Preventing Safety Recalls in a Legacy Software Medical Device

The legacy product:

- Mature
- Reliable
- Lots of code
- Lots of features

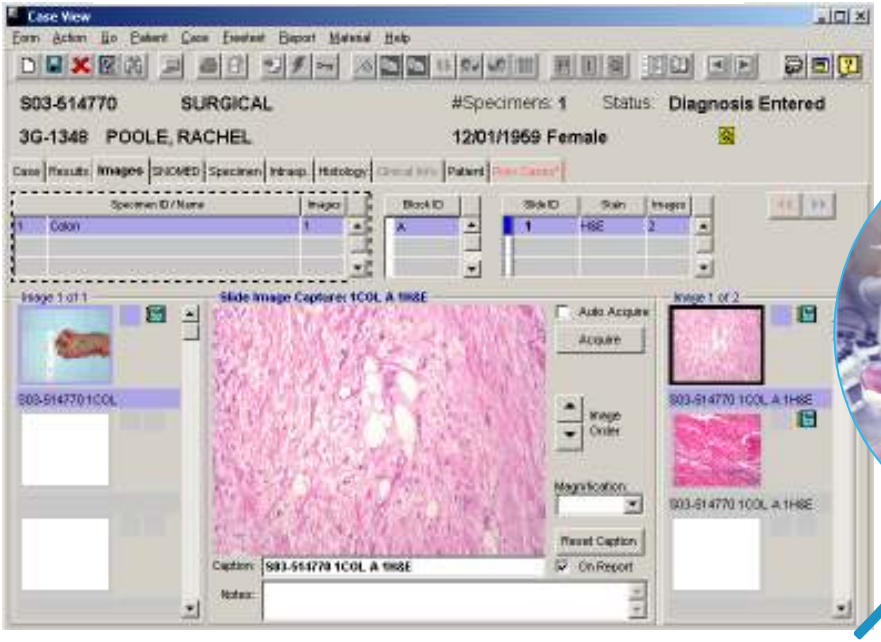
What if safety concerns drove us to search for further potential unknown issues?

Where would one start?

It could feel like an insurmountable challenge...

... this is the story ...

What's a Laboratory Information System?



Its main uses are to:

- Take orders from physicians
- Manage order through specimen testing
- Report the results

Its users are:

- Lab assistants
- Lab techs
- Pathologists
- Physicians
- Ward nurses & clerks

A Typical Use Case

Medical devices improve patient care

But, if something goes wrong… they can endanger the patient


How Do We Manage Risk?

Risk Management Process

Key elements of risk management process

ISO 14971

- Risk analysis
- Risk evaluation
- Risk control
- Production and post-production information

Key components of risk

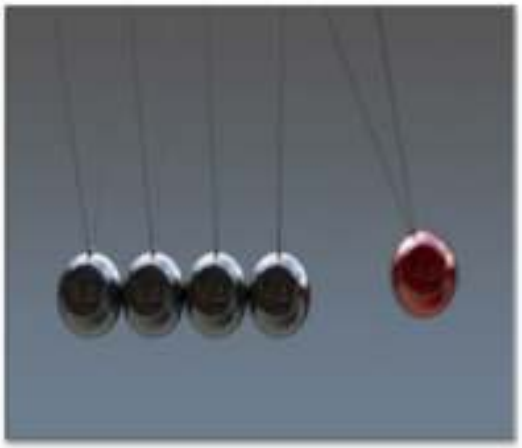
- Probability of occurrence
- Consequences of harm

#1 Software Risk: Incorrect treatment decisions

- Blood type mix-up
- Operating on wrong organ
- Prescribing wrong medication or dosage

Risk Control: Causes and Mitigations

Some typical causes



- Patient/data mismatch
- Incorrect information
- Missing information

Inherent safe design

- Patient banner on every form to prevent patient mix-up
- Resending messages not acknowledged by target system
- Error checking on lab results based on acceptable ranges

Information for safety:

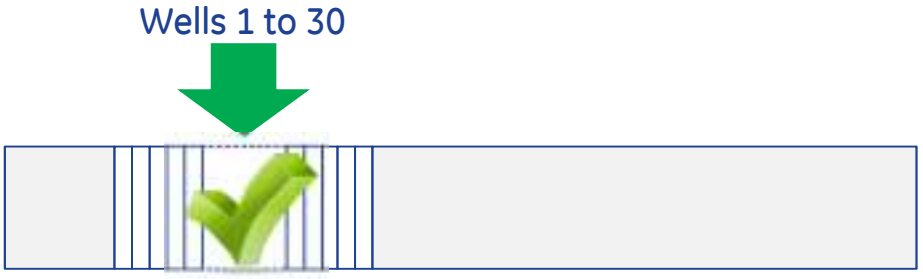
- Cautionary information in user and service manuals


When software fails…
…what could go wrong?
… some examples …

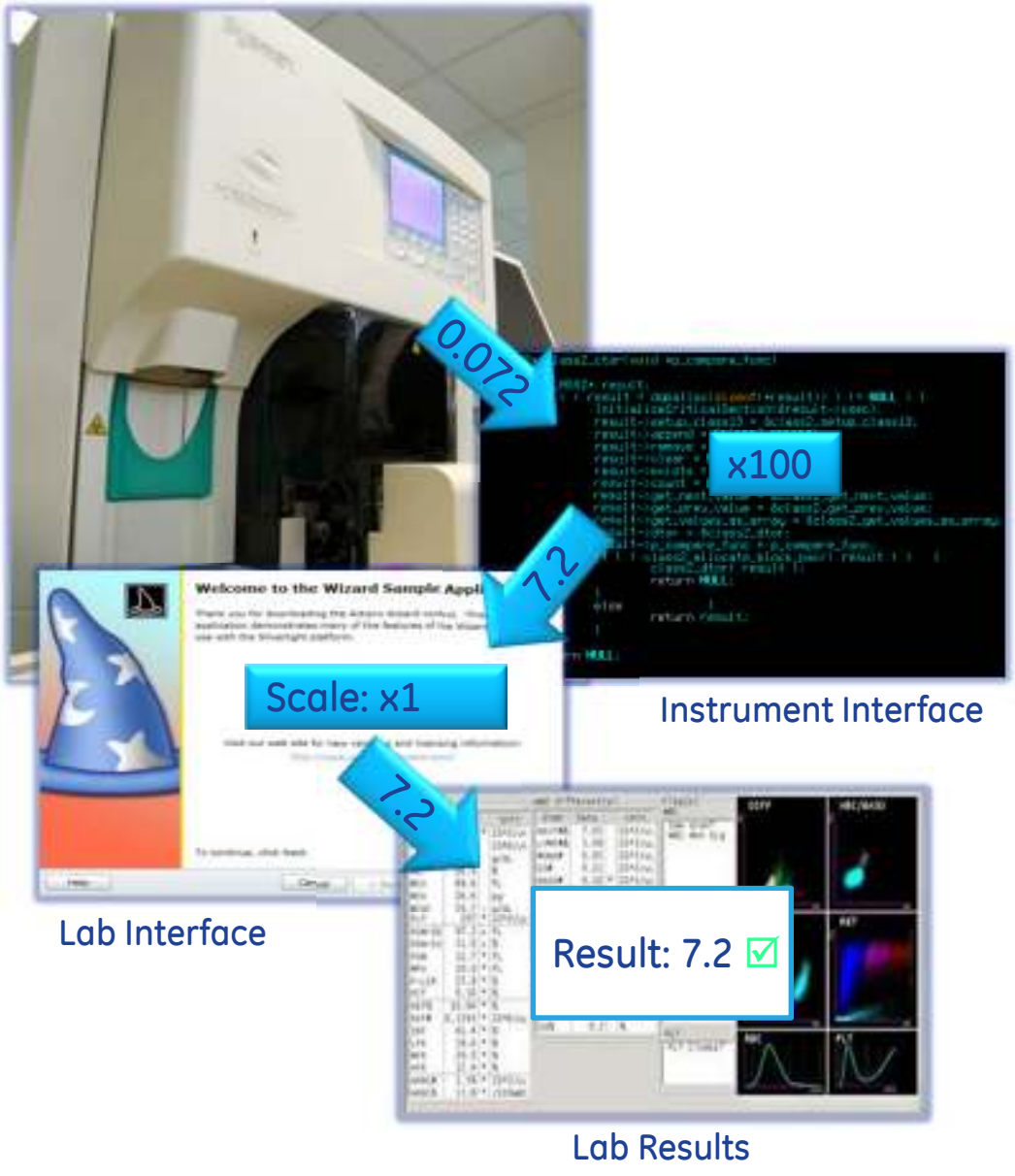
Incorrect Antibiotic Susceptibility Results…

Antibiotic susceptibility tests:

- Designed to identify most effective antibiotic
- Cards have 30 wells
- Technician places culture & antibiotic in each well
- Instrument measures resistance / susceptibility
- Physician prescribes most effective antibiotic



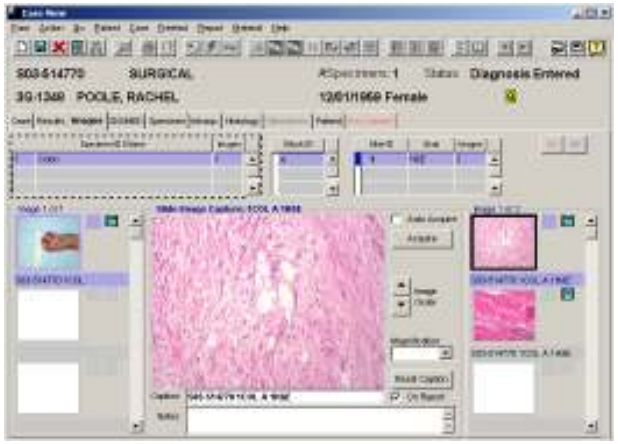
Incorrect Test Results…



Lab instrument interfaces:

- Collect lab results from instruments
- Instrument interface: Device driver for instrument
- Lab interface: User interface for configuring instruments
- Data flows from instrument, through interfaces, to lab
- Clinicians retrieve lab results:
  - Directly from Lab
  - From external applications connected to Lab

Mismatched Addendum…



External interfaces:

- Lab can exchange data with external systems
- A common interface is with Electronic Health Records

- The interface can transmit diagnostic reports
- Addendums provide additional Dx information
- Addendums are attached after the fact (e.g. detailed analyses)

Results Not Sent to Physician…

Handling panic results:

- Lab results that exceed limits require immediate attention
- Software can be configured to handle panic results:
  - Send to STAT printer
  - Add to phone list
- Clinical staff monitor STAT printers for immediate action
- Lab technicians use phone list to call physicians

What Did We Learn From These Issues?

What used to work can stop working in an evolving environment…

- Evolving technology
- Growing markets
- Changing standards
- More complex workflows
- Increased automation
- Increased reliance on software

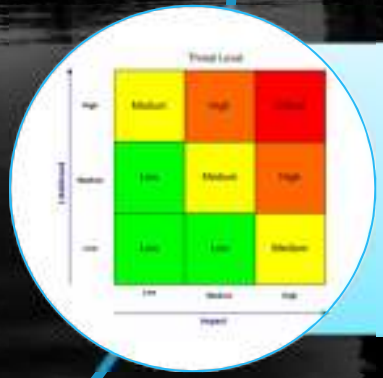
We needed an approach to find residual anomalies that could become potential safety issues

What Is CAPA?

- Problem Statement
- Evaluation
- Investigation
- Action Plan
- Implementation
- Effectiveness

Problem Statement:

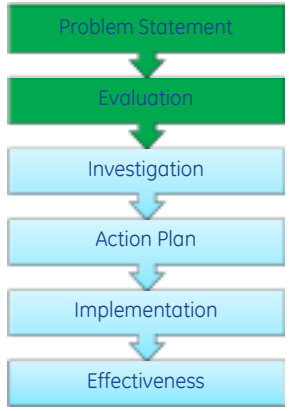
- Create a tracking record
- Define the problem



Evaluation:

- Identify scope
- Evaluate risk / impact level:
  - Health or safety
  - Quality or compliance

Problem Statement & Evaluation



Problem Statement

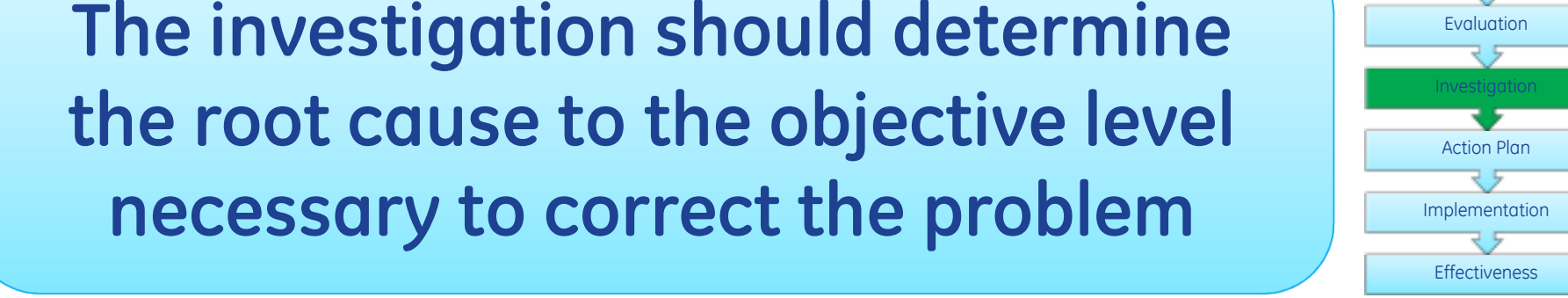
- Prevent recalls due to residual anomalies in legacy code

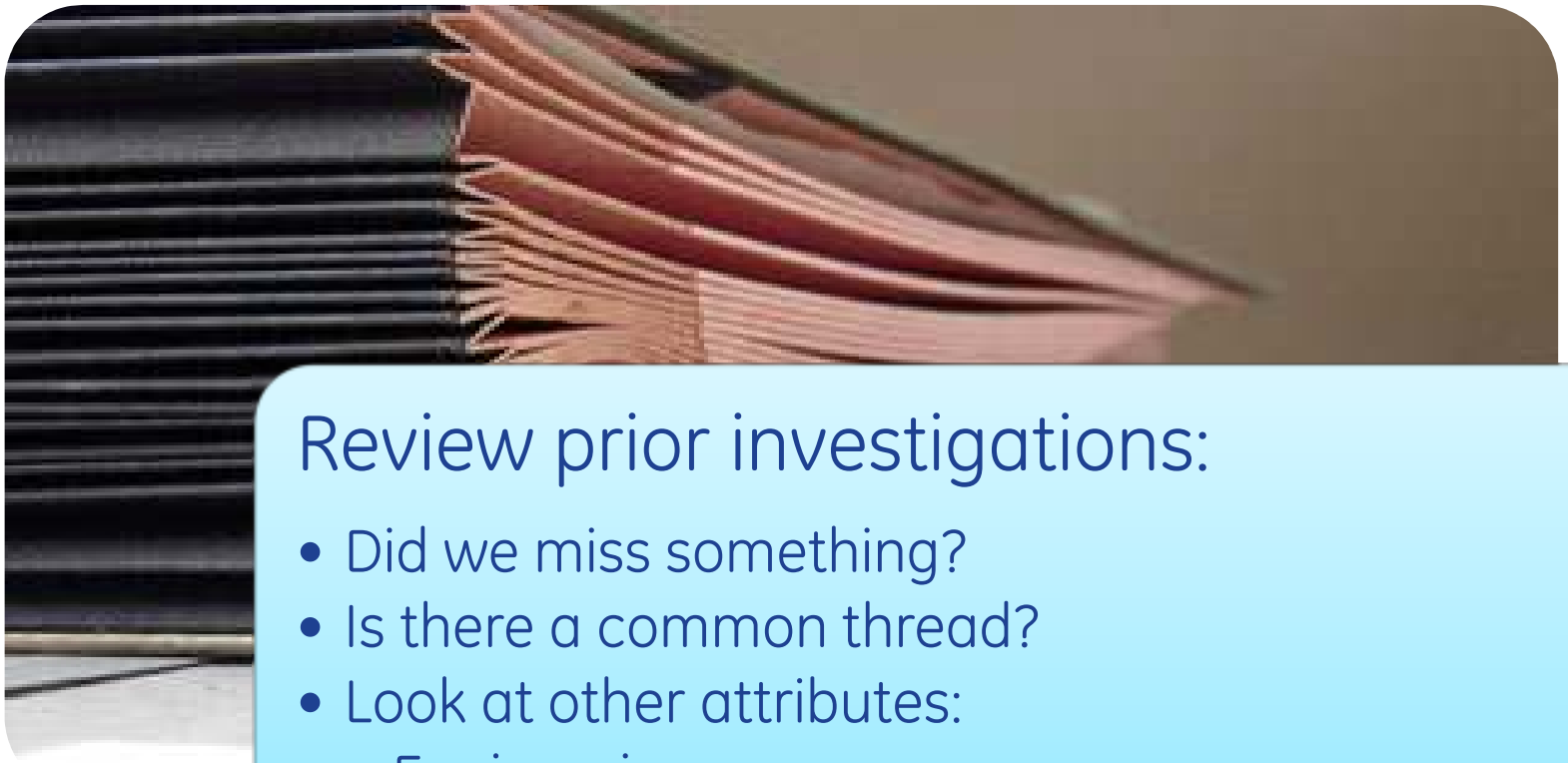
Evaluation

- ☐High ☑**Medium** ☐ Low

- Not High because no adverse events
- Not Low because of potential safety issues
- Medium is appropriate – requires investigation to root cause

The investigation should determine the root cause to the objective level necessary to correct the problem





Review prior investigations:

- Did we miss something?
- Is there a common thread?
- Look at other attributes:
  - Engineering cause
  - Module

Finding Root Cause: 5-Why Analysis

Date:

Contradiction Matrix Participants:

Lead	[Enter Name Here]	
Clincal	[Enter Name Here]	
Engineering	[Enter Name Here]	
Engineering	[Enter Name Here]	
QARA	[Enter Name Here]	

Description of the Problem

Scoring
1: Minimal factor to contribute to creation of FMI
3: Potential/Partial factor to contribute to creation of FMI
5: Definite factor to creation of FMI

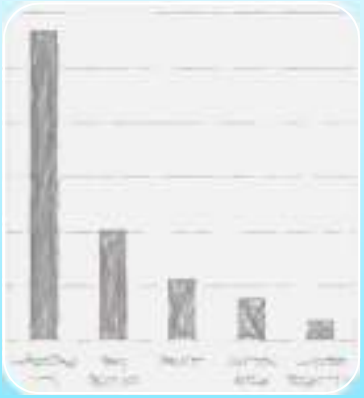
What factors are directly responsible for the problem?

	Potential main causes	Why?	Why?	Why?	Why?	Why?	[Enter Name Here]	[Enter Name Here]	[Enter Name Here]	[Enter Name Here]	[Enter Name Here]	
Cause 1												0
Cause 2												0
Cause 3												0
Cause 4												0
Cause 5												0
Cause 6												0
Cause 7												0
Cause 8												0
Cause 9												0

Additional Comments/Notes

a.
b.
c.
d.
e.
f.

What Did We Conclude From This?



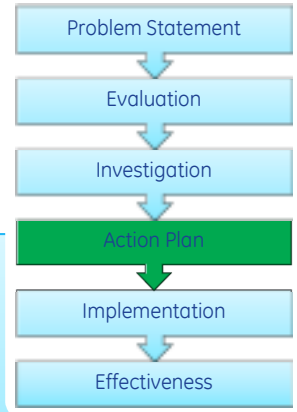
All issues due to legacy code:

- Implemented many years ago
- Implemented prior to current QMS

What should we do now?

- We have the controls for new code
- We need to address the legacy code

Figuring Out A Plan



Brainstorming yielded several solutions…

Combining solutions gave best results

Use filters to identify highest risk areas

- Use code complexity to identify problem areas
- Use risk analysis to identify code tied to risk

Conduct code review to find anomalies

- Use splint to identify potential coding mistakes
- Use coding standards that address engineering causes from recalls

Code Complexity

Strategy: Focus on the most complex code because it is more likely to contain errors

Cyclomatic code complexity

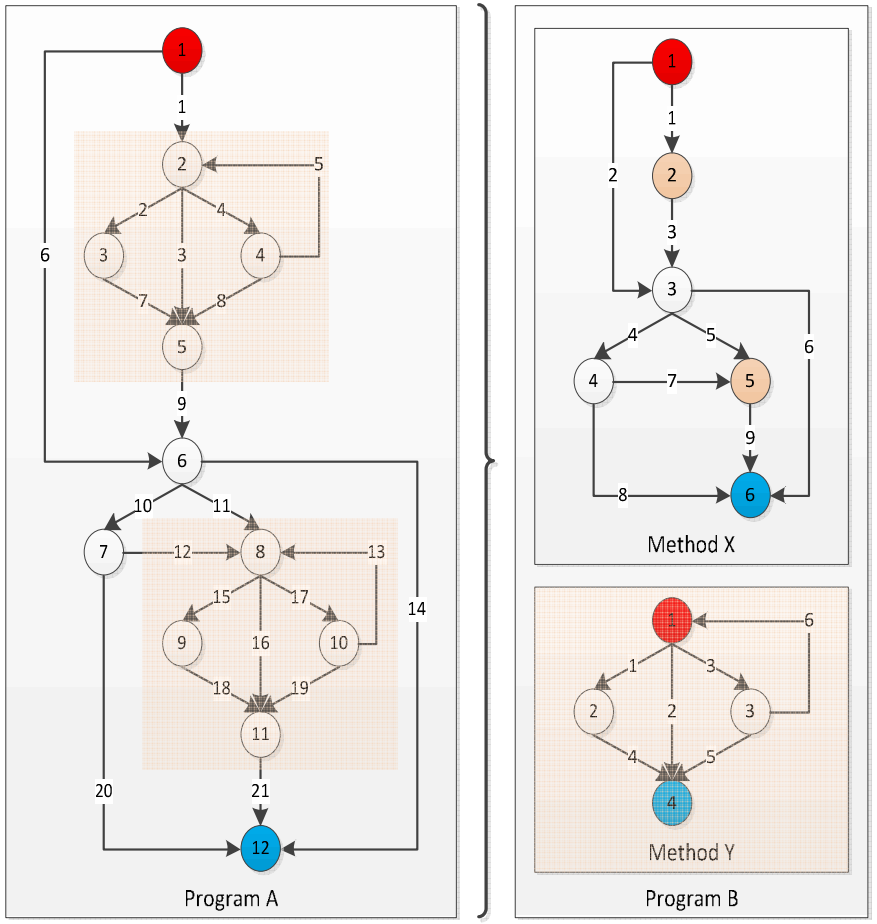
- # of logic paths through the code
- Easily calculated: E – N + 2P

Complex software units are…

- More likely to have coding errors
- Harder to test
- Harder to understand
- Harder to change
- More difficult to reuse
- Longer to produce

For example, compare…

- Program A: 11
- Program B:
  - Method X: 5
  - Method Y: 4



Risk Analysis

Strategy: Focus on code associated with potentially hazardous situations

Leverage established Risk Assessment Control record (RAC)

RAC identifies potential risks from software malfunction

Retain code linked to hazards – examples…

- Patient identifying data
- Lab results

Exclude code unrelated to risk – examples…

- Billing
- Insurance claims
- Statistical reports

Splint

Strategy:

- Use splint on selected code files
- Evaluate splint results as part of code review

Splint is a C static analyzer that identifies potential coding mistakes:

- Buffer size overflow
- Data type checking
- Uninitialized variables
- Memory management

```
int sampleFunc()
{
    char typedChar;
    while (typedChar != 'x');
    {
        typedChar = getchar();
        if (typedChar = 'x')
            return 0;
        switch (typedChar) {
        case '\t':
            printf("You entered tab\n");
        default:
            printf("%c",typedChar);
        }
    }
    return 0;
}
```

```
int sampleFunc()
{
    char typedChar = (char) 0;
    while (typedChar != 'x'){
        typedChar = (char) getchar();  /* Type casting to char */
        if (typedChar == 'x')/* Fixed :Make it as comparison operator */
            return 0;
        switch (typedChar){
        case '\t':
            printf("You entered tab\n");
            break; /* Added break statement to prevent fall-through */
        default:
            printf("%c",typedChar);
            break;
        }
    }
    return 0;
}
```

Code Review

Strategy: Use code review on selected code to identify residual software anomalies

Update standards to address anti-patterns

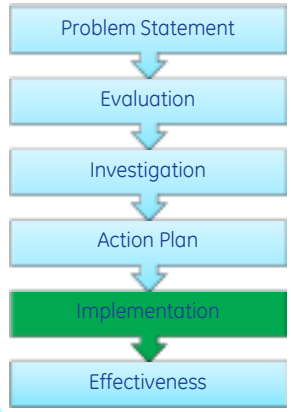
- Buffer overflow
- Inadequate error handling
- Uninitialized variables

Review output from risk and complexity filters

- Rely on subject matter expertise to analyze output from tools
- Leverage most senior developers to conduct the reviews
- Focus on instances that could lead to a hazardous situation

Leverage updated coding standards for all new code changes

Implementation

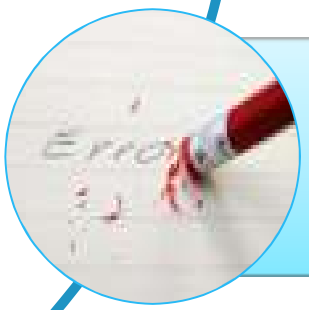


… so we execute the action plan …

- Identify subset of files using code complexity and risk filters
- Conduct in-depth code analysis on the subset

Find potential safety issues – examples:

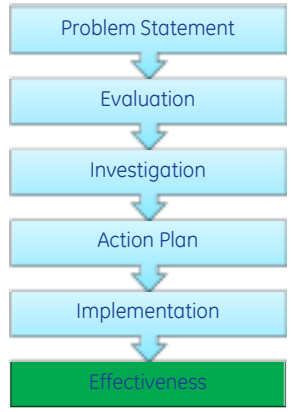
- Error return code issue could lead to data/patient mismatch
- Buffer overflow issue could lead to incorrect results being reported



Report all issues into complaint handling system:

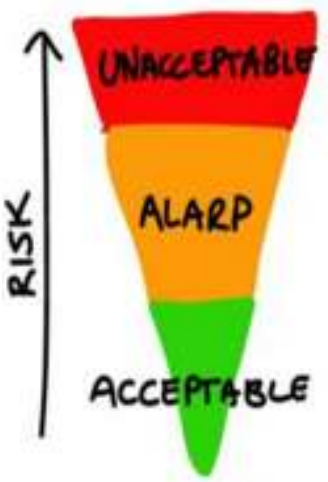
- Follow process to deploy corrections to the field

Effectiveness…



How to gauge effectiveness of a preventive action?

- One measure: Monitor the hazardous complaint rate
- What if the hazardous complaint rate is zero?
- We can only claim that we reduced the likelihood



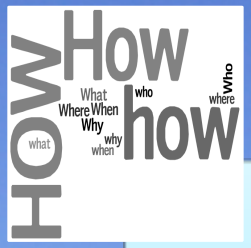
We took all reasonable measures to ensure our product is safe

Risk was reduced to as low as reasonably practicable (ALARP)

Closing Thoughts…

Why search for legacy defects?

- For our patients & families
- For our customers
- For the regulators
- For our peace of mind
- Doing everything we can to ensure our product is safe



How to find legacy defects?

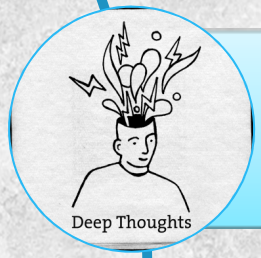
- Use principled approach:
  - Focus on risk
  - Focus on probability
  - Leverage tools
- Drive due diligence with efficacy of effort

These challenges can feel insurmountable at first…

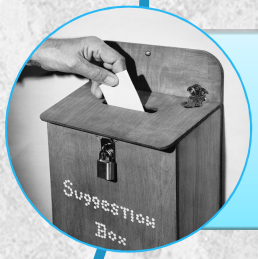
But with a principled approach,
they can be straightforward.

The hardest part is figuring out the principles.

Questions?



Thoughts?



Suggestions?

What could be improved?