SOCIETY OF ACTUARIES®

2019 ANNUAL MEETING & EXHIBIT

October 27-30
Toronto, Canada

Session 024: Applying FinTech and IT Principles to Actuarial Modeling

Session 24: Applying FinTech and IT Principles to Actuarial Modeling

Igor Nikitin, ASA and M. Crew Sullivan, FSA

October 28, 2019

SOCIETY OF ACTUARIES
Antitrust Compliance Guidelines

Active participation in the Society of Actuaries is an important aspect of membership. While the positive contributions of professional societies and associations are well-recognized and encouraged, association activities are vulnerable to close antitrust scrutiny. By their very nature, associations bring together industry competitors and other market participants.

The United States antitrust laws aim to protect consumers by preserving the free economy and prohibiting anti-competitive business practices; they promote competition. There are both state and federal antitrust laws, although state antitrust laws closely follow federal law. The Sherman Act, is the primary U.S. antitrust law pertaining to association activities. The Sherman Act prohibits every contract, combination or conspiracy that places an unreasonable restraint on trade. There are, however, some activities that are illegal under all circumstances, such as price fixing, market allocation and collusive bidding.

There is no safe harbor under the antitrust law for professional association activities. Therefore, association meeting participants should refrain from discussing any activity that could potentially be construed as having an anti-competitive effect. Discussions relating to product or service pricing, market allocations, membership restrictions, product standardization or other conditions on trade could arguably be perceived as a restraint on trade and may expose the SOA and its members to antitrust enforcement procedures.

While participating in all SOA in person meetings, webinars, teleconferences or side discussions, you should avoid discussing competitively sensitive information with competitors and follow thse guidelines:

- Do not discuss prices for services or products or anything else that might affect prices
- Do not discuss what you or other entities plan to do in a particular geographic or product markets or with particular customers.
- Do not speak on behalf of the SOA or any of its committees unless specifically authorized to do so.
- Do leave a meeting where any anticompetitive pricing or market allocation discussion occurs.
- Do alert SOA staff and/or legal counsel to any concerning discussions
- Do consult with legal counsel before raising any matter or making a statement that may involve competitively sensitive information.

Adherence to these guidelines involves not only avoidance of antitrust violations, but avoidance of behavior which might be so construed. These guidelines only provide an overview of prohibited activities. SOA legal counsel reviews meeting agenda and materials as deemed appropriate and any discussion that departs from the formal agenda should be scrutinized carefully. Antitrust compliance is everyone’s responsibility; however, please seek legal counsel if you have any questions or concerns.

Presentation Disclaimer

You know
nothing
Jon Snow
©
Game of Thrones

Why do we change platforms

- Scalability issues
- Too difficult to update
- Technological advancements
- Better alternatives in the market

At some point every model was new and awesome!
And at some point the time comes to refactor it!

Advantages and Disadvantages of In-house Models

Advantages

- Innovation
- Speed of change

Disadvantages

- Cost of original build
- Maintenance
- Expertise required for original build

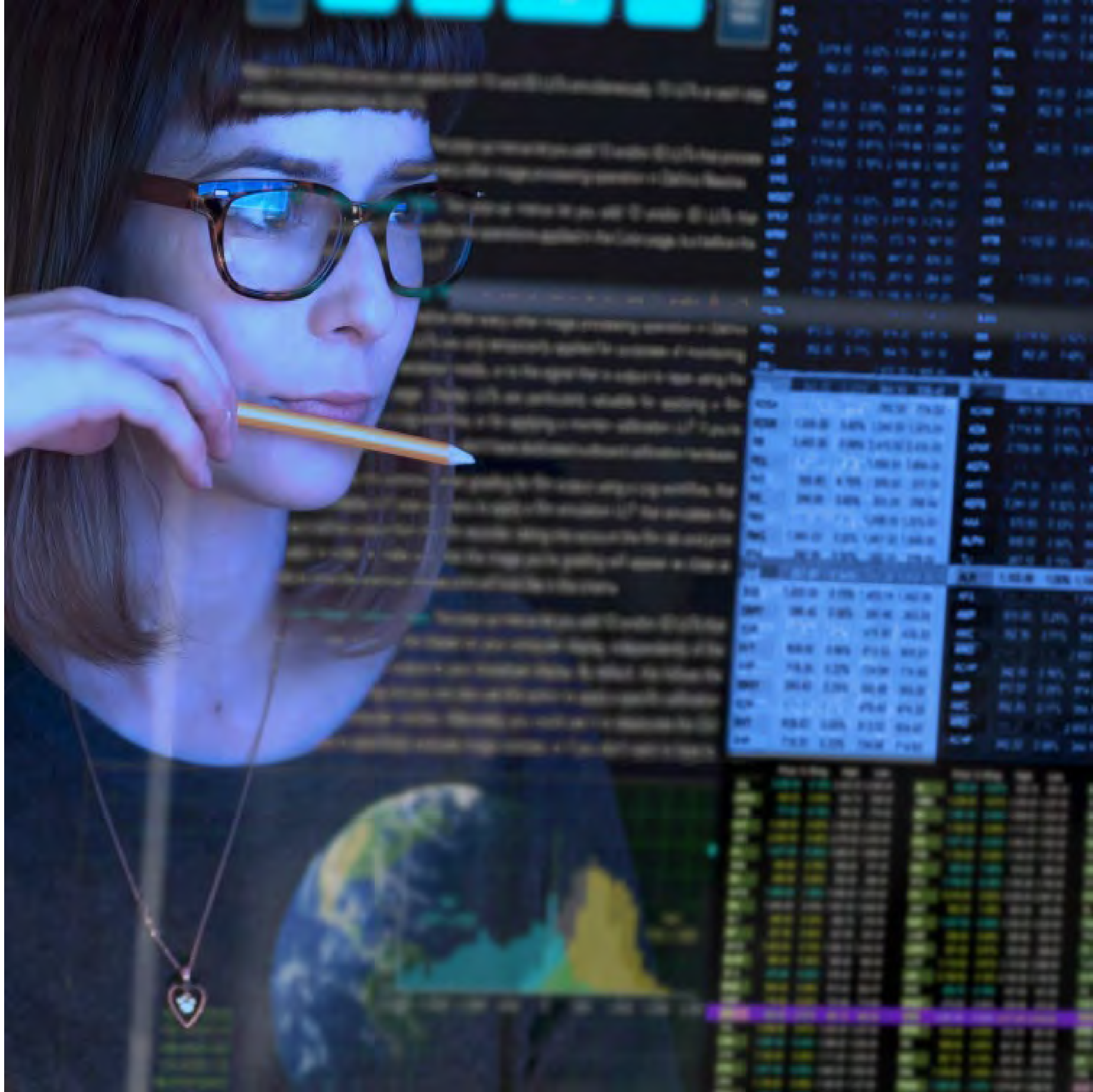
Necessary Resources

People with the right skillsets:

- **Software engineer**
- **Actuary**
- **Programmer**

Time:

- **Design**
- **Implement**
- **Test**



Stakeholder Buy-in

- Describe a future state that:
  - Addresses <u>their</u> priorities / pain points
  - Explains how payoff over time is worth the upfront investment
  - Broader benefits (e.g., cross functional use)
  - Long term maintenance plan

Design Phase

Design goals

- Pick one or two qualities that your software should maximize.
- These will act as your tie breakers when multiple valid choices exist.

Runtime

Time to Completion

Code organization

Procedural Programming – operates on data using functions and procedures.

+ Easiest to conceptually grasp

+ Fastest possible runtime

+ Great choice for small programs

- Hardest to maintain

Object Oriented Programming – managing data and functionality with classes.

+ Easiest to maintain

+ Easiest to develop with large team

- Conceptually more complex

Functional Programming - treats functions as data.

+ Most flexible

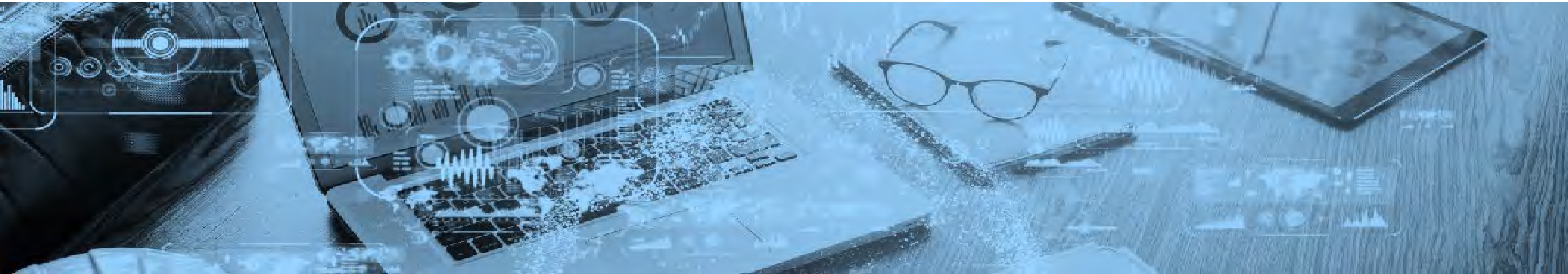
+ Fastest to develop with

- Least controlled & hardest to debug

Software design principles

- Set of the most general and highest- level aspirations for software
- Examples include:
  - Separate code that varies from code that stays the same.
  - Program to an interface, not an implementation.
  - Favor composition over inheritance.

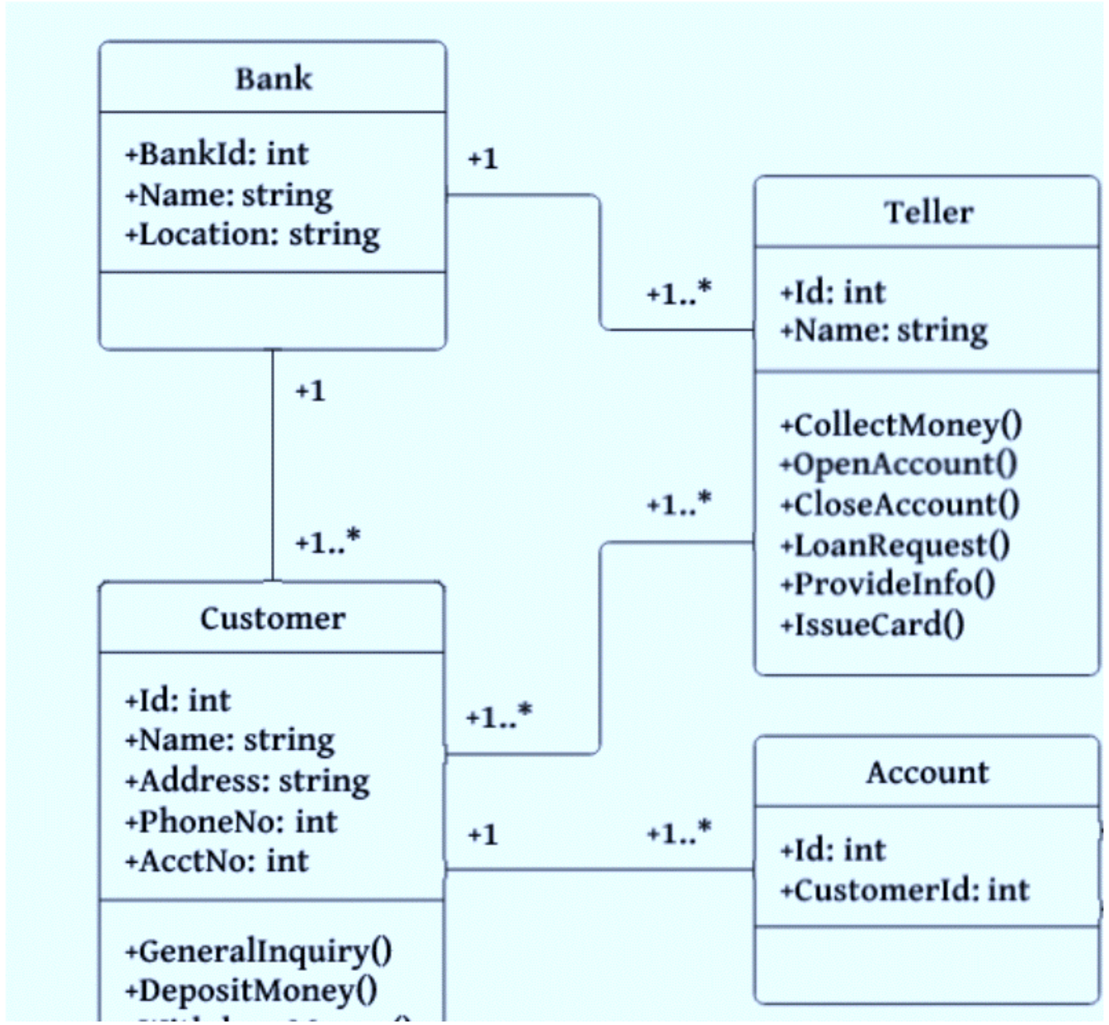
A good introductory discussion is available here: https://wiki.base22.com/display/btg/Core+Software+Design+Principles



Blueprint

Unified Modeling Language (UML):

- Define object responsibilities and communication protocols
- Enable parallel development by multiple programmers
- Conduct review of the system prior to writing code

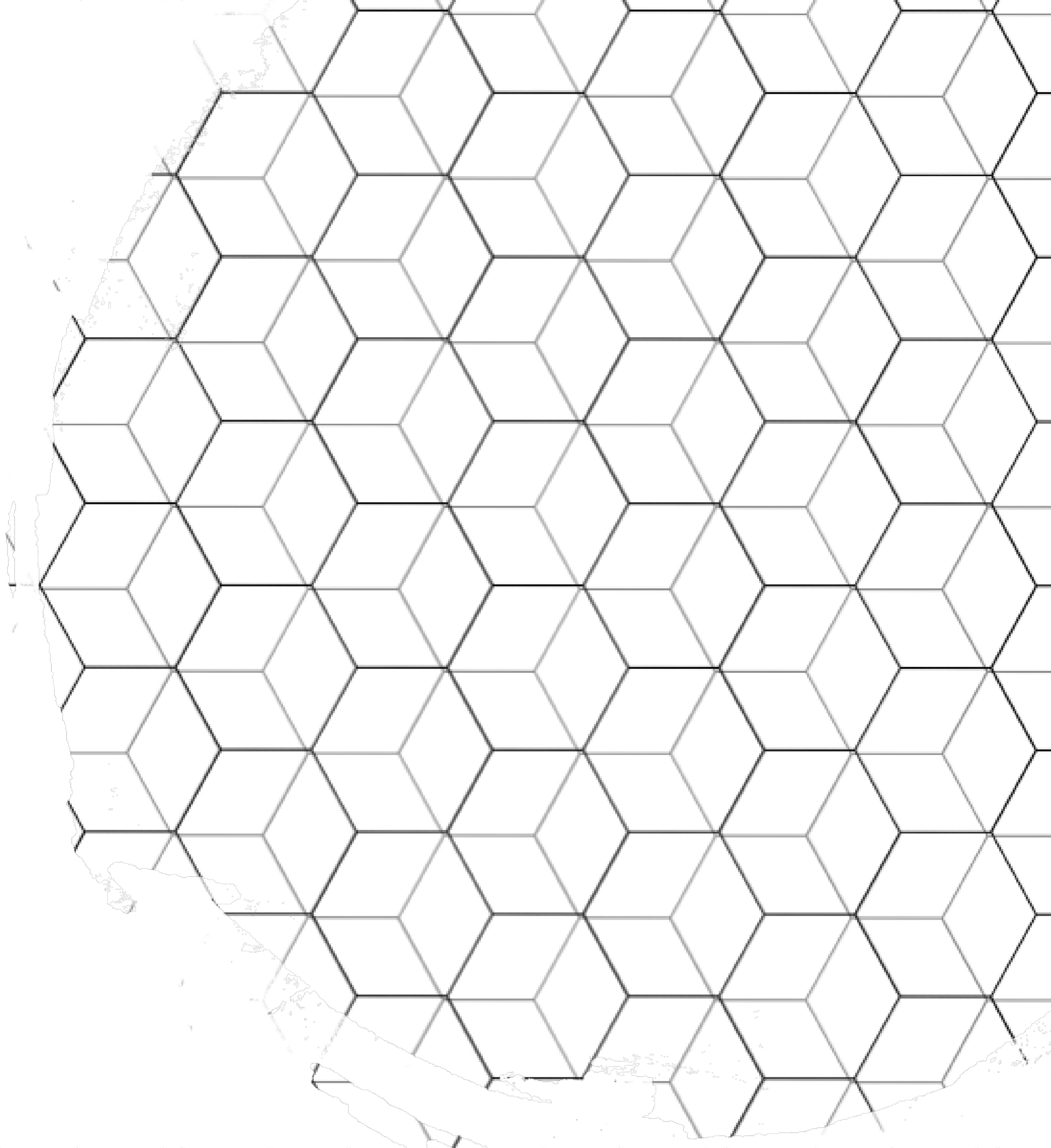


Design patterns

- A design pattern is a general repeatable solution to a commonly occurring problem in software design.

- Examples of design patterns are:
    - Strategy pattern
    - Factory pattern

- An outstanding introduction to design patterns is Head First Design Patterns by Eric Freeman and Elisabeth Robson.



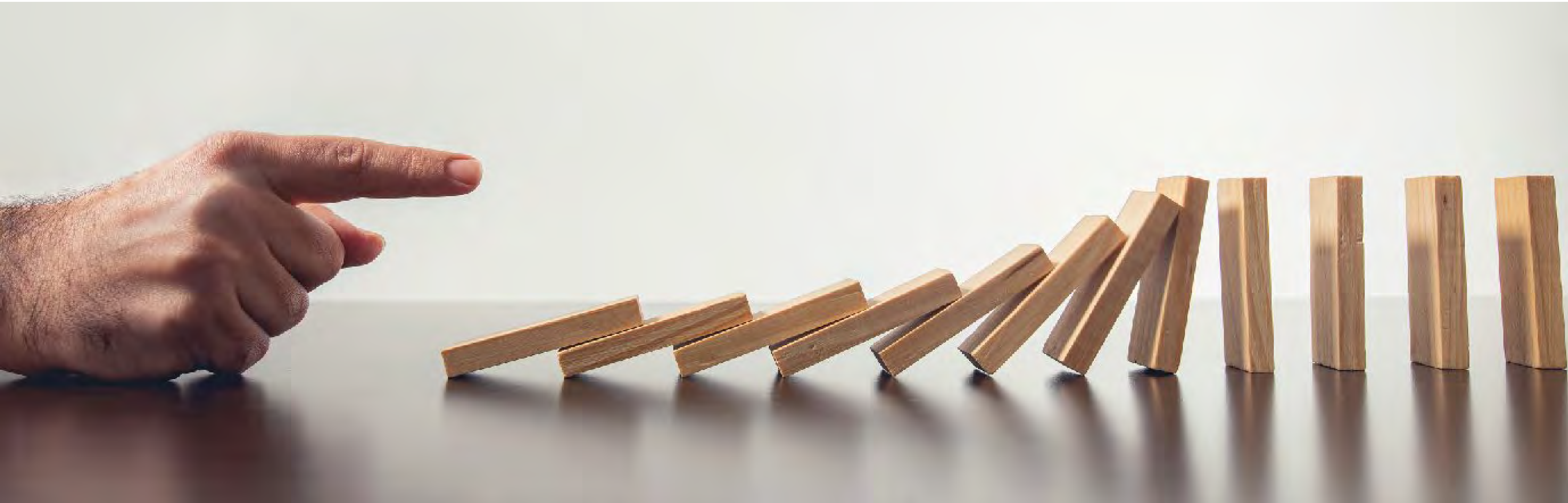
Programming language choice

Language choice considerations:

- Higher or lower level?
- Specialized libraries?
- Grid?
- Availability of developers?

VBA is generally a poor choice due to scalability and runtime issues!

Execution Phase



Utilize a Style Guide

- Clearly lays out the rules for writing perfect code.
- Promotes code readability and uniformit
- One of the most popular style guides is Google C++ style guide, which is available: https://google .github.io/styleguide/cppguide.html.

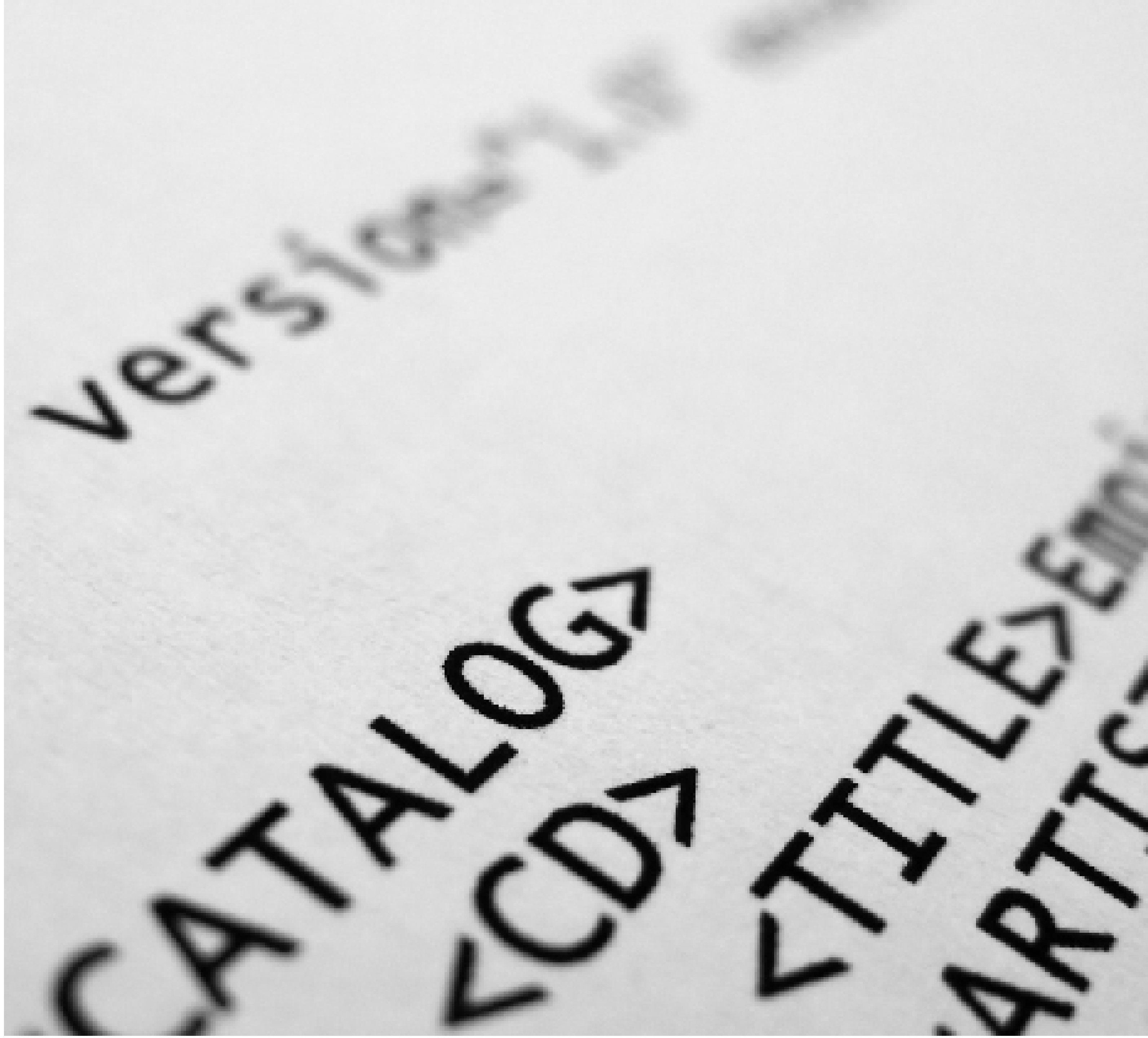
```
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
elif _operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

    #selection at the end -add back the deselected mirror modifier object
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob is the active ob
```

Because we all write perfect code, but somehow a lot of code is less than perfect.

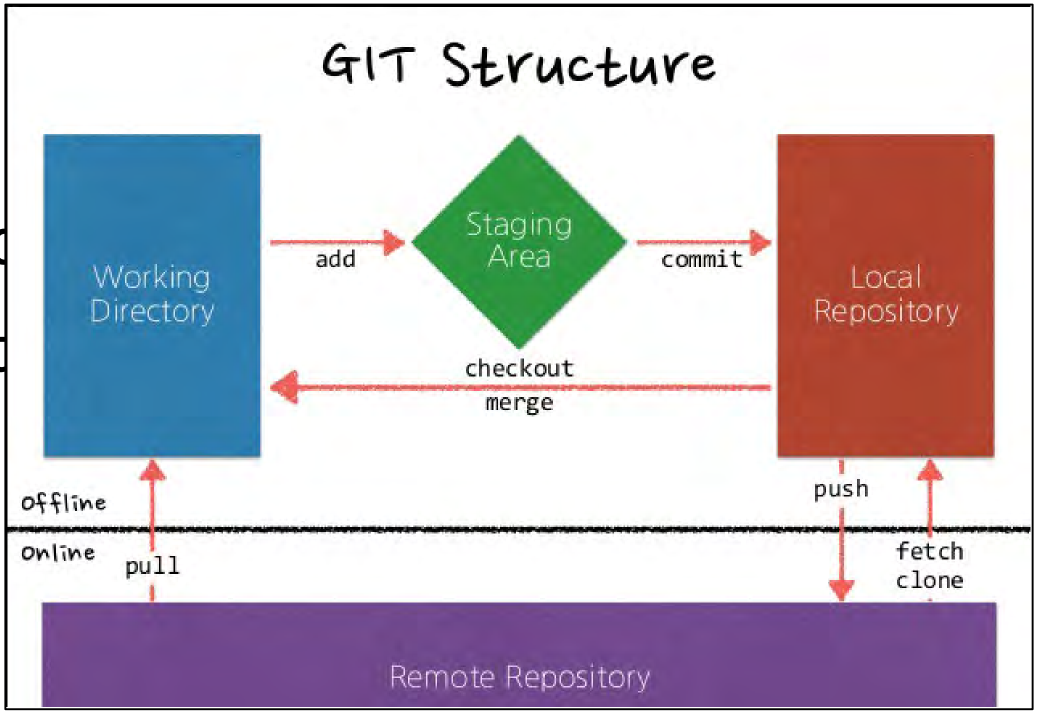
XML Doc

- Writing documentation is easiest when you are coding the logic, not a month later when the testing is complete.
- XML doc allows you to integrate documentation into the code and automatically produce html documentation.



Version Control System (Git)

- Enables orderly co-development with multiple people
- Merging code is easy and straightforwarc
- Provides audit trail and rollback capabilit



Learn more about GIT

http://nvie.com/posts/a-successful-git-branching-model/

Input Structure

- Consider multiple formats: csv, Excel, XML, JSON, database, something else?
- Consider generic data transfer capabilities

Error Handling

- Error messages should be:
  - Descriptive
  - Targeted to the audience:

    Age 154 exceeds maximum allowed age 120.

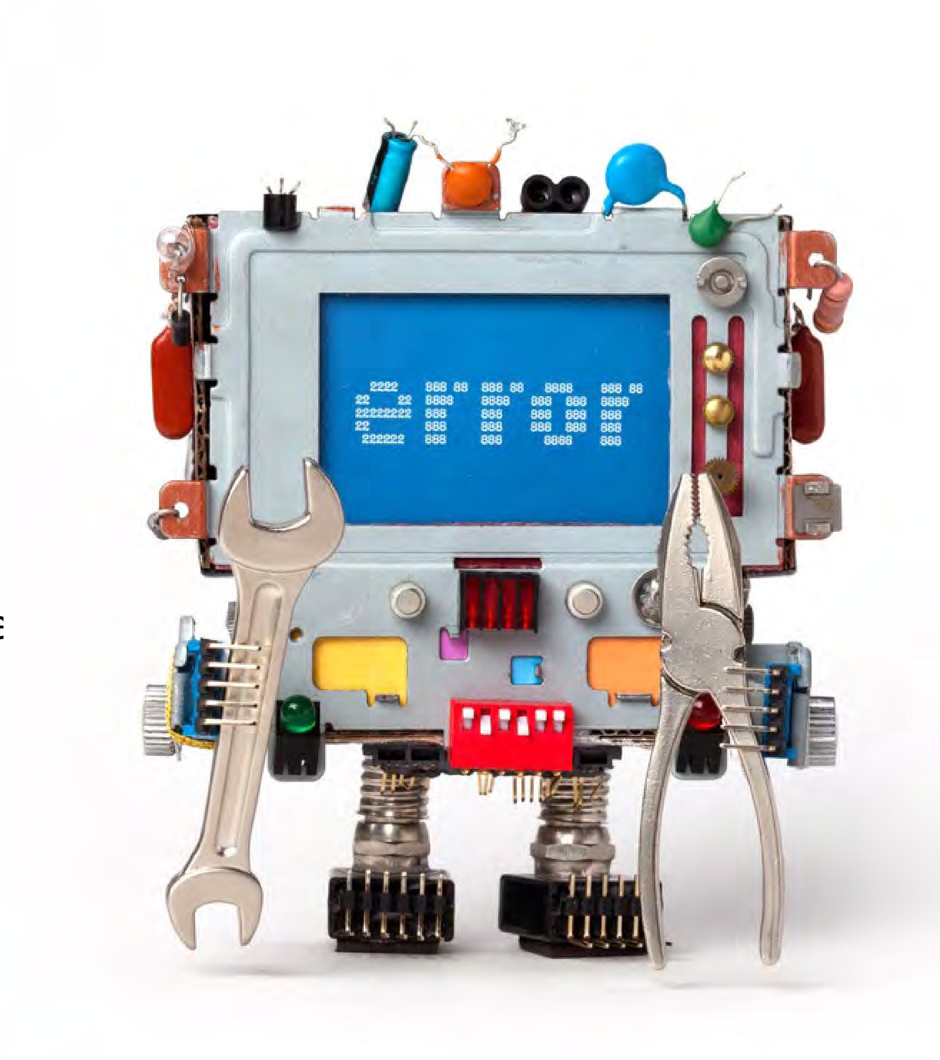
    Age.Calculate(…) encountered an exception.

    Policy.Calculate(…) of Policy 53453 encountered an exception.

    Contract.RunContract(folderPath C:\Test, inforceName testInforce

    parametersName testParameters.xml) encountered an exception.

- The can be implemented:
  - Using try catch blocks
  - Using logging libraries



Build Order

- Objects in an object oriented platform can be built in different orders.

Recommendations:

- Launcher
- Empty base classes and interfaces
- Input classes
- Detailed report
- Implementations of actual calculations

Unit Testing Framework

Testing is largely driven by amount of time that could be practically spent on it

- Unit testing frameworks enable very efficient regression testing on class level

Optimization

- **Optimization is definitely worth an investment of time and effort.**
- **Same code can be hundred times faster with relatively minor efficiency tweaks.**
- **Tools are available in visual studio and other IDEs to help with optimization.**

Testing phase

Build a quality test bank

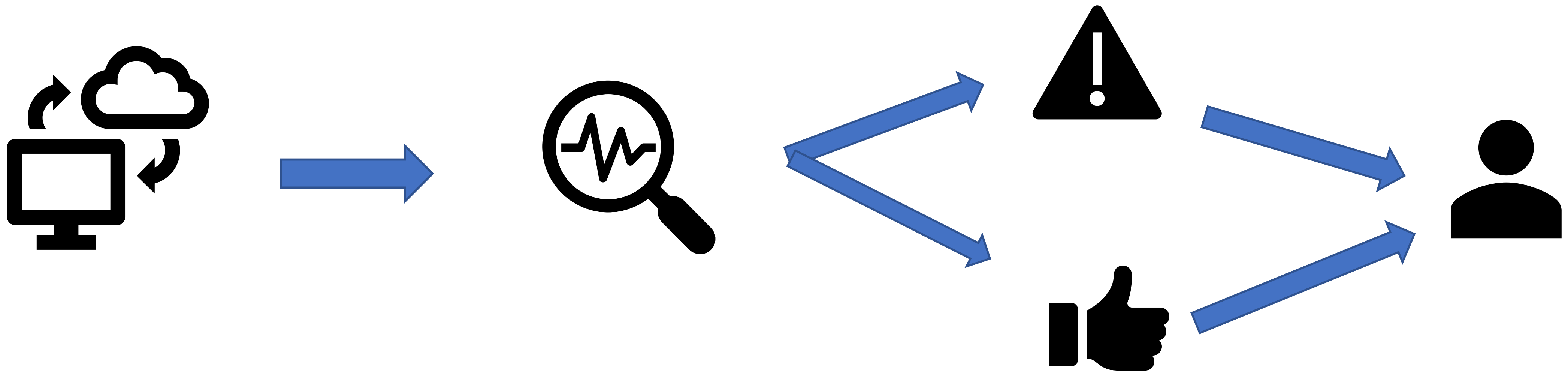
A well thought out Test bank will:

- Be fairly comprehensive
- Have reasonable run time
- Ensure nothing was broken
- Evolve as features are added

Automate regression testing

- The system should handle 1 test just as easily as 1 million tests.

- Efficiency of regression test often drives its quality.



Maintenance Phase

Maintain UML

- Large systems are hard to learn and maintain without UML
- Must keep up to date!

Don’t make a mummy

How do I unwrap the midsection but not touch the other parts?

Maintain documentati

- It is easiest to describe changes when you are making them.
- Remember you are writing for everyone who comes after you.
- There is no such thing as too much documentation (well maybe).

Questions