Driving Quality In-Between Requirements and Testing

Presented by:

Darin Kalashian and **Neha Chopra**

SQGNE Software Quality Group of New England

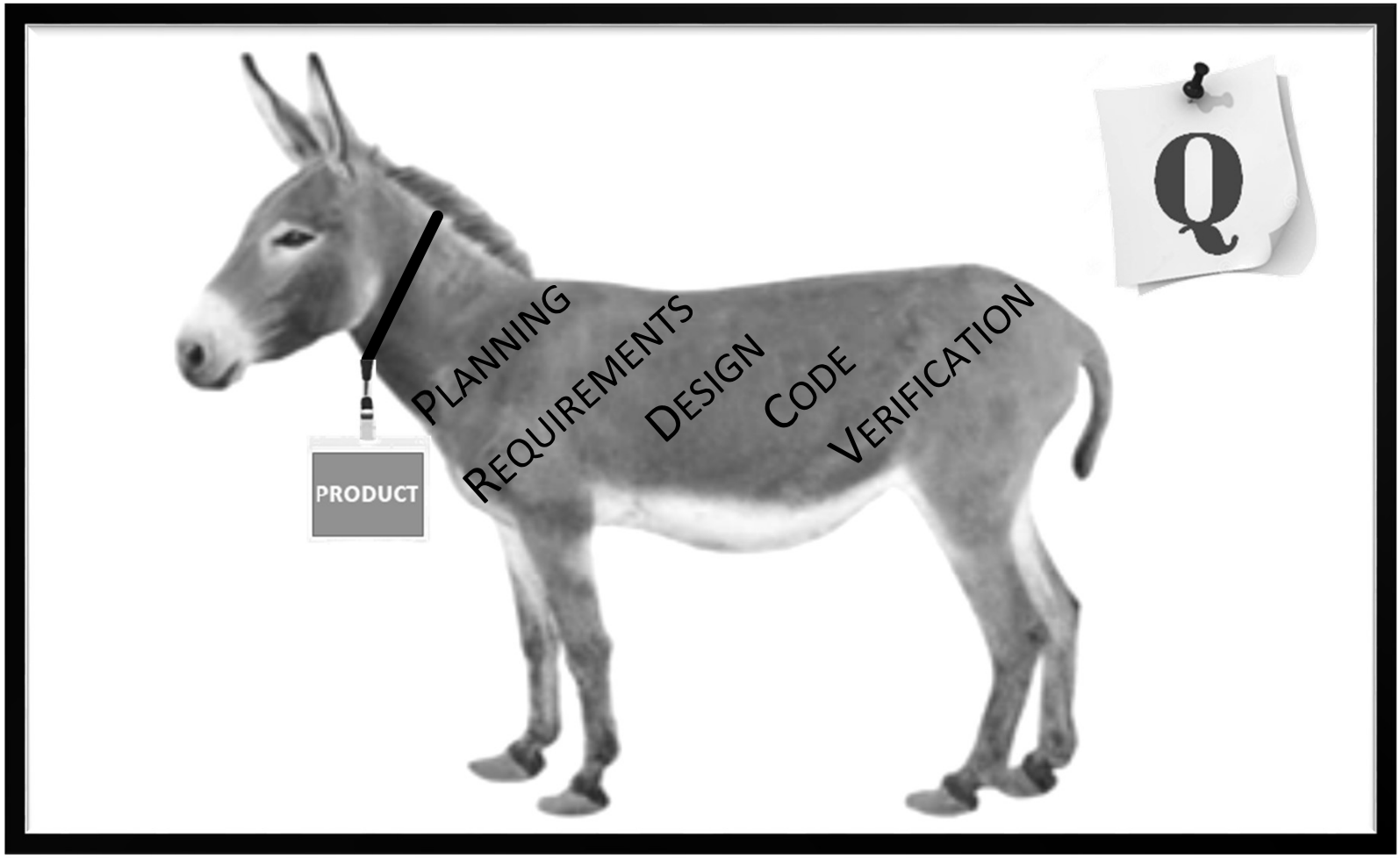
January 19th, 2022

Let's Warm Everyone Up With A Game First

Let's Play…

Pin The Quality On The Product

Pin The Quality On The Product



The Quality Engineering Misdirection

In most organizations today, you'll find Software Quality Assurance (SQA) mainly participating in only ***two (2)*** distinct phases of the Software Development Life Cycle (SDLC)

- ***Requirements Analysis Activities/Phase***

  "The goal of analysis is to determine where the problem is, in an attempt to fix the system. This step involves breaking down the system in different pieces to analyze the situation, analyzing project goals, breaking down what needs to be created, and attempting to engage users so that definite requirements can be defined" - wikipedia

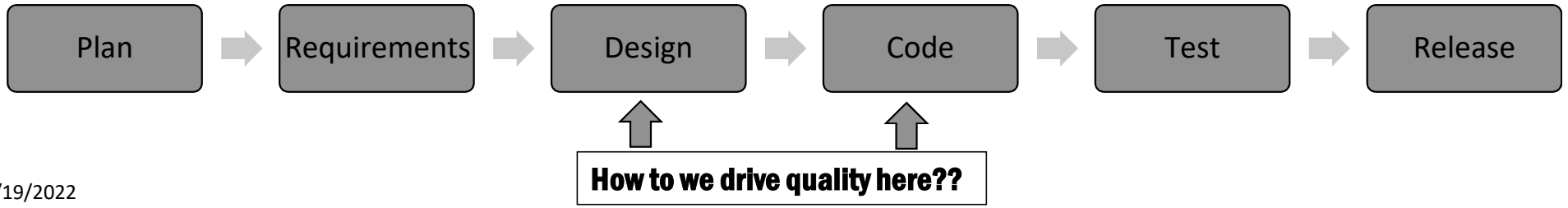
  - **Good User Stories**
  - **Acceptance Criteria**
  - **Testability**

- ***Testing Activities/Phase***

  "The execution of an Object Under Test (OUT) under specific preconditions with specific stimuli so that its actual behavior can be compared with its expected or required behavior"
  – Software Engineering Institute, Carnige Mellon

  - **Test Cases**
  - **Test Data**
  - **Test Results**

Looking at a typical SDLC:



Our Goals For This Presentation

Demonstrate how to engage software teams during *Design* and *Architecture* activities to ensure that the outputs are ready for construction and a seamless path to *production* deployment

- **Reduce** defects and risks in product design and architecture, long before there is even a product to test.
- Engaged to whole team to proactively **prevent** problems from being built into the product.
- As with all quality engineering patterns, the team should take **ownership** and drive ….but….

  …. *You can come across really **smaaaaart** if you are the one to introduce them to it*

Specifically, we will explore the following practices:

1. Building a model of the design/architecture to use as a proxy until we have an implementation
2. Use team brainstorming and collaboration techniques to identify risks and defects in the design
3. Figure out how to prioritize the identified risks so we address the more impactful ones first

Bonus Benefit: *Will reduce the load and dependency on testing to find defects in the "end-game", giving testers more time to do whatever they enjoy.*

(typically finding "other" ways to improve the product)

Software Modeling

As a Developmental Quality Tool

What is a Model and Why is it Useful?

What comes to mind when you think of what is a model?

1111

1111111111110

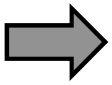
11110011111111000000

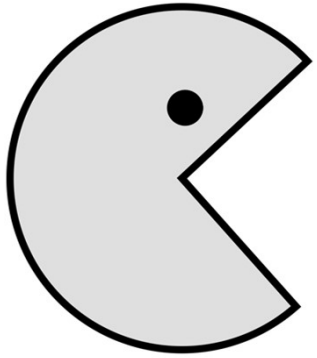
1111111111000000000000

11111111111111000000

1111111111110

1111





What is a Model and Why is it Useful?

What comes to mind when you think of what is a model?

An abstraction or representation, usually smaller and simpler, of what the final product will be when finished

*Models use familiar objects to represent **unfamiliar** things.*

*Models can help you **visualize**, or picture in your mind, something that is difficult to see or understand.*

*Models can help ~~scientists~~ engineers **communicate** their ideas, understand processes, and make predictions.*

-Texas Education Agency

Modeling and the use of Simulation on that model is a powerful tool for the evaluation and analysis of new system designs, modifications to existing systems and proposed changes to control systems and operating rules.

- J.S. Carson, Introduction to Modeling and Simulation,
IEEE Winter Simulation Conference, 2005.

We use a model as a tool to represent how a "system" could be realized
We can then use the model to explore simulated responses to real life situational events and stimuli.

Why Model A Software Application's Architecture?

Software Applications Tend To Be Very Complex

- Hard to Understand End-to-End or at the System Level
- Hard to communicate across the team and to external stakeholders

A model of our Software Architecture enables us to:

- See and provide feedback prior to implementation
  (does it support high cohesion and low coupling)
- Map requirements into the design *(What needs to do what/where)*
- Have a reference or blueprint for supporting, maintaining and testing

A Software Model is a High Value Quality Tool for Design Activities
It allows us to Identify Risks in the Design and Address them PROACTIVELY

STOP! Don't Pass Go (without a Design Model)

"Hey we're Agile! I have my User Story! Why can't I just start implementing?"

"We use the Unified Modeling Language (UML) and already do Class Diagrams and Sequence Charts. Isn't that the same"

Good Start, but you need to understand the following:

- The system from ***top-to-bottom, left- to-right***, not just one level like "Classes"
- The flow of data and control across the system
  (like a Sequence Diagram; but easier to translate into design)
- Identification of requirements to be implemented in the components of the design
- Any un-addressed risks in the system

Modeling the Design is a good investment to promote quality

May be the difference between releasing a

Production Ready design versus a ***Proof of Concept*** design

What Should A Software Model Provide?

The Model should identify:

- **Entities and interfaces :** *Defines what are we responsible for and what are we interfacing too*
  - ❑ External
  - ❑ Internal
- **Programmatic flow** : *How does data and control move though our system*
  - ❑ Data flow
  - ❑ Control Flow
  - ❑ Architecture Flow
- **Feature Abstraction** : *Understand the boundaries of functionality; where and with what purpose*
  - ❑ Top-To Bottom (System, Interfaces, Components, Objects, Modules, Classes, …)
  - ❑ Smaller, more Cohesive, less coupled design
  - ❑ Requirements Mapping

With this Model, we'll have a tool to aid in:

- **Communication**
  - ❑ Becomes a contract/reference point for expressing a common vision of the design
  - ❑ Reference for training/maintenance/debugging
- **Gathering Feedback** as a proxy for the implementation,
  - ❑ Allows us to "test" the design
  - ❑ Identifying risks
  - ❑ Identifying new requirements

Welcome to ND Industries: TikyMail Application

BUSINESS OPPORTUNITY:

Hard to communicate with younger generation, who don't want to "read" but would rather watch a "TikTok" video to communicate

OBJECTIVE:

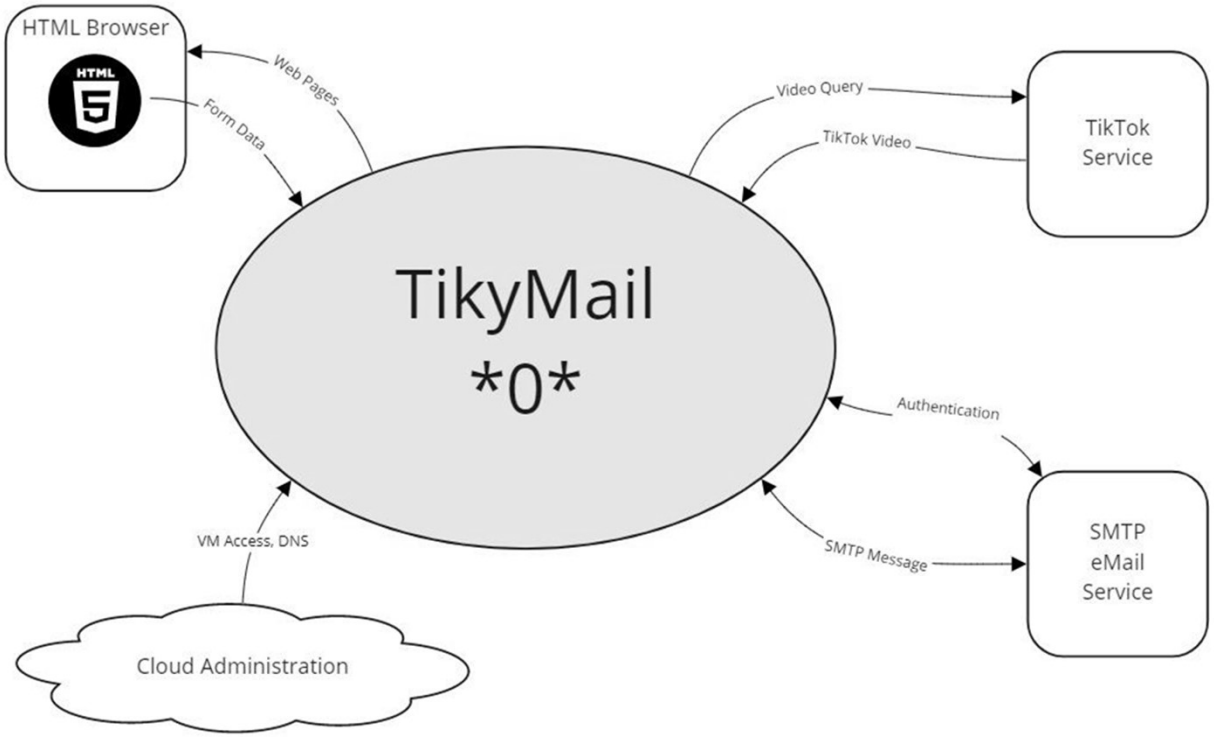
Build a system that can automatically turn a "text" based email into a TikTok video using a series publically available video snippets to communicate the message of the email.

TIKYMAIL (TM) REQUIREMENTS:

- TM SHALL[1] be a cloud based Software as a Service Application
  :
- TM SHALL[22] support any HTML5 compliant browser
  :
- TM SHALL[43] support a "preview" of the outgoing video prior to sending
  :
- TM SHALL[102] ensure all user information is stored encrypted at rest

MODEL: TikyMail Context Diagram (Level 0)

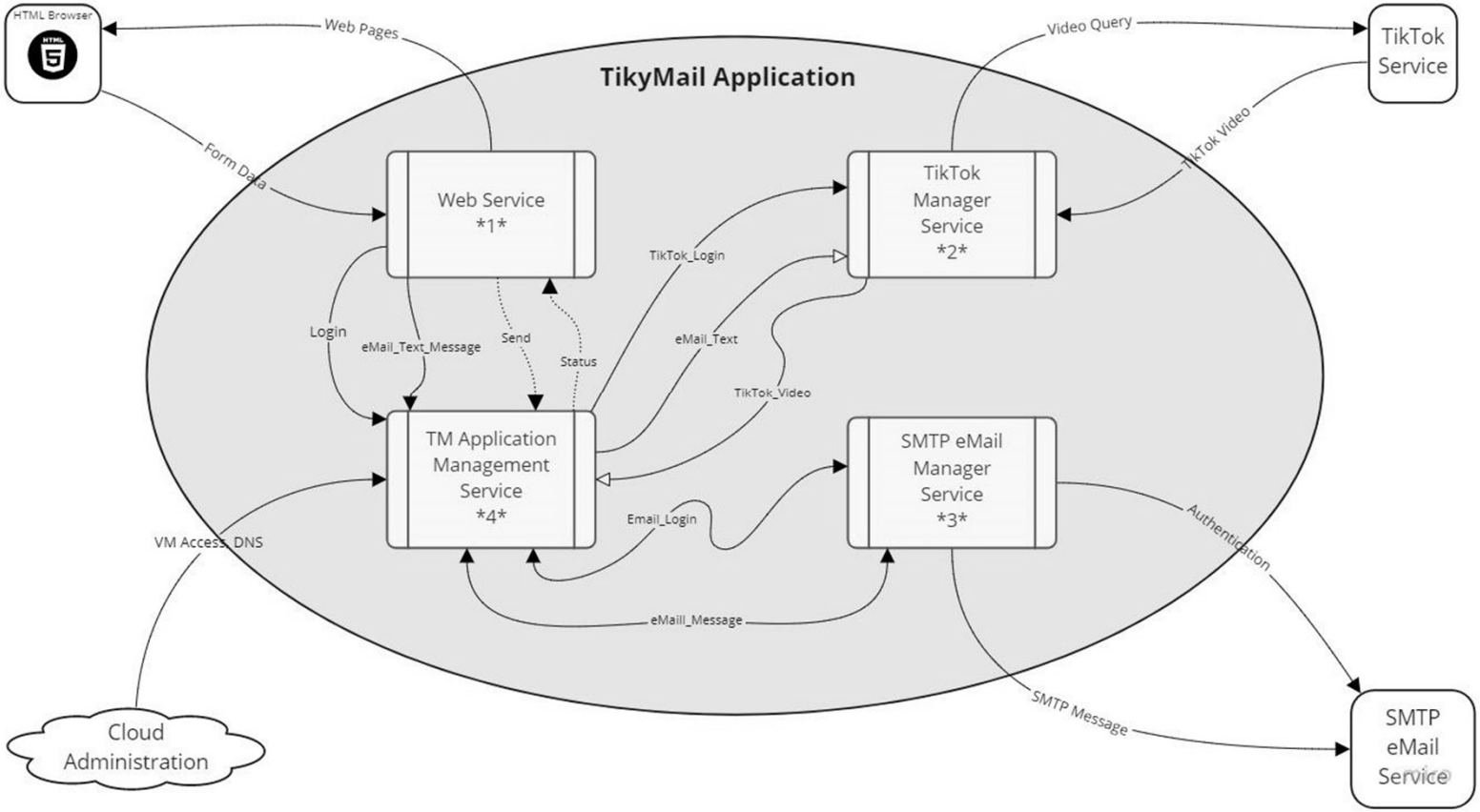
Objective: EXTERNAL .vs. INTERNAL; interfaces



https://miro.com/app/board/uXjVOYNHAy8=/?invite_link_id=725469525778

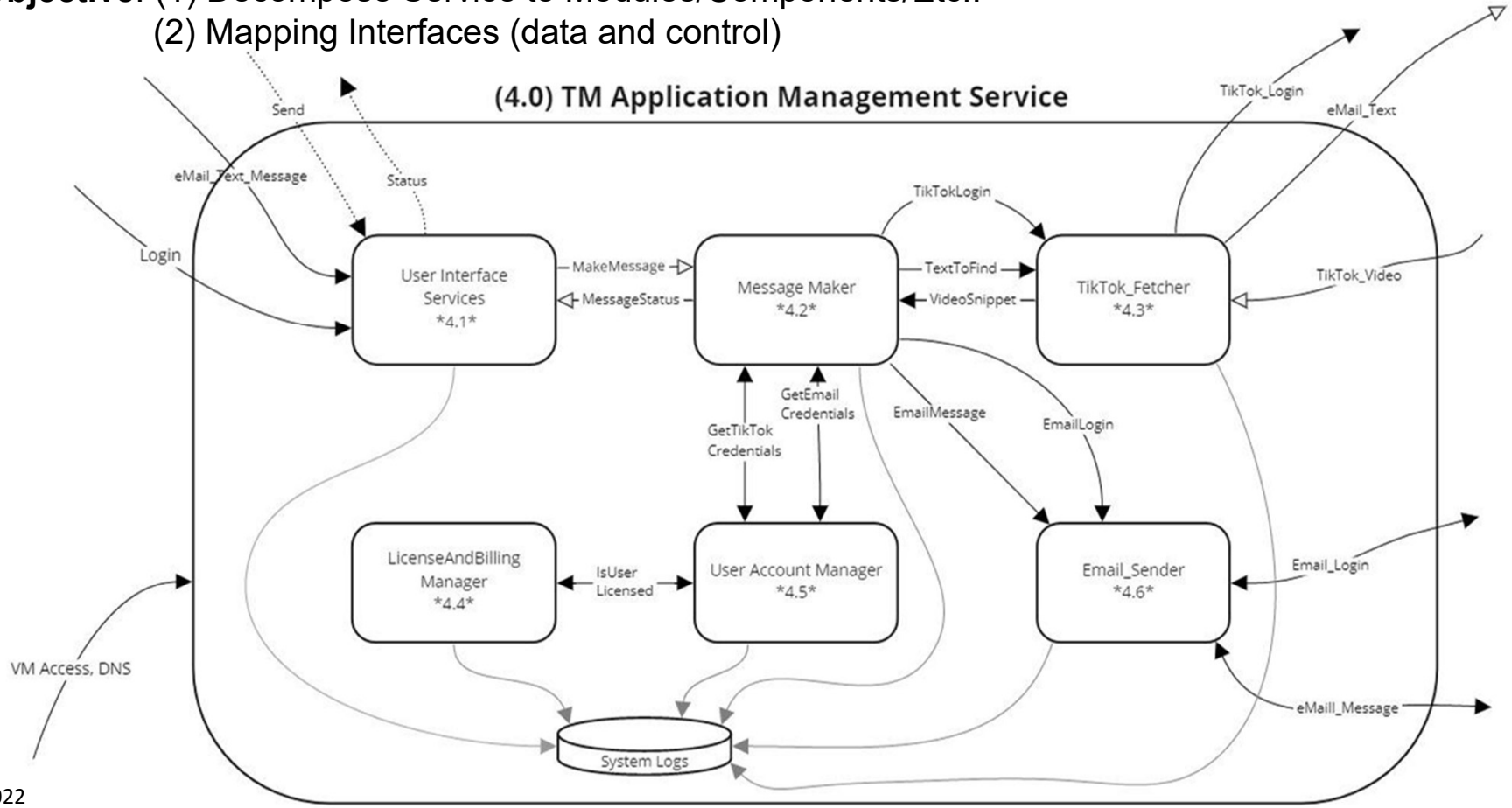
MODEL: TikyMail Application Diagram (Level 1)

Objective: (1) Identify Internal Services within our Application
(2) Map External Interfaces to Internal Services



MODEL: TikyMail Service Diagram (Level 2.x)

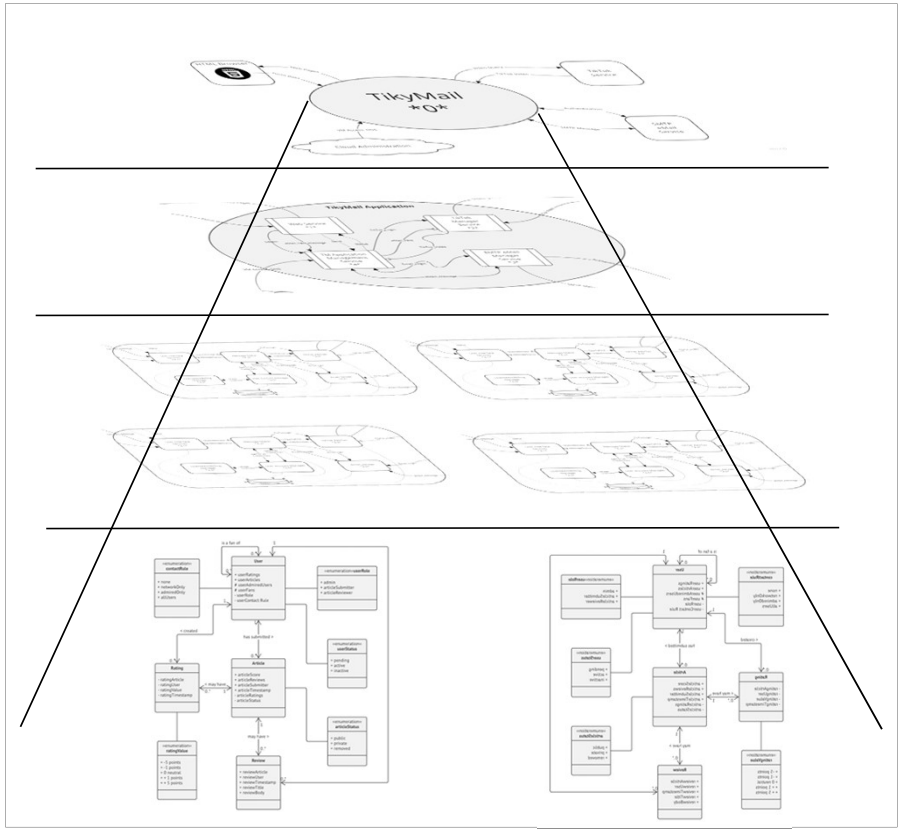
Objective: (1) Decompose Service to Modules/Components/Etc..
(2) Mapping Interfaces (data and control)



Design Model Layers

Defining different layers of abstraction aides in organization and focus

- Context Diagram (Level 0)
- Application Diagram (Level 1)
- Service Diagrams (Level 2)
- Package Diagrams
- Component Diagram
- Class Diagram (Level x)
- Data Object Diagram

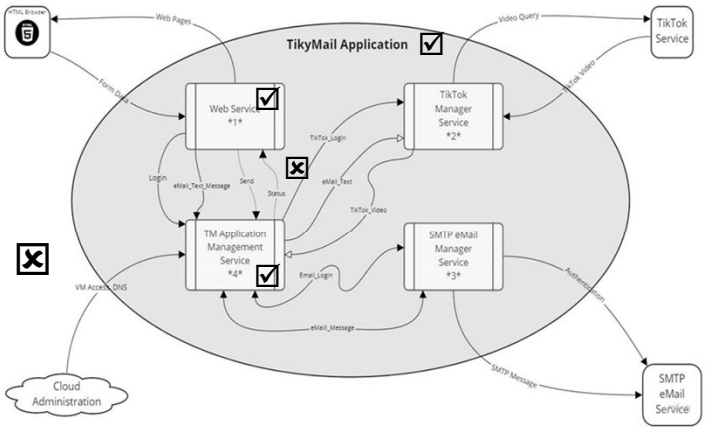


Validating Design Meets Requirements

- ***Quality Check:*** Design implements all requirements?
  Have we mapped our requirements into design?
- ***Quality Check:*** Is the design easy to follow/understand?
- ***Quality Check:*** Does the design follow good design patterns?

TikyMail (TM) Requirements:

- TM SHALL[1] be a cloud based Software as a Service Application ☑
- TM SHALL[22] support any HTML5 compliant browser ☑
- TM SHALL[43] support a "preview" of the outgoing video prior to sending ☒
- TM SHALL[102] ensure all user information is stored encrypted at rest ☑



Identifying Risks and Defects in The Design

Why Do Teams Fail While Testing Design?

Problem 1: In the Design Phase, testing the product when **there is no real Product** seems infeasible.

Problem 2: Teams **don't collaborate** as much on the design and architecture. They miss the opportunity to apply different experiences and skillsets to improve the design.

Problem 3: Not clear **how to test or improve** a design or architecture? What makes a good design is not easy to define.

Addressing Defects and Risks in Design

- A ***Defect*** in simple terms seems to be a problem with an object with what it is expected to do.
  Ex: If an umbrella can not protect you from rain, it is defected.

  In terms of software, a Defect is an error which causes incorrect or unexpected results from a software program which does not meet actual requirements.

  Quality checks that fail validation uncovers defects.

- **Risk** is an expectation of loss; a potential problem that may or may not occur in the future. Risks are all about uncertainty: **not what will happen, but what might happen.**

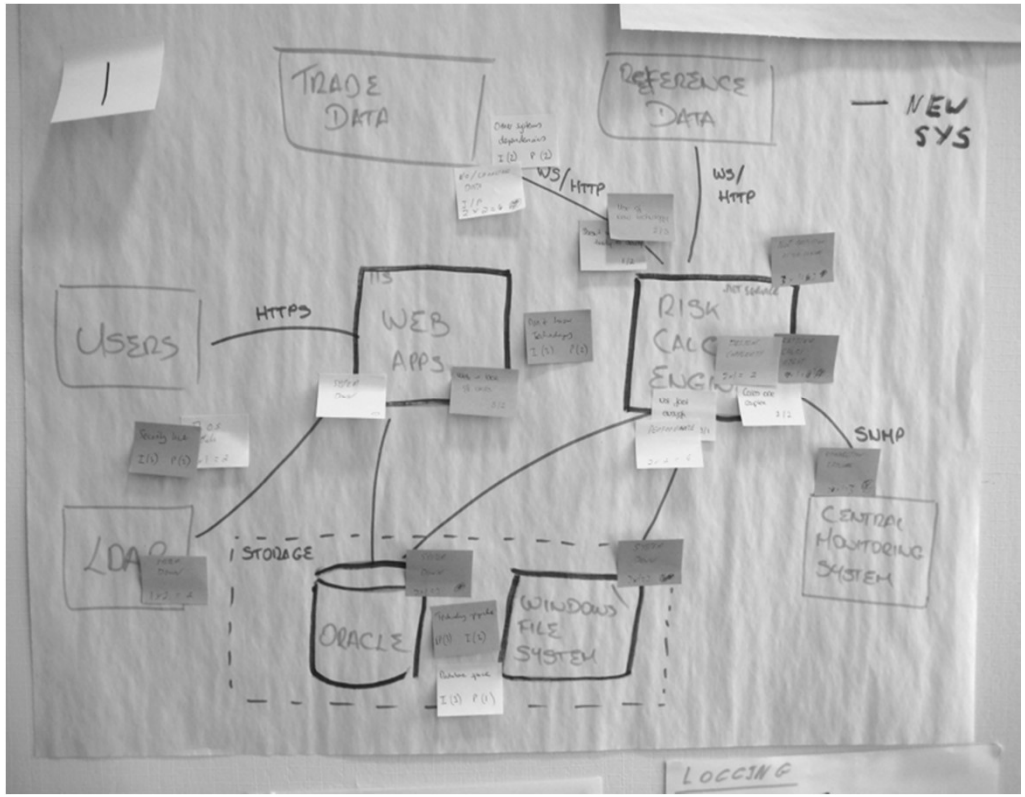
  Every functional/non-functional system has risks associated with it. Even Covid-19 vaccines have risks associated. The sooner we identify and take preventive actions, the better.

  Risks can be associated with People, Technology, Process.

Using a software model with a collaborative brainstorming activity to focus on identifying defects and risks, the team can significantly improve the quality of the design and thus, implement less defects.

Risk Storming – A Collaborative Approach

- Risk-storming is a quick technique that provides a collaborative way to identify and visualize risks.
- Collaboration during risk storming is the essence as people working on building the actual system to the people managing are great candidates who can identify the risks better. Including more people would remove the biases associated with the skill, perspectives.
- The whole team architects, developers, testers, project managers, operational staff, etc. can take part in this.



Risk Storming Process

1. **Draw architecture diagrams**

Modelling technique like c4 can be used to create an architecture diagram.

2. **Identify the risks individually**

Ask People to identify risks and summarise each risks.

3. **Converge risks on diagrams**

Relate Each risks to the associated component in architecture diagram.

4. **Review & summarise risks**

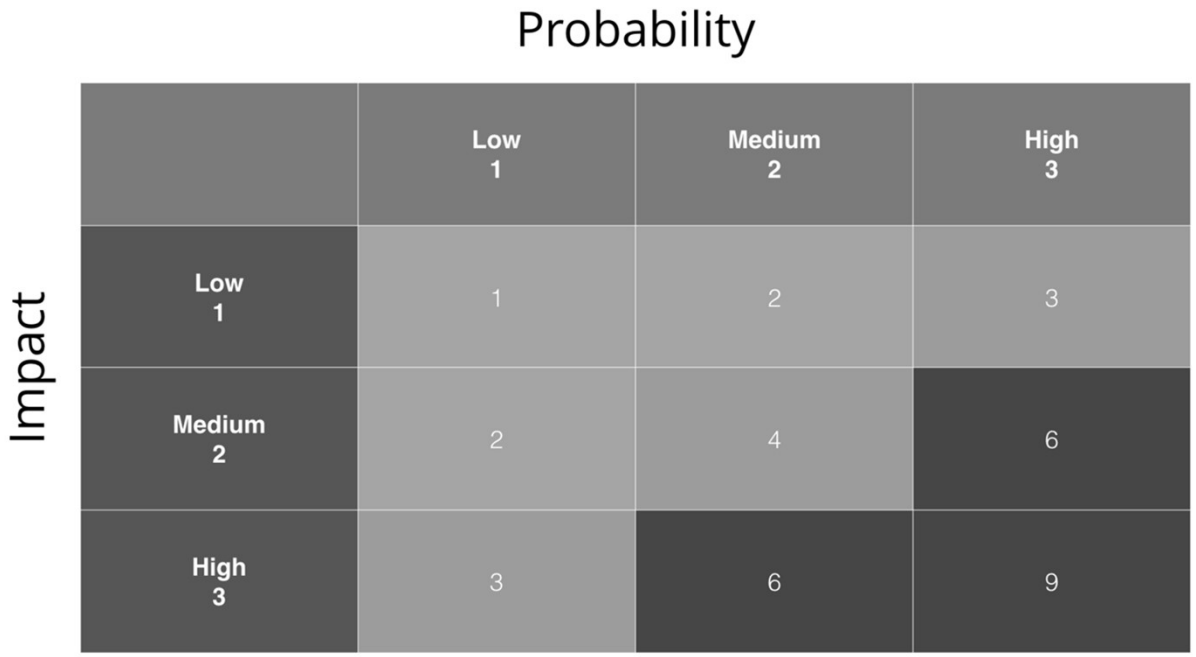
Discuss and summarise outputs to resolve disagreements and remove less probable risks.

Quantifying and Prioritizing Risk

Evaluate the probability of that risk happening against the negative impact of it happening.

1. **Probability**: How likely is it that the risk will happen?
2. **Impact**: What is the negative impact if the risk does occur?

Prioritizing risks is then about ranking each risk after multiplying the numbers in the risk matrix together



Exercise: Risk Storming TikyMail

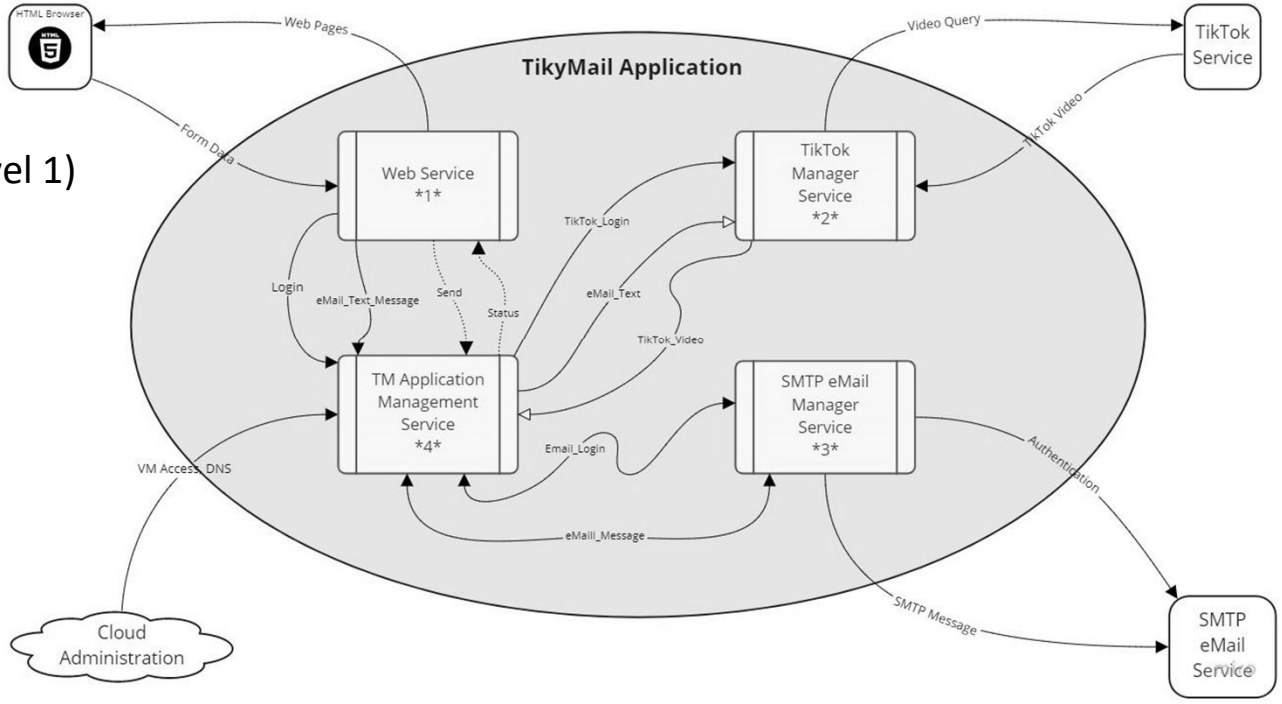
In Today's Exercise:
Risk Hunting in Application Design (Level 1)

Invited:

- Developers
- Quality Engineering
- Support Engineering
- Anyone else who likes donuts ☺

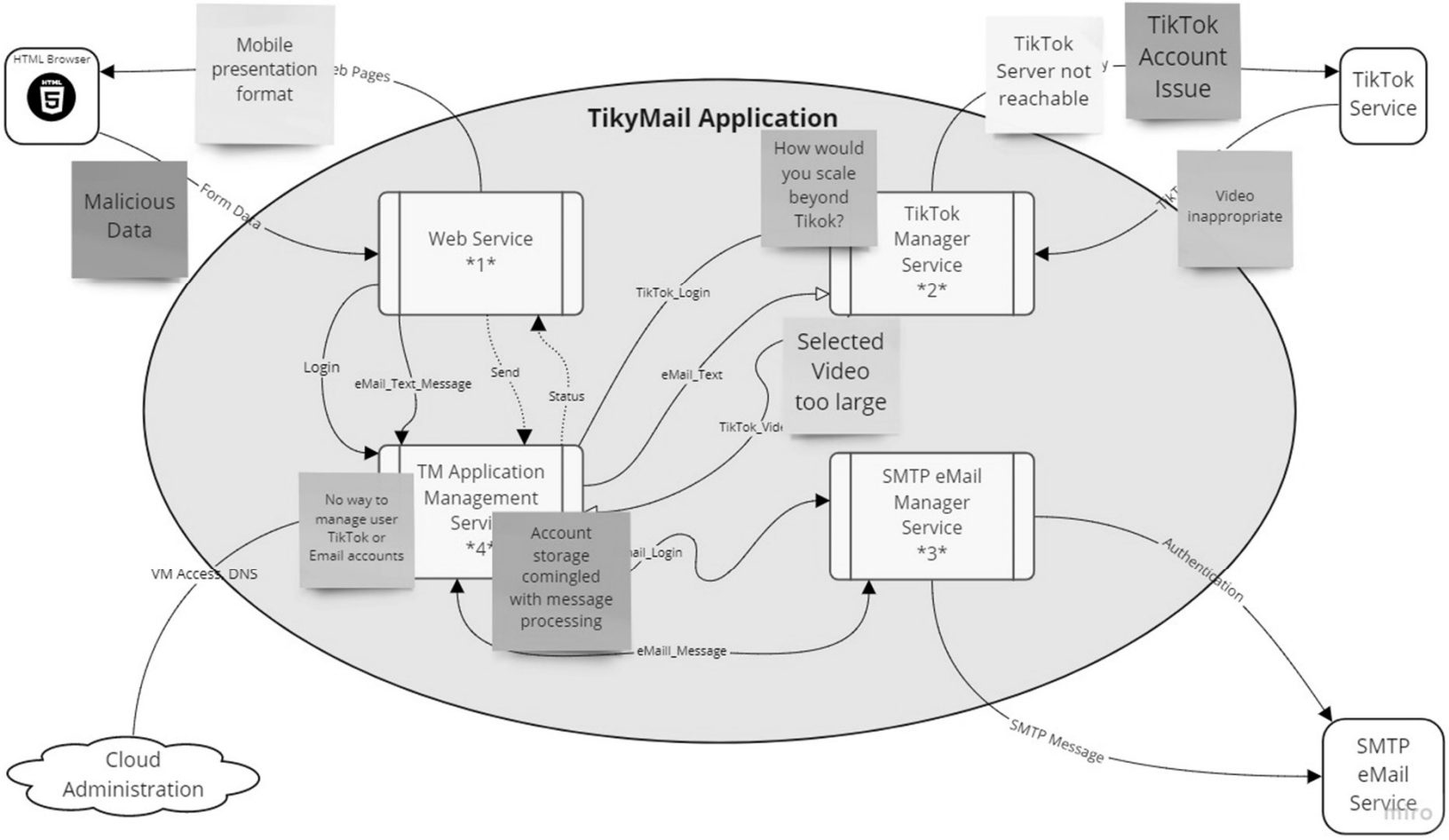
Goal:

Identify where this design fails?



https://miro.com/app/board/uXjVOYNHAy8=/?invite_link_id=725469525778

TikyMail Risk Storming Results



TikyMail Risk Storming Results

Low	**1**
Medium	**2**
High	**3**

Risk #	Description	Probability	Impact	Priority	Risk Mitigation Actions
R1	Malicious Data	2	3	6	A1: Regular Backup …..
R2	No Way to Manage User Data	1	2	2	A2: Add CRUD for user…
R3	Account Storage commingled with Processing	1	2	2	A3: …
R4	Returned Video Too Large	3	3	9	….
R5	External TikTok Server not reachable	2	3	6	…
R6	External Mail Server not reachable	1	3	3	…

Benefits of Risk Storming

- **Quickly identify risks in the proposed system architecture.**
- **Visualize the system by considering the level of risk**
- **Constrain risk identification to architectural concerns**
- **Provide a platform for all team members to elevate their concerns.**
- **By having Testers participate in the design Review**
  - Develop an understand the design and the architecture to be implemented
  - Insert their “tester” perspective early on in the development process
  - Better understand how to test the product with a higher likelihood of finding a hidden bug/defect.

Advanced Risk Mitigation

Examples Applying Quality Tools

Risk Mitigation on Steroids

***Risk Identification* is a good first step.. but what next?**

How do you minimize/negate the Risk during development?

- **Ignore** – Won't happen here
- **Ad-hoc** mitigation – hope it gets addressed
- Use **structured** mitigation techniques

Two Effective Tools to use in a structured mitigation approach:

- **Cause-Effect Diagrams (Fishbone Diagram)**
- **Failure Modes and Effect Analysis (FMEA)**

Cause/Effect (Fishbone) Diagram

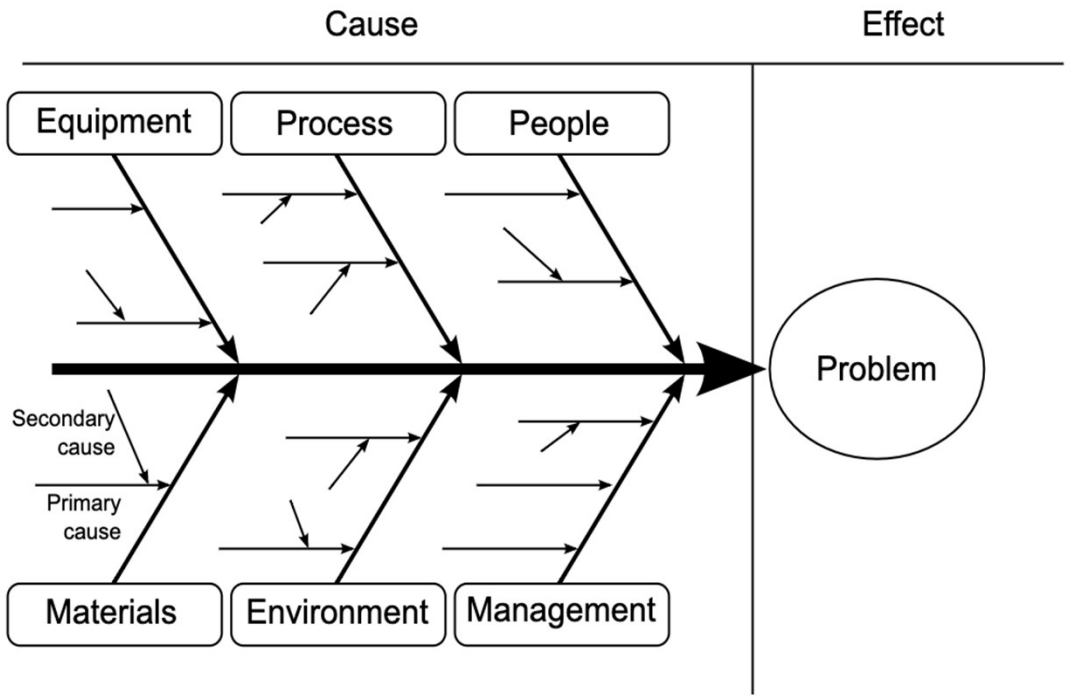
A graphical representation used to organize and display the interrelationships of various possible root causes of a problem. Possible causes of a real or potential defect or failure are organized in categories and subcategories in a horizontal tree-structure, with the (potential) defect or failure as the root node.

- ***ISTQB Glossary***

Fishbone diagram is a is a visualization **tool for categorizing the potential causes** of a problem.

Also known as Cause and Effect Diagram, Ishikawa Diagram, Herringbone Diagram, and Fishikawa Diagram

It combines brainstorming with Mindmap for **identifying all areas** that are contributing to a problem rather than only the most obvious ones.



Steps to draw Fishbone Diagram

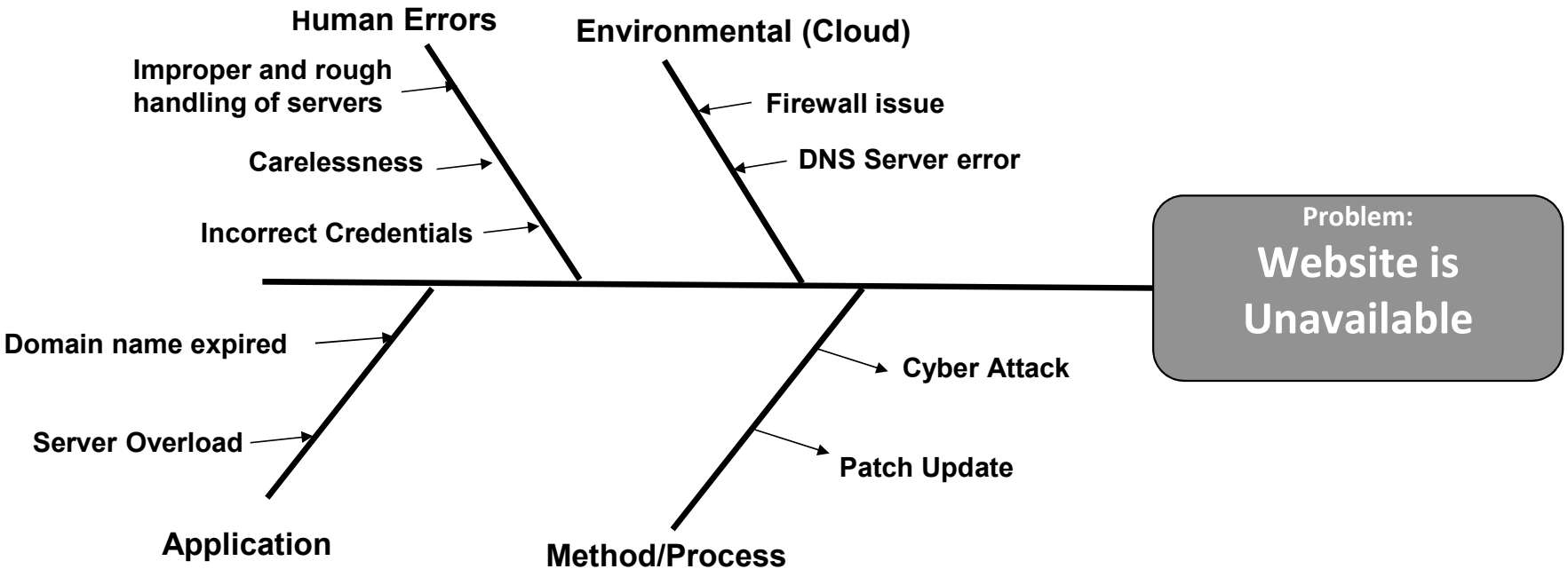
Agree on the problem statement

Agree on the major categories of causes of the problem. (Branch of the fish)

Brainstorm all the possible causes of the problem. Ask “Why does this happen?” to find sub-causes.

Continues to ask “Why?” and generate deeper levels of causes and continue organizing them

TikyMail Cause/Effect Diagram Example



Failure Mode and Effects Analysis (FMEA)

Failure Modes and Effects Analysis (FMEA) is a step-by-step approach for identifying all possible failures in a design. Failures are prioritized according to how serious their consequences are (Severity), how frequently they occur (Occurrence), and how easily they can be detected (Detection).

- American Society for Quality (ASQ)

Failure Modes means the ways, or modes, in which something might fail. Failures are any errors or defects, especially ones that affect the customer.

Effects analysis refers to studying the consequences of those failures.

Our goal is to prioritize the potential failures from most serious to least.

To prioritize, we'll use a formula to calculate a ***Risk Index.*** The Risk Index will be calculated by:

Risk Index = Severity * Occurrence * Detection

Where:

- Severity defines how serious the consequences would be if the risk occurred; [0.1 (least) ~ 1.0 (most)]
- Occurrence defines how frequently the risk would be encountered; and ; [0.1 (infrequent) ~ 1.0 (frequent)]
- Detection defines how easily the risk could be detected if it occurred [0.1 (easily detected) ~ 1.0 (difficult to detect)]

TikyMail FMEA Example
Application Design Risk

1. List Risks
2. Assign Severity; Occurrence; and Detection factors to calculate Risk
3. Sort Risks
4. Identify Actions to mitigate risks

Risk #	Description	S	O	D	Risk Score	Actions To Mitigate Risk
R1	Malicious Data Sent to TikyMail	1.0	0.3	0.3	0.09	
R2	No Way to Manage User Data	1.0	1.0	1.0	1.00	
R3	Account Storage comingled with Processing	0.8	1.0	0.8	0.64	
R4	Returned Video To Large	0.9	0.5	0.1	0.045	
R5	External TikTok Server not reachable	1.0	0.2	0.1	0.02	
R6	External Mail Server not reachable	1.0	0.2	0.1	0.02	

TikyMail FMEA Example
Application Design Risk Prioritized

1. List Risks
2. Assign Severity; Occurrence; and Detection factors to calculate Risk
3. Sort Risks
4. Identify Actions to mitigate risks

Risk #	Description	S	O	D	Risk Score	Actions To Mitigate Risk
R2	No Way to Manage User Data	1.0	1.0	1.0	1.00	A1: Add CRUD for User Data
R3	Account Storage comingled with Processing	0.8	1.0	0.8	0.64	A2: Factor design to separa..
R1	Malicious Data Sent to TikyMail	1.0	0.3	0.3	0.09	A3: ….
R4	Returned Video To Large	0.9	0.5	0.1	0.045	
R5	External TikTok Server not reachable	1.0	0.2	0.1	0.02	
R6	External Mail Server not reachable	1.0	0.2	0.1	0.02	

In Summary

- ***Don't leave your design and architecture to chance – Engineer IT!***
  - Model it to get a better understanding of it
  - Make sure it addresses all of your requirements
  - Future proof it – Change WILL happen

- ***Use team collaboration to review your design model***
  - Understand where are the risks in the proposed design
  - Where will it fail – it will
  - Have focused reviews on Technology; Security; Maintainability; Performance

- ***Address your risks in Priority Order***
  - You most like won't have resources to address them all
  - Go for the ones with the highest return on investment
  - Use structured techniques to problem solve and drive exploration

TikyMail – ProtoType v1

Thank You!

Questions??

Contacts:

Darin Kalashian : darin@dskquality.com

Neha Chopra : chopraneha2711@gmail.com

Helpful References

Modeling resources

- https://www.c4modeling.org
- https://structurizr.org/
- https://simonbrown.je/
- https://en.m.wikipedia.org/wiki/4%2B1_architectural_view_model
- https://modeling-languages.com/benefits-modeling-or-how-convince-your-project-manager/
- https://www.lucidchart.com/blog/types-of-UML-diagrams

RiskStorming: https://riskstorming.com

Tools:

- https://www.miro.com - White boarding application
- https://www.canva.com - Fishbone creation tool

Backup

Driving Quality In-Between Requirements and Testing

Presented by Darin Kalashian and Neha Chopra

Quality's typical role is to champion testing and validation at the end of the product development cycle, with the goal to confirm that the product meets defined requirements and use cases. As we strive to excel in our "push left" world, frequently, you'll find Quality Engineering actively in play, working with the team to ensure that clear and concise requirements are produced during story planning. Unfortunately, the span of development after planning and before testing, that being the Design and Architecture phase, typically has quality absent. In practice, quality will not be achieved if the product was not designed right from the start, no matter how good the testing efforts are.

In this presentation, we'll explore how to advance the overall product quality from within the product development path, identifying a class of impactful issues that may elude the best tests. We will explore three patterns that can be used to improve product architecture and design quality, proactively. These techniques will empower the development team to take ownership of the product quality, in ways that testing would have a hard time finding (or at least in a cost efficient manner).

At the conclusion of this talk, participants will know how to:

- Trace requirements into design architecture
- Facilitate team collaboration to identify risks in the architecture
- Rank and prioritize identified risks to build a plan to proactively address design weaknesses (long before the code has even been written).

Author Biography's:

Darin Kalashian is a software engineering and quality evangelist, currently working at VMware as a Senior Software Engineering Manager. Darin has always believed Quality is a team sport and has spent most of his career to insert quality engineering principles throughout the development process. While testing surely provides a safety net for release, inserting quality habits from day one drives efficiency and effectiveness improvements that testing has a hard time competing with.

Neha Chopra is a software quality advocate working on her Thesis while working at Red Hat as a Software Automation Engineer. For Neha, the irony of being a successful tester is to believe in failure. Neha is motivated to work with her teams to effectively apply quality techniques where they will be most impactful. Understanding that test automation has its limits, Neha works to explore ways to insert human logic and heuristics to complement testing and deliver quality.