	13	Verify features that share data. What happens when the data is corrupted?

Some examples

# EXAMPLES OF FAULT INJECTION (2)

System interface attacks	1	Try disabling a critical interface
	2	Lock an element then try communicating with it
	3	Reset an element and try communicating with it while it is resetting
	4	Introduce delays or latency on the interface.
	5	Introduce packet loss on the interface
	6	Insert malformed message on communication link
	7	Generate out of order messages
Generate interrupts, unsolicited message, series of interrupts	8	Interrupt from a device related to the task (e.g. network port)
	9	Interrupt from a device unrelated to the task (e.g. timer)
	10	Interrupt from a software event (e.g. set another program's (or this program's) timer to go off during the task under test)
Change something that this task depends on	11	Swap out a disk; data file
	12	Change the contents of a file that this program is reading
	13	Change the port that the program will write to (without signaling the change)
	14	Change the packet/link rate

Some examples

# EXAMPLES OF FAULT INJECTION (3)

Some examples

Cancel/abort/kill	1	Cancel the task (at different points during its completion)
	2	Cancel some other task while this task is running
	3	Cancel a task in communication with this task (the core task being examined)
	4	Cancel a task that will eventually have to complete as a prerequisite to completion of this task
	5	Cancel a task totally unrelated to this task
Pause, Hang	6	Create a temporary interruption in the task
	7	Pause the task/thread for a short time
	8	Pause the task/thread for a long time (long enough for a timeout, if one will arise)
System and Security	9	Try logging in several times with the same user name and password
	10	Try changing the time zone. Are all of the time stamps correct?
	11	Try locking the user out of the network element (which would require a reboot)
	12	Try inputting password with wrong case (upper case/lower case). Does it still work?
Lock	13	Lock a resource that is needed for the task, for example a back-up blade. Is an alarm raised?
	14	Put a database under use by a competing program, lock a record so that it can't be accessed.

# EXAMPLES OF FAULT INJECTION (4)

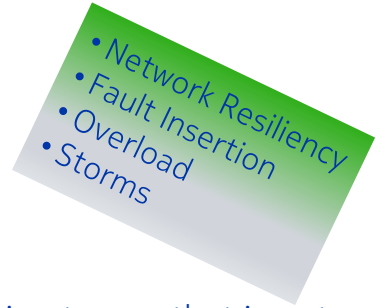
Traffic Tests	1	Send traffic to a node that cannot process traffic (locked, not configured)
	2	Send traffic to an invalid IP address.
	3	Force a switchover of traffic destination address
Access to resources	4	Have multiple processes generate interrupts (e.g. time-alarm)
	5	Fill the file system/hard disk to its capacity. Is there an alarm?
	6	Assign an invalid file name
	7	Force the hard disk or flash memory to be busy or unavailable
	8	Turn on logging and tracing to maximum extent
Upgrade	9	Start upgrade and then roll back in the middle of the upgrade
	10	Upgrade with data or file needed for the upgrade missing
	11	Abort upgrade
	12	Start upgrade. Do all management interfaces show the correct upgrade status. Rollback the upgrade. Do all interfaces have the correct status.

Some examples

Consider test cases which have single, double, multiple and sequential failures  
These can uncover the less common failure modes

# Examples 1: Stress Test a Telecommunications Network

- Push the system past its capacity limit to trigger overload
  - Push the system to its limit, then change the configuration to reduce available capacity
- Changing configurations
  - Set the system up with configuration A and then set the system running. Modify configuration to one that is not allowed
- Try something “not clever”. For example, try to shut down a live node with thousands of live calls/data sessions
- Perform bulk updates in OA&M while pushing traffic to its maximum
- Overload the system with End User Equipment
- Set the date/time to be incorrect or out of sync (should cause messages to be rejected)
- Block or delay heartbeats between nodes (simulates I/O congestion)
- Block a resource, for example block a port or remove a needed file
- Failover in an N+1 system to a dead system
- Break links between nodes at random times (simulates real life radio connections going down, fibre cuts and power outages)



In general, teams need to consider what are the most likely areas for software robustness defects

## Examples 2: Stress Test a Telecommunications Network

Test with max capacity traffic flowing through the user plane.

- Change traffic profiles - various types of service (data, web browsing, VoIP, handovers, add/drop users)
- Overload. Try to push a higher number of connections into the channel than it can handle.
- Overload the signalling/control plane
- Push high throughput with varying packet sizes (large, small, corrupt, packet loss)
- Perform stressing OA&M operations during high traffic load e.g. garbage collection, download back-up data, reconfigure the parameter set.

## Test Cases – Questions To Consider

- What areas of the software to focus on for robustness testing ?
- What are the realistic faults ?
- What types of fault to insert - invalid input, corrupt messages, stressful environment ?
- Where to inject (external or internal interface) ?
- Characteristic of fault injected (transient, intermittent or permanent) ?
- How can we stress the software in the background, for example to stress CPU, memory, I/O channels ?
- Are we testing the critical areas – what is the coverage of:
  - Areas of focus identified in test planning phase
  - Error handling mechanisms
  - Critical interfaces

Lastly, let us look at how we can use the stability testing phase to test for software robustness

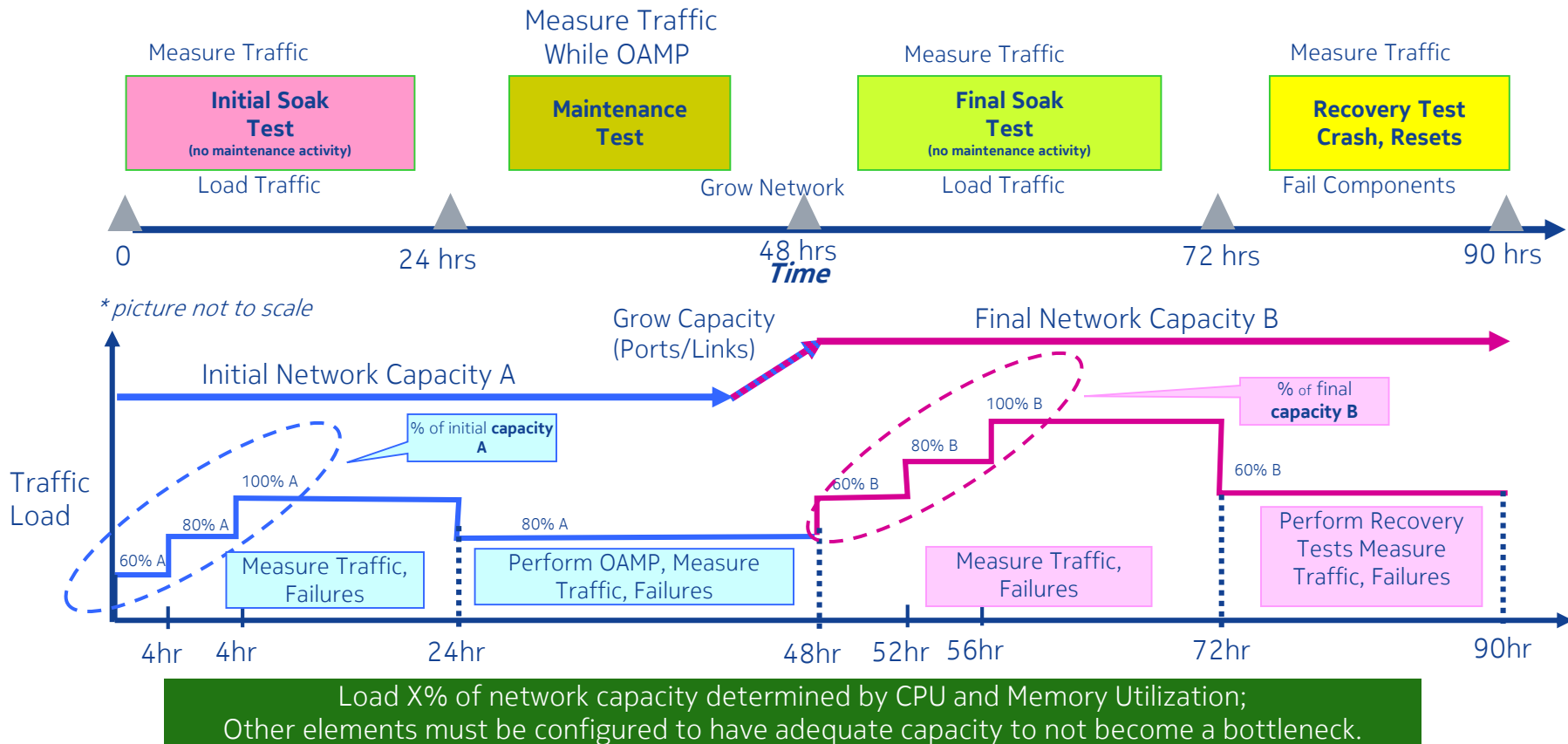


# Steps For The Stability Run

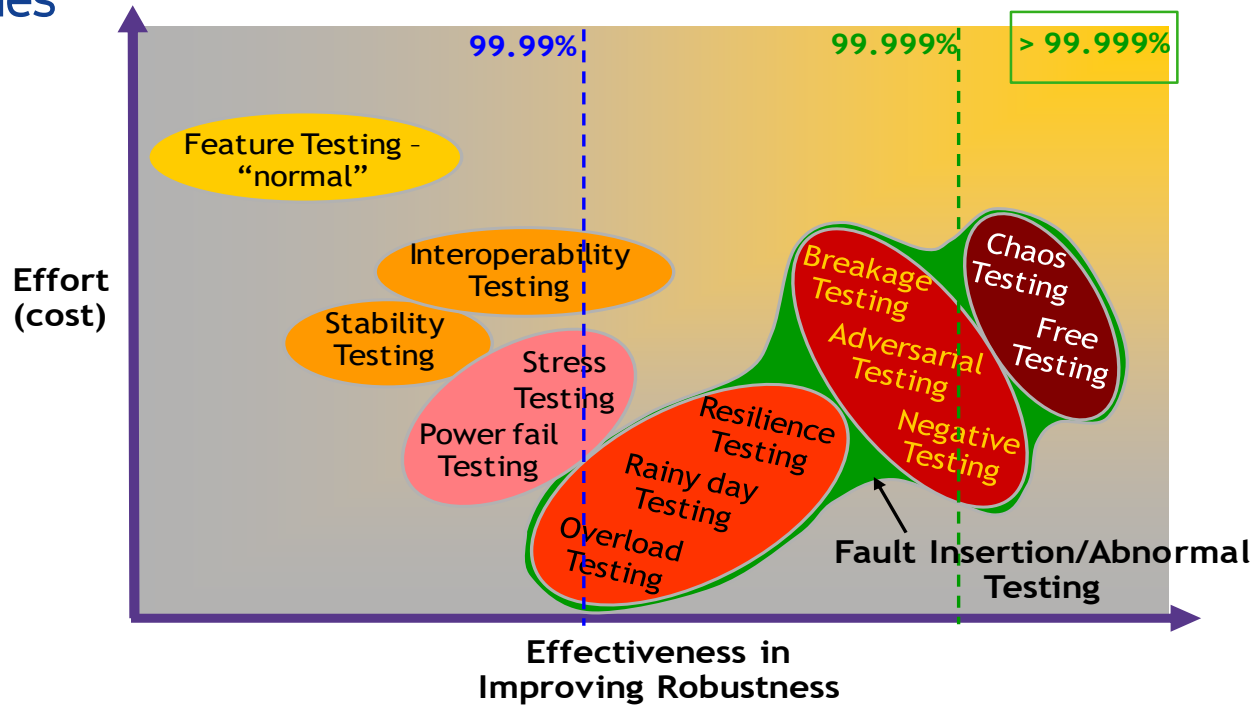
## The long run stability test is an excellent opportunity to perform robustness testing

- **Initial soak cycle:** Start with Initial soak configuration. Increase traffic during soak; Start with 60% load increasing to 80% moving up to 100% (mixed traffic). Monitor Memory & CPU usage, monitor database integrity on key elements, monitoring the number of failures.
- **Maintenance Test:** Run maintenance tests and during the maintenance interval, grow the network to increase its traffic carrying capacity, e.g. add another interface card, add another switch etc. Monitoring the number of failures. Maintenance activities performed with mixed traffic.
- **Final soak cycle:** Increase the load to 60%, to 80% moving up to 100% (mixed traffic) of the new capacity. Monitor the number of failures over the cycle (w/ mixed traffic).
- **Perform maintenance/recovery tests:** on a loaded network - monitor failures of key elements and recovery

# Stability Run Details



# Robustness Testing Effort-Effectiveness Guidelines



Reaching Five 9's requires robustness testing, including software fault injection  
What about the non software parts of the system ?

# What Is Procedural Robustness?

- Procedural robustness refers to the aspect of robustness associated with the human aspect of the system, including the operating personnel, documentation & training
- Procedural errors typically happen during:
  - Normal OA&M type operations
  - Configuration activities
  - Installation and upgrade activities
  - Abnormal conditions – something is different/something is done under time pressure (human dimension)
- Software robustness to procedural errors
  - Does the software manage invalid inputs or stressful environments during these procedural activities
  - Does the software alert the user when they want to shut down a node with live traffic
- Problems related to procedural errors
  - Reluctance of organizations and individuals to expose human errors
  - Human error is highly situation dependent
  - Field issue systems report what happened, not why it happened. Hence, development get poor feedback from the field

Improving procedural robustness will improve the overall system robustness

# Types of Procedural Error

- While applying the procedure
  - Applying the procedure to the wrong network element (a live element instead of a back-up element)
  - Skipping a step in the procedure
  - Using wrong version of procedure
  - Not performing the pre-checks
  - Not performing the post checks
- Errors involving process;
  - Performing hardware change in middle of soak of software retrofit
  - Performing software change in the middle of multi-day hardware change

# Challenges for Testers

Aspect	Challenge	Solution
Learning	How do we get to know the program?	Testers need input from systems engineers, architects and designers
Visibility	How to see below the surface to see what is inside the program?	Testers need input from systems engineers, architects and designers
Control	How to set internal data values?	Testers need input from architects and designers
Risk / selection	Which are the right tests to run? What are the priorities?	Testers need input from previous defects, architects and designers
Execution	What's the most efficient way to run the tests?	Distributed across different phases of testing

It is not just black box or sunny day scenario testing  
Testers have to get to know the product (from systems engineers, architects and developers)

# Ever changing software development practices

## Working Agile

- Agile teams work in short sprints
- Focus is on creating small chunks of working features
- Not always sufficient focus on software reliability during these short sprints
- Some Agile techniques can help:
  - Continuous integration
  - High levels of automated tested
  - Automated static analysis on check-in of code
  - Maintaining only one main code branch
- Types of testing not done in sprints:
  - Network level verification
    - Challenges of real equipment versus simulators
  - Performance testing (capacity, throughput, latency, response times)
  - Interoperability testing (network elements and different vendors of the same network element)

Teams must consider what are the most appropriate phases to detect software robustness defects

# Third Party and Open Source Software

## Testing challenges

- How do you test multiple configurations in a short timeframe
- How to test areas such as installation, upgrade and migration in a short timeframe
- To assure sufficient coverage, teams must perform software robustness testing at every stage of testing.
- Telecommunications network comprise internally developed software. However, they also comprise elements of:
  - External software from third parties
  - Open source software
- Examples include:
  - Operating Systems
  - Protocol Stacks
  - Database Management Software
  - Firmware in devices such as Remote Radio heads(RRH)

In general, teams need to consider what are the most likely areas for software robustness defects



# Summary

## Gather Input

- Customer Complaint data
- Outage data
- Customer ticket data
- Customer Scenarios & Configurations
- Software robustness requirements
- Architectural input - Hot Spots and Weak Points
- High Risk Areas

## Analyse

- What to test – Which types of test to run
- Identify which sub-systems, components, features, interfaces should be the focus of robustness testing
- Identify which types of software robustness tests are most important

## Plan

- Determine what types of fault categories and fault types are relevant
- Determine which test areas are appropriate for each Fault Category/Fault Type
- Assign Fault Categories/Fault Types to the appropriate test area

## Test Cases

- Each test area develops the test cases needed to cover the testing assigned to them

# 4. Conclusion

# Topics

1. **Reliability in Telecommunications Networks**
    - Overview of Nokia
    - Current & Future Telecommunications Networks
    - Reliability Requirements in Telecommunications Networks
  2. **Software Robustness**
    - Definition
    - Examples of Software Robustness Defects
    - Origin of Software Robustness Defects
  3. **Software Robustness Testing**
    - Role of Software Robustness Tester
    - Software Robustness Testing as a Distributed Activity
  4. **Building a Software Robustness Test Plan, including Fault Modelling**
    - Inputs to the software robustness test plan
    - Software Robustness Test Case Examples
    - Extending Typical Stability Run
    - Procedural Reliability
    - Challenges for testers
  5. **Conclusions**
-

**NOKIA**