SENG 275

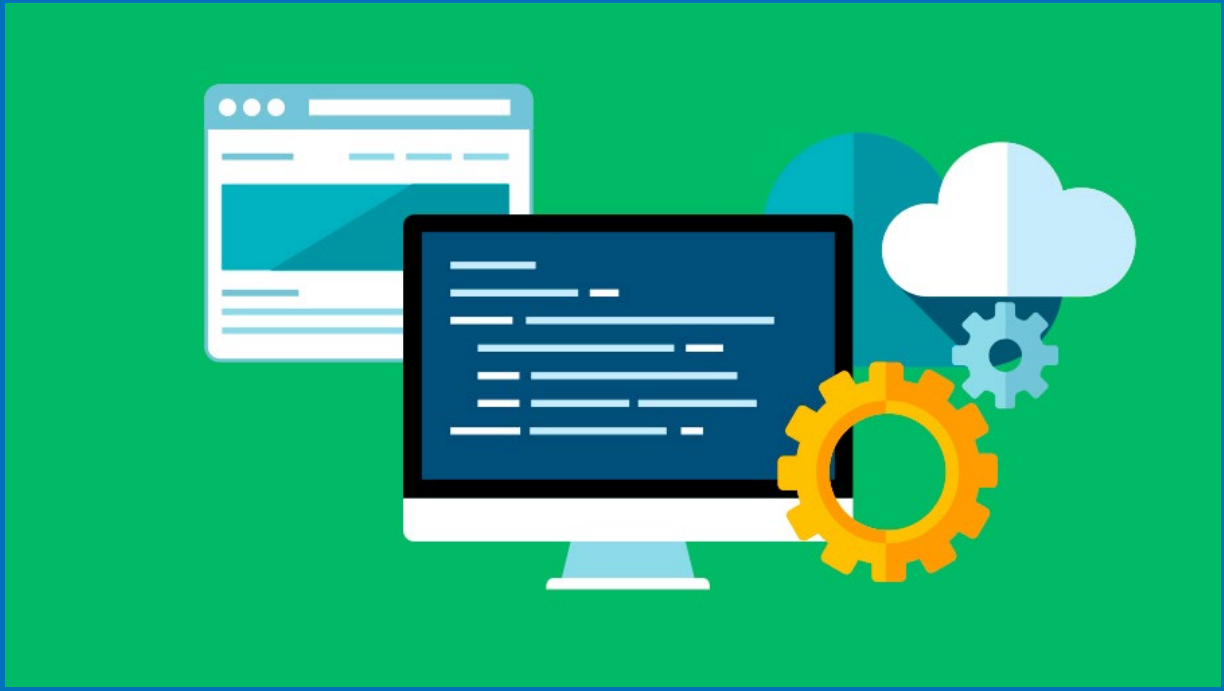
SOFTWARE TESTING

DR. NAVNEET KAUR POPLI

DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING

University of Victoria

NEED OF TESTING

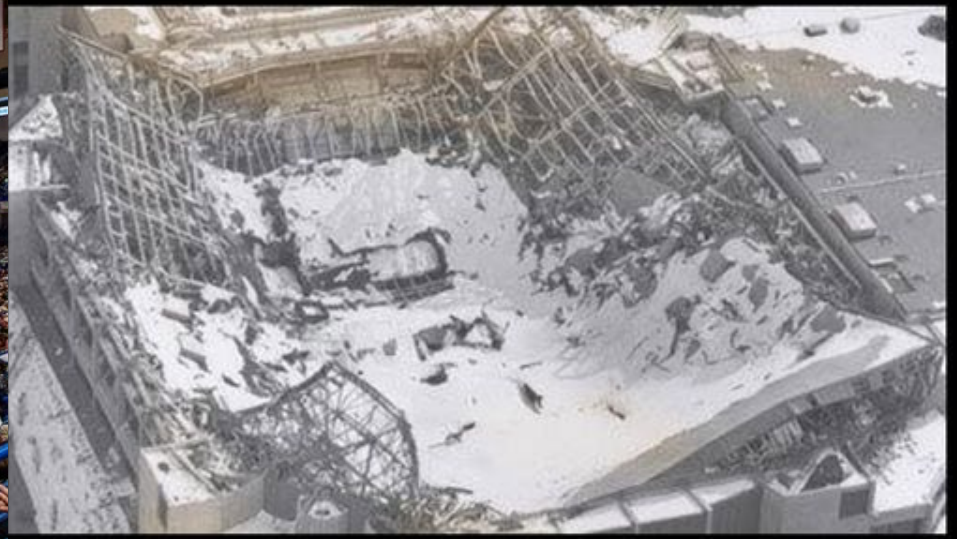


University of Victoria

TOPICS

- Disasters due to lack of testing
- Definition of testing
- Importance of testing
- Difference in developer and tester perspective

Hartford Civic Center Roof Collapse in 1978



The first computer-aided failure

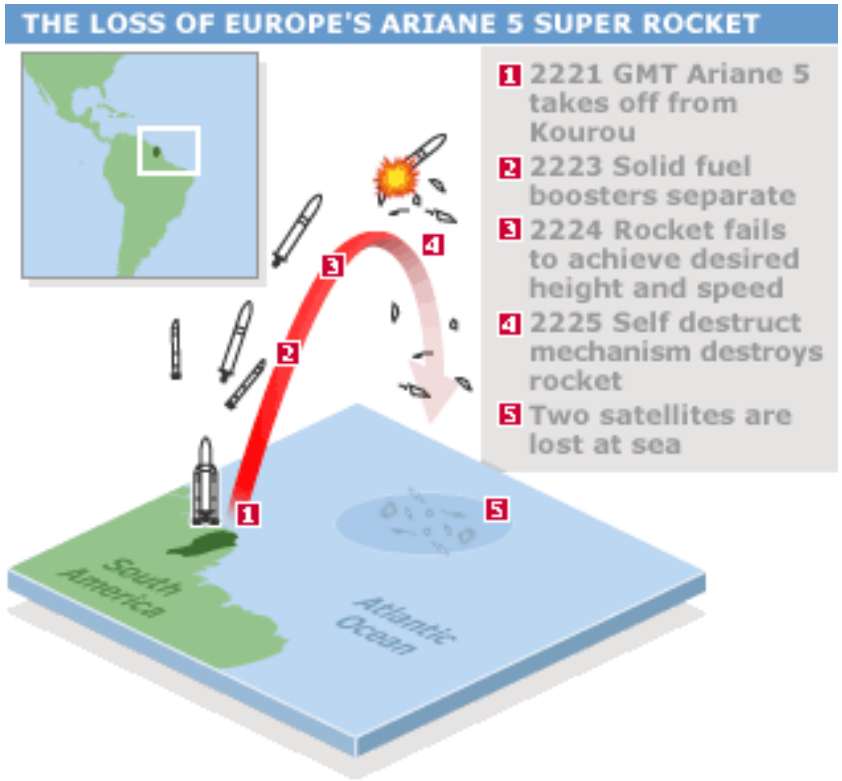
- The Hartford Arena was constructed in 1973 and housed a basketball court and seating for 5,000 spectators.
- On January 18, 1978, during a heavy snowfall, the huge roof suddenly collapsed, only hours after a well-attended game.
- Engineers traced the collapse to an “oversimplified computer analysis.”
- This failure is known as the first computer-aided failure.

Over dependence on computer analysis

- The engineers for the Hartford Arena depended on computer analysis to assess the safety of their design. Computers, however, are only as good as their programmer.
- The roof design was **extremely susceptible to buckling** which was a mode of failure not considered in that particular computer analysis and, therefore, left undiscovered.
- Any engineer could easily have discovered this error, at the earliest stages of the project, by comparing the **stress calculated** by the computer against the well-known **Euler buckling equation**—a simple hand calculation that can be performed in minutes.

Software disasters due to lack of testing

- ARIANE 5 Failure: On 4th June 1995, Europe's newest satellite-launching rocket, ARIANE 5 ended in failure. Just after 40 seconds of it taking off, the launcher broke down and exploded.



University of Victoria

Type casting and Overflow error (reuse old components wisely!):

- A software bug in the rocket's **Inertial Reference System (IRS)**. The rocket used this system to determine whether it was pointing **up or down**, which is formally known as the **horizontal bias**, or informally as a **BH** value. This value was represented by a **64-bit floating variable**, which was perfectly adequate.
- However, problems began to occur when the software attempted to stuff this 64-bit variable, which can represent billions of potential values, into a **16-bit integer**, which can only represent **65,535** potential values.
- For the first few seconds of flight, the rocket's acceleration($a=\frac{\Delta v}{\Delta t}$)was low, so the conversion between these two values was successful.
- However, as the rocket's **velocity increased**, the 64-bit variable exceeded 65k, and became too large to fit in a 16-bit variable. It was at this point that the processor encountered an operand error and populated the **BH variable with a diagnostic value.**
- The error showed up because **Ariane 5 was able to create a higher horizontal bias than its predecessor Ariane 4** from whom the IRS component was re-used.
- **Error handling was suppressed** because of performance reasons.

University of Victoria

Do you have some stories of your own?

University of Victoria

At Least 56 Deaths from Sudden Acceleration in Toyota

Toyota deaths (Detroit Free Press) Off-duty California Highway Patrol Officer Mark Saylor, center, was driving through rush-hour traffic in suburban San Diego when his Lexus took off like a runaway rocket. With him were his wife, Cleofe Lastrella, left, their 13-year-old daughter, Mahala, right, and Cleofe's brother, Chris Lastrella, not pictured.

University of Victoria

Ref: http://www.human-resonance.org/acceleration.html

Toyota

- In the mid-2000's many Toyota drivers were reporting that their car was accelerating without them touching the pedal.
- After a series of accidents, which led to investigations, investigators discovered that **software errors** were the cause of the **unintended acceleration.**
- In this case, there was a series of things wrong with the software installed in Toyota cars- the **Engine Control Module (ECM) firmware** : **Memory corruption, wrong memory handling, disabling safety systems, systems with single points of failure, and thousands of global variables.**
- Toyota **recalled** millions of vehicles and Toyota's stock price decreased 20% a month after the cause of the problem was discovered.
- This case demonstrates the consequences of not giving enough attention to good programming practices and testing as a result of wanting to **launch the product in a hurry.**

University of Victoria

St. Mary's Mercy Hospital

- Imagine waking up one day, checking your mailbox an receiving a letter from your hospital saying you died.
- Well, that is precisely what happened to **8500** people who received treatment between Oct 25 and Dec 11 at St. Mary's Mercy Hospital. So, what happened?
- It turns out the hospital had recently upgraded its **patient-management software system.**
- However, a **mapping error** in the software resulted in the system assigning a code of 20 (which means "expired") instead of 01 which meant the patient had been discharged. But that is not all.
- The erroneous data was not only sent to the patients but also to **insurance companies and the local Social Security Office.**

University of Victoria

National Health Service

- I don't know what is worse: Not taking your medicines at all or taking the wrong medication.
- Either way, at least **300,000 heart patients** were given the **wrong drug or advise as a result of a software fault**. So, what happened?
- In the year **2016**, it was discovered that the clinical computer system **SystmOne** had an error that since **2009** had been **miscalculating patient's risk of heart attack.**
- As a result, many patients suffered **heart attacks or strokes** since they were told they were at **low-risk**, while other suffered from the **side-effects of taking unnecessary medication.**

University of Victoria

Therac-25- A safety-critical system

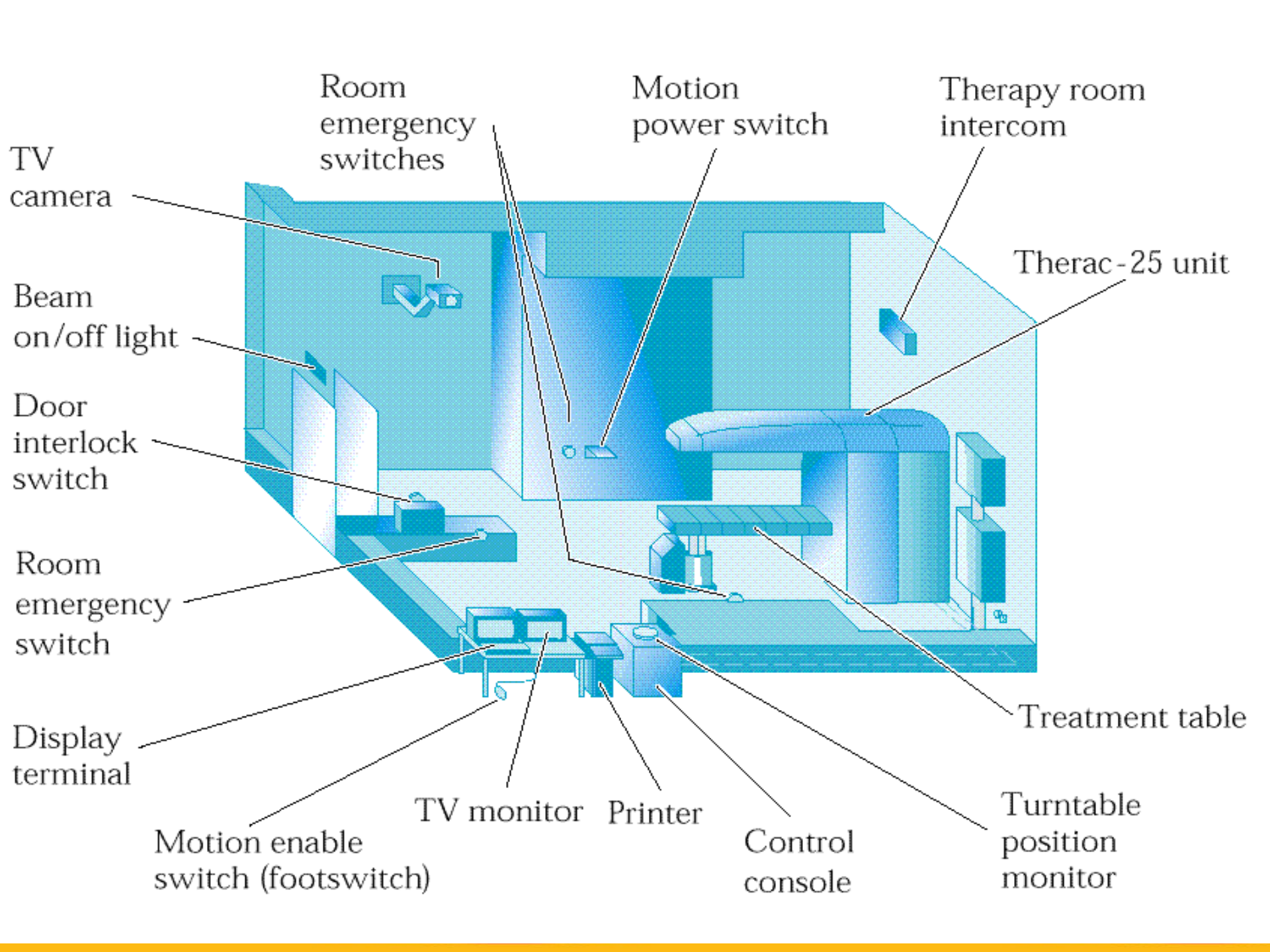
Radiation therapy:
Cancer is treated by damaging/killing the tissue with radiation.

Lineage:

- Therac-6
- Therac-20
- Therac-25 was first computer-controlled treatment machine.

Impact: 6 accidents, 4 deaths.
Normal dose:

- 80-180 rads to a small target area.
- For reference, 500 rads at once to the whole body is typically fatal.


Room emergency switches
Motion power switch
Therapy room intercom
TV camera
Therac-25 unit
Beam on/off light
Door interlock switch
Room emergency switch
Treatment table
Display terminal
TV monitor
Printer
Turntable position monitor
Motion enable switch (footswitch)
Control console

Therac-25 ‘The Killing Machine’

Reasons of failure

1. Race condition
2. Poor interface
3. Lack of hardware locks
4. Software reuse from old, faulty models which masked software defects by hardware interlocks.

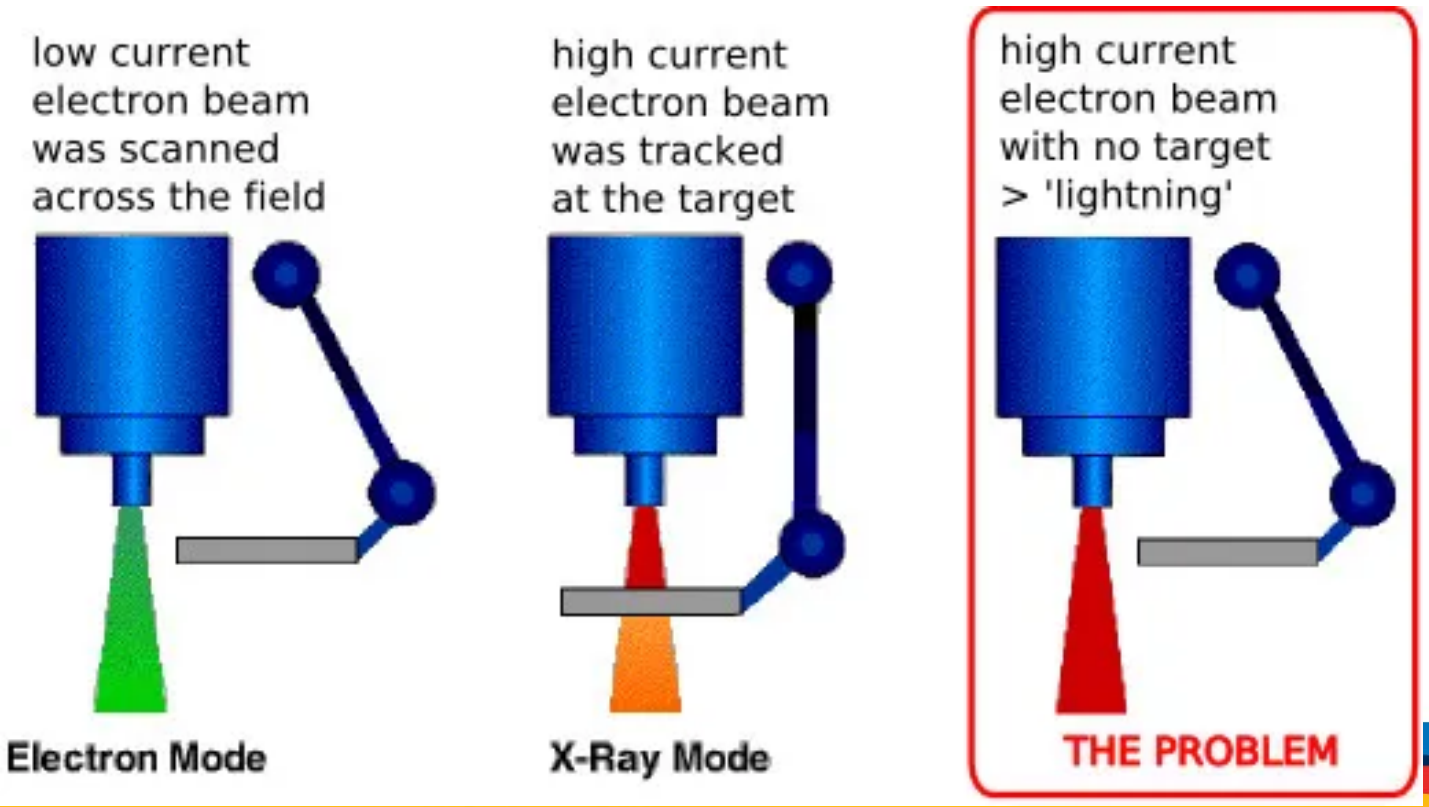
University of Victoria

Therac 25-working and problems

- The machine **moved magnets** between the two **modes** of the system: **x-ray** and **electron**.
- However, the mechanism gave **no feedback** to the computer to tell it that the magnets were in the correct mode.
- **So, if the operator typed in the wrong mode but then changed it, the operator assumed that the machine would go back to the right mode, but it didn't.**
- This created an in-between state(due to race condition), where the **magnets were in x-ray mode**, and the **turntable was in electron mode**.
- In this state, the Therac-25 could bombard the patient with **dosages 100 times greater than was intended.**

Two different treatment modes

- An **electron mode** which used an electron beam (beta radiation) to treat surface-level cancers.
- An **X-ray mode** which turned that same electron beam into X-rays by increasing the current and pointing it at an X-ray target. This could be used to treat deeper tumors.



University of Victoria

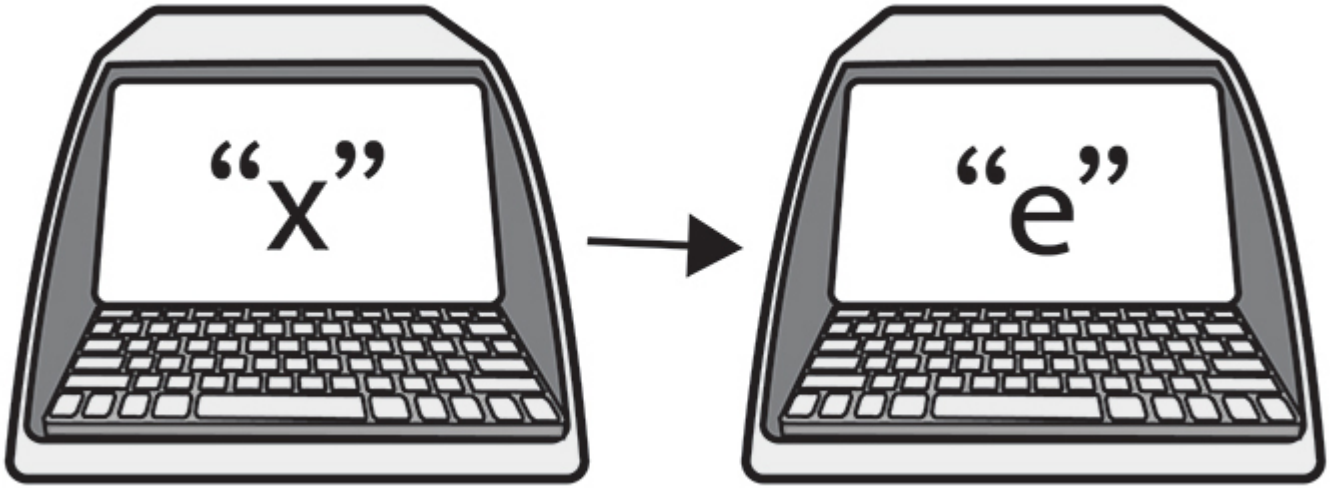
The story of Ray Cox

- On March 21st, 1986, Voyne Roy Cox had been receiving follow-up radiation therapy for a tumor surgery.
- He was to receive the therapy at the back of his left shoulder.
- The technician, Mary Beth, helped him take position on the table.
- He was supposed to receive a treatment of 22 MeV electron beams of 180 rads.

University of Victoria

‘x’ to ‘e’ with just an up arrow

- She entered the patient's prescription data with great efficiency but then realized she had made a mistake in entering the wrong mode, **x instead of e**. That means that she had entered for the x-ray mode instead of the electron mode he was supposed to receive. She had been administering most x-ray treatments so was accustomed to the typing errors. It was an easy to fix mistake, just an up key that would edit the mode entry. After verifying all parameters and correcting the error within eight seconds she began the treatment process.



University of Victoria

‘x’ to ‘e’. Really?

University of Victoria

Ref: https://www.dsource.in/course/human-machine-interaction-design/poorly-designed-interfaces-kill-people

Poor interface

- The machine also regularly gave **confusing error messages** (such as "Malfunction 42"), with **poor documentation** about the error.
- Operators became used to hitting the "Retry" button upon error -- but some errors reported that no radiation was administered even though it had been. Hitting "**retry**" hit the person with a **second dose of radiation**.
- The problem existed on previous models, but those models had **hardware interlocks** that prevented the machine from entering that **megadose** mode that wasn't supposed to exist.
- With the **hardware interlocks removed and total reliance on software,** the bugs in the program were allowed to cause serious harm.

University of Victoria

Operator Interface of Therac-25. Recreated from Leveson and Turner (1993).

PATIENT NAME : TEST
TREATMENT MODE : FIX

BEAM TYPE: X ENERGY (KeV):

A 1
25

	ACTUAL	PRESCRIBED	
UNIT RATE/MINUTE	0	200	
MONITOR UNITS	50 50	200	
TIME (MIN)	0.27	1.00	
GANTRY ROTATION (DEG)	0.0	0	VERIFIED
COLLIMATOR ROTATION (DEG)	359.2	359	VERIFIED
COLLIMATOR X (CM)	14.2	14.3	VERIFIED
COLLIMATOR Y (CM)	27.2	27.3	VERIFIED
WEDGE NUMBER	1	1	VERIFIED
ACCESSORY NUMBER	0	0	VERIFIED

DATE : 84-OCT-26
TIME : 12:55. 8
OPR ID: T25VO2-RO3

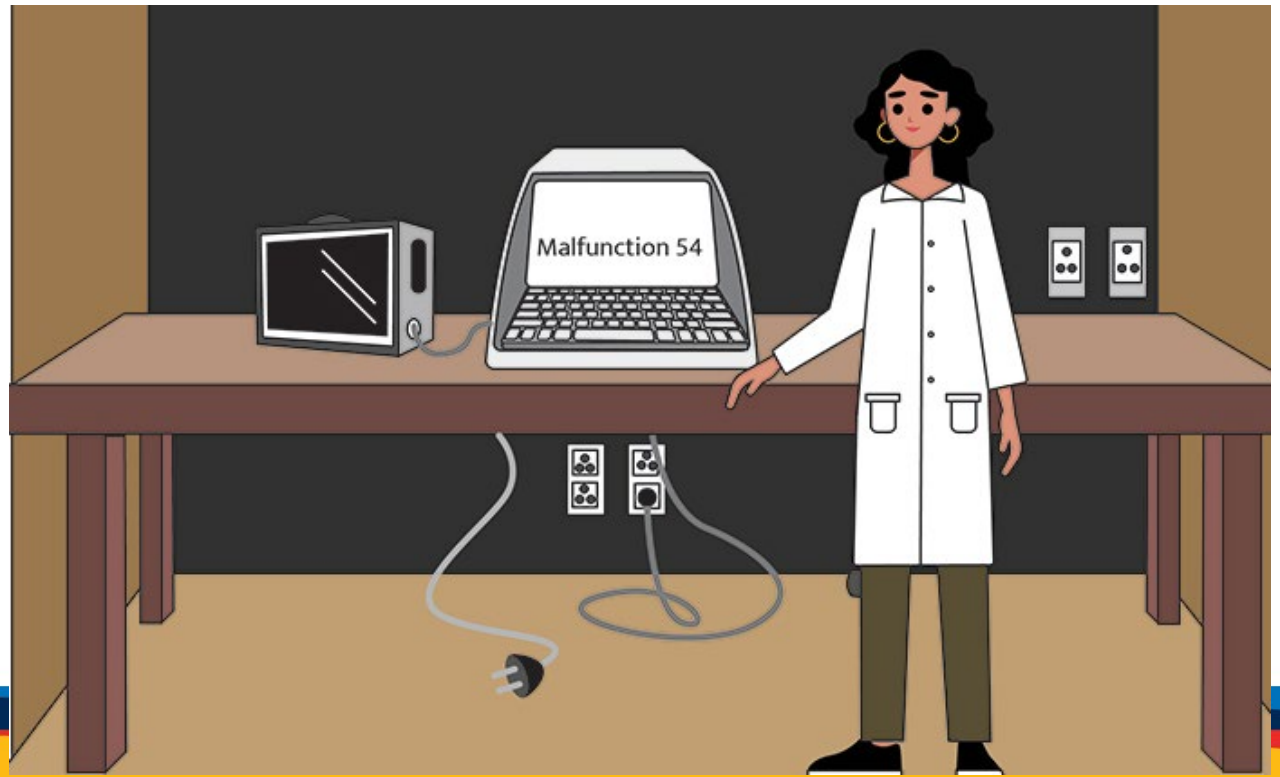
SYSTEM: BEAM READY
TREAT : TREAT PAUSE
REASON: OPERATOR

OP.MODE: TREAT
X-RAY
COMMAND:

AUTO
173777

Poor interface design in a safety critical system

- The system noticed that something was wrong and halted the X-ray beam, but merely displayed the word **"MALFUNCTION"** followed by a number from **1 to 64**.
- The user manual did not explain or even address the error codes, so the operator pressed the **P key to override the warning and proceed anyway**.



University of Victoria

Ray Cox- the 4th victim

- In actuality, the patient, Ray Cox, had received a massive overdose of radiation concentrated to the center of the treated area, which was estimated a possible dose of 16,500 to **25,000 rads** in less than 1 sec over an area of 1 cm.
- The patient experienced continued pain in his neck and shoulder area, later **lost the function of his left arm** and had periodic bouts of nausea and vomiting too.
- He was later hospitalized for radiation induced myelitis of the cervical cord causing paralysis to his left arm and both legs, left vocal cord and left diaphragm.
- He finally died of complications from the overdose five months later.

University of Victoria

TOPICS

- Disasters due to lack of testing
- Definition of testing
- Importance of testing
- Difference in developer and tester perspective

University of Victoria

Software testing

- *Software Testing* is the process of identifying the *correctness and quality* of software programs.
- The purpose is to check whether the software satisfies the specific requirements, needs, and expectations of the customer.
- Testing is also executing a system or application in order to find software *bugs, defects or errors*.
- The job of testing is to find out the reasons for application failures so that they can be corrected according to requirements.

In modern software development, we *release* often!

This means we need to **automate testing**

We need a **testing system** that

- **exercises** the system under test
- **verifies** the observed behaviour is correct
- tests "**good weather**" (conforms to user stories) and
  - "**bad weather**" (what happens when a user makes a mistake/exceptional and corner cases)

But we can't exhaustively test all feature interactions!

University of Victoria

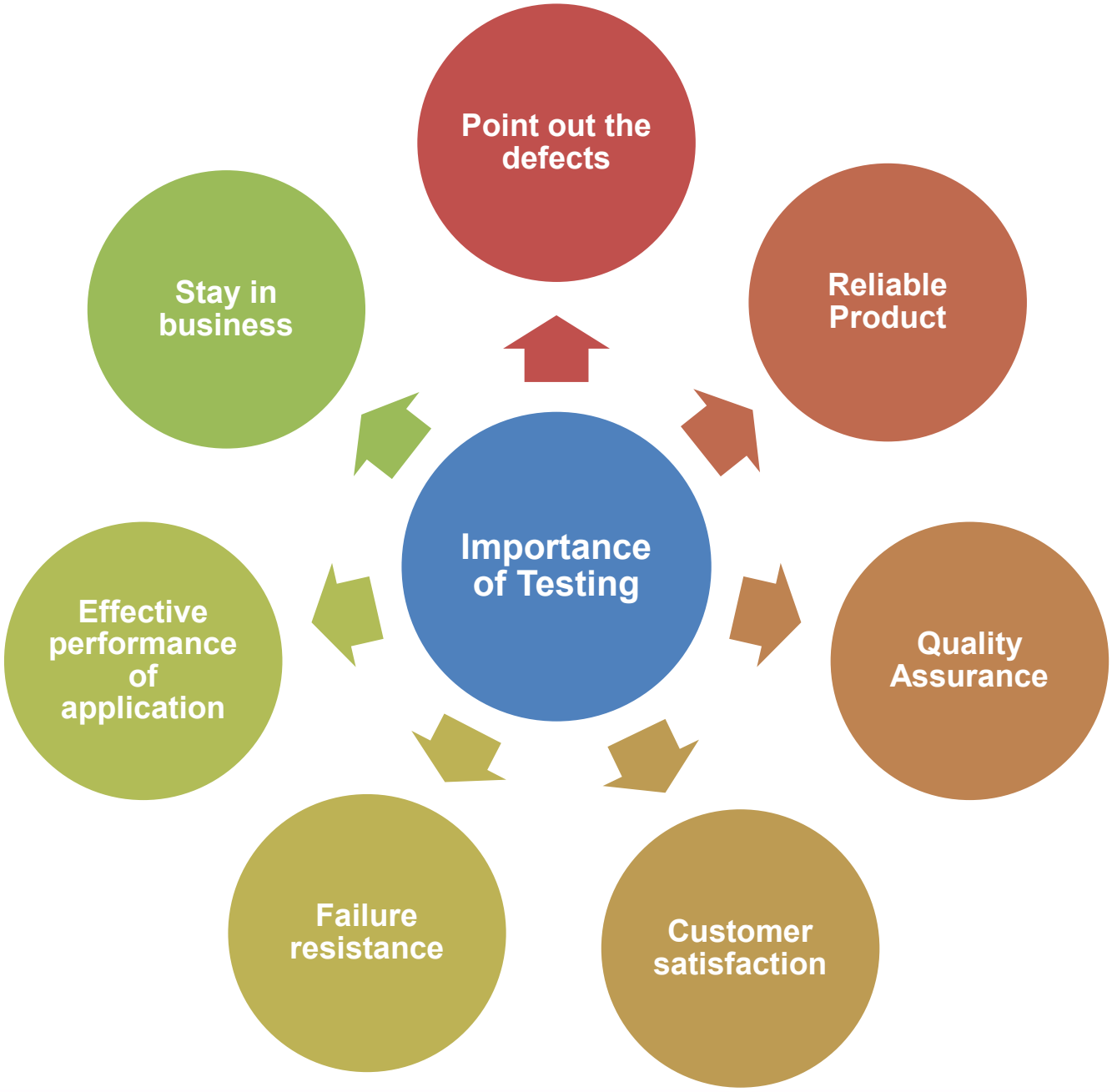
Software deployment frequencies of different companies

Source: O'Reilly

Company	Deploy Frequency	Deploy Lead Time	Reliability	Customer Responsiveness
Amazon	23,000 / day	minutes	high	high
Google	5,500 / day	minutes	high	high
Netflix	500 / day	minutes	high	high
Facebook	1 / day	hours	high	high
Twitter[2]	3 / week	hours	high	high
typical enterprise	once every 9 months	months or quarters	low/medium	low/medium

TOPICS

- Disasters due to lack of testing
- Definition of testing
- Importance of testing
- Difference in developer and tester perspective



University of Victoria

Importance of testing

- Software testing is really required to point out the defects and errors that were made during the development phases.
- It's essential since it makes sure that the customer finds the organization reliable and their satisfaction in the application is maintained.
  - If the customer does not find the testing organization reliable or is not satisfied with the quality of the deliverable, then they may switch to a competitor organization.
  - Sometimes **contracts** may also include **monetary penalties** with respect to the timeline and quality of the product. In such cases, if proper software testing may also prevent monetary losses.
- It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence
- Testing is necessary in order to provide the facilities to the customers like the delivery of high-quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.

Importance of testing cont..

- Testing is required for an effective performance of software application or product.
- It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.
- It's required to stay in the business.
  - **Users** are not inclined to use software that has bugs. They may not adopt a software if they are not happy with the stability of the application.
  - In case of a product organization or **startup** which has only one product, poor quality of software may result in lack of adoption of the product, and this may result in losses which the business may not recover from.

University of Victoria

You can be sued for a software defect

- You can be sued for damages if your application crashes and causes financial harm.
- Fortunately, **errors and omissions insurance (E&O)** can help pay for legal costs when a client blames your software for financial harm.
- If a problem with your software could lead to property damage or physical injuries, you might want to invest in **product liability insurance** as well.

University of Victoria

Ref: https://www.insureon.com/blog/what-you-need-to-know-about-software-liability

End user license agreements

- Most software vendors use contracts or end user license agreements to **define the terms of the relationship**.
- It's becoming quite common for these contracts to include language that **limits a software developer's liability**, even if the software doesn't work. Courts have typically upheld these contracts and limited the liability of the software company.
- Though software developers enjoy a larger degree of protection than medical device companies, **contracts and license agreements may not always shield you from personal injury and tort claims**. Sometimes complex circumstances can leave you vulnerable.
- For example, 3D printing uses software and hardware to create a physical product. If a software defect caused the 3D printer to produce a flawed product that physically injured someone, the developer could face a lawsuit.

University of Victoria

Current software liability laws-Tort Law and Contract Law

For someone to successfully sue a software developer for negligence, they must prove that:

- The software vendor had a **duty** to provide functioning software to the user (Duty of Care)
- The software **did not perform** as promised (Standard of Care)
- The user **suffered harm** (Damages/Injury)
- The **software caused** that harm (Causation)
- These criteria apply when there's no contract between the developer and client, according to standard **tort law**.

If someone can prove that your work caused **financial harm**, they will likely be compensated for their damages in court.

University of Victoria

Rigorous testing and quality control is key to preventing software-related injury

- Software defects can be disastrous, leading to costly settlements or court-ordered judgments.

University of Victoria

Duty of Care

- The mere fact that the software contains a defect does not mean that the developer will have been negligent.
- If the developer took **all the steps that a reasonable software developer** in its position would have taken, it will have discharged its duty of care and will **not be liable in negligence** even if, despite having taken those steps, defects remain in the software.
- These steps encompass those that the Developers have taken to thoroughly test the code until they are satisfied that it is ready for release.

University of Victoria

Manufacturer Knowledge and Liability in a ‘Failure to Warn’ claim

- A manufacturer will not necessarily escape liability if it did not know about the risk for which it failed to provide warnings. Actual knowledge is not required in a **failure to warn claim**.
- If the manufacturer reasonably should have known about the risk, it may be held liable for the injuries.
- A defendant is under a duty to stay knowledgeable about its product. If it was possible to discover the risk through reasonable research, **testing** and investigation, the defendant will be held liable for failing to warn about a risk it *should have known* about.
- The courts would normally assume that the client is less technical than the developer and would put the duty of care to the developer.

University of Victoria

Due Diligence

- Helps you in the ‘Due Diligence’ Defence in a liability claim.
- Company has taken reasonable steps to ensure the software application meets the functional and non-functional requirements set out in the SRS document by adequate testing.

TOPICS

- Disasters due to lack of testing
- Definition of testing
- Importance of testing
- Difference in developer and tester perspective

But testing is done by developers too !

University of Victoria

Developers think: ‘How can I make the application?’

Testers think: ‘How can I break the application?’

University of Victoria

How can I make it?

The task of the developer is to find a possibility ***"to create something"***. The developer knows how the code works, he can not criticize his own product, as well as the artist can not criticize his own picture or sculpture. Developers are very confident and thinks that their code is the best one, bugs and defects are out of the question ;).

How can I break it?

The task of the tester is ***"to break something"***. He is purposeful in discovering gaps and all kinds of problems in the system, to ensure that the potential client or user will not find them. He is like always to ask "Hey, what if a user presses this button now?".

Developers know perfectly the code architecture, that's why it is rather more difficult for them to put themselves in the place of users.They have too much information about the program. Often the developer 'knows' how it should work, and assumes (!) that it does. Which inevitably leads to test cases being omitted.

Testers evaluate the product from the end-user's point of view, who doesn't know how the program works. They are required to test it using various techniques that allow them to simulate the way the product will eventually work in the real world.

The developers have their own approach to the perception of the soft - they, generally, concentrate on a ***single component or feature*** in their product. As a rule, different developers write different parts of the code. Therefore all of them has no clear vision of how the system works as a whole and the components' interrelationship.

The tester concentrate on an entire system. He observes ***the system as a whole*** better, than separate parts or components. He sees the system as a interrelationship among it's components. And since QA engineers don't usually go deep into implementation methods and tons of code, they are able to look at problems from different perspectives.