The functionality behind non-functional requirements

Picture source: http://www.toolagen.com/wp-content/uploads/2014/08/requirement.png

האיגוד הישראלי להנדסת מערכות
INCOSE IL
Israeli Society For Systems Engineering

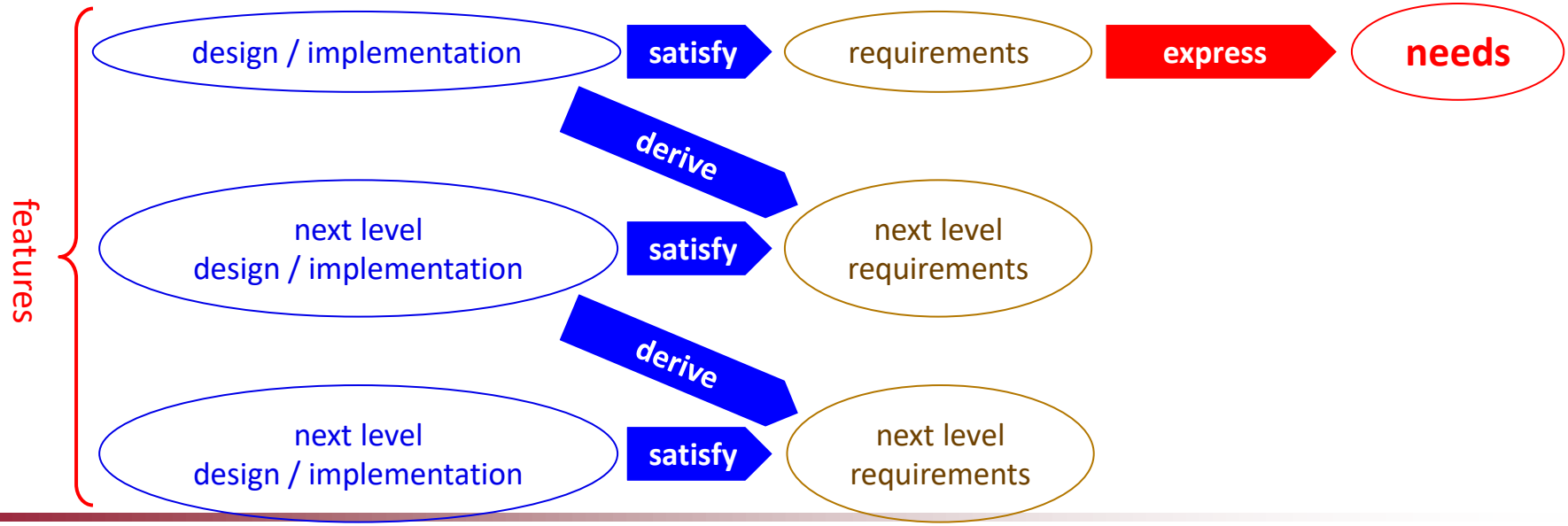
amir@amirtomer.com

Prof. Amir Tomer – short bio

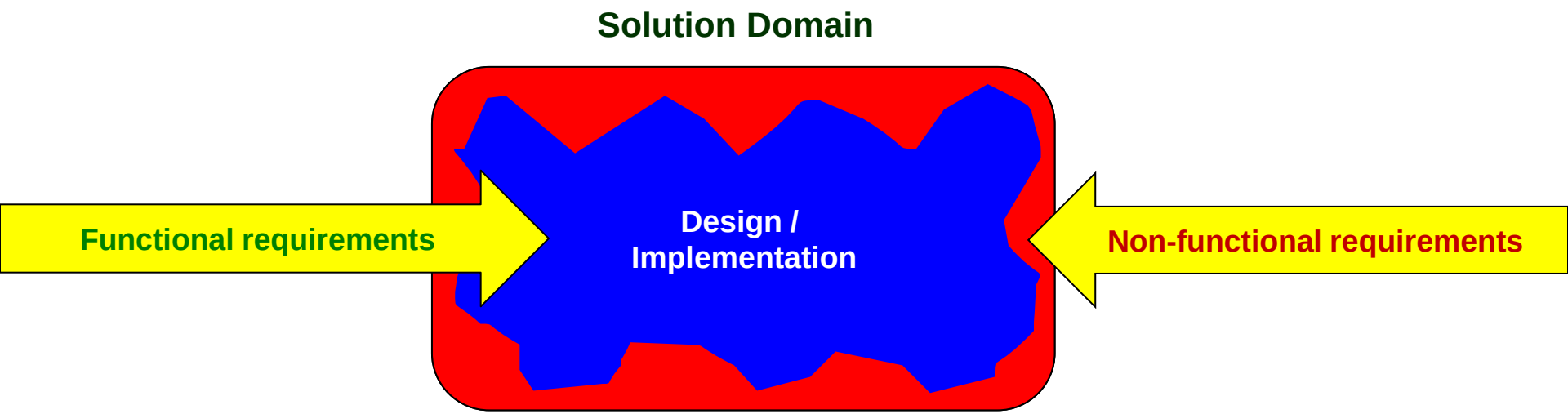
- Education
  - B.Sc. & M.Sc. in Computer Science – Technion, Israel
  - Ph.D. in Computing – Imperial College, London
- Industrial experience
  - 1982-2009: Rafael, Advanced Defense System
    - 1999-2006: Director of Software and Systems Engineering Processes
- Academic experience
  - 2009-now: Head of Software Engineering Department, Kinneret College on the Sea of Galilee, Jordan Valley, Israel
  - 1994-now: Senior lecturer, Technion, Haifa, Israel
- Research interests
  - Software-intensive Systems Engineering
  - Model-Based systems/Software Engineering
  - Software reuse

Requirements vs. design

- Requirement
  - A statement that translates or expresses a **need** and its associated constraints and conditions [ISO/IEC/IEEE 12207:2017]
- Design / Implementation
  - A (technical) solution by which one or more requirements may be satisfied
    - The design at every development level is the basis for that level's requirements



Functional and non-functional requirements



Functional Requirements

- Define the solution ***contents***
  - **What** is the system required to do?

Quality Attributes (QAs / Constraints)

(part of the non-functional requirements)

- Define the solution quality
  - **How well** should the system carry out its functional requirements?
    - Performance
    - Availability
    - Security
    - Interoperability
    - …

The Conflict

- System specifications concentrate mainly on **functional requirements**
  - They describe the purpose of the system
  - They are conceptual (implementation-free)
  - They remain valid even when quality attributes change

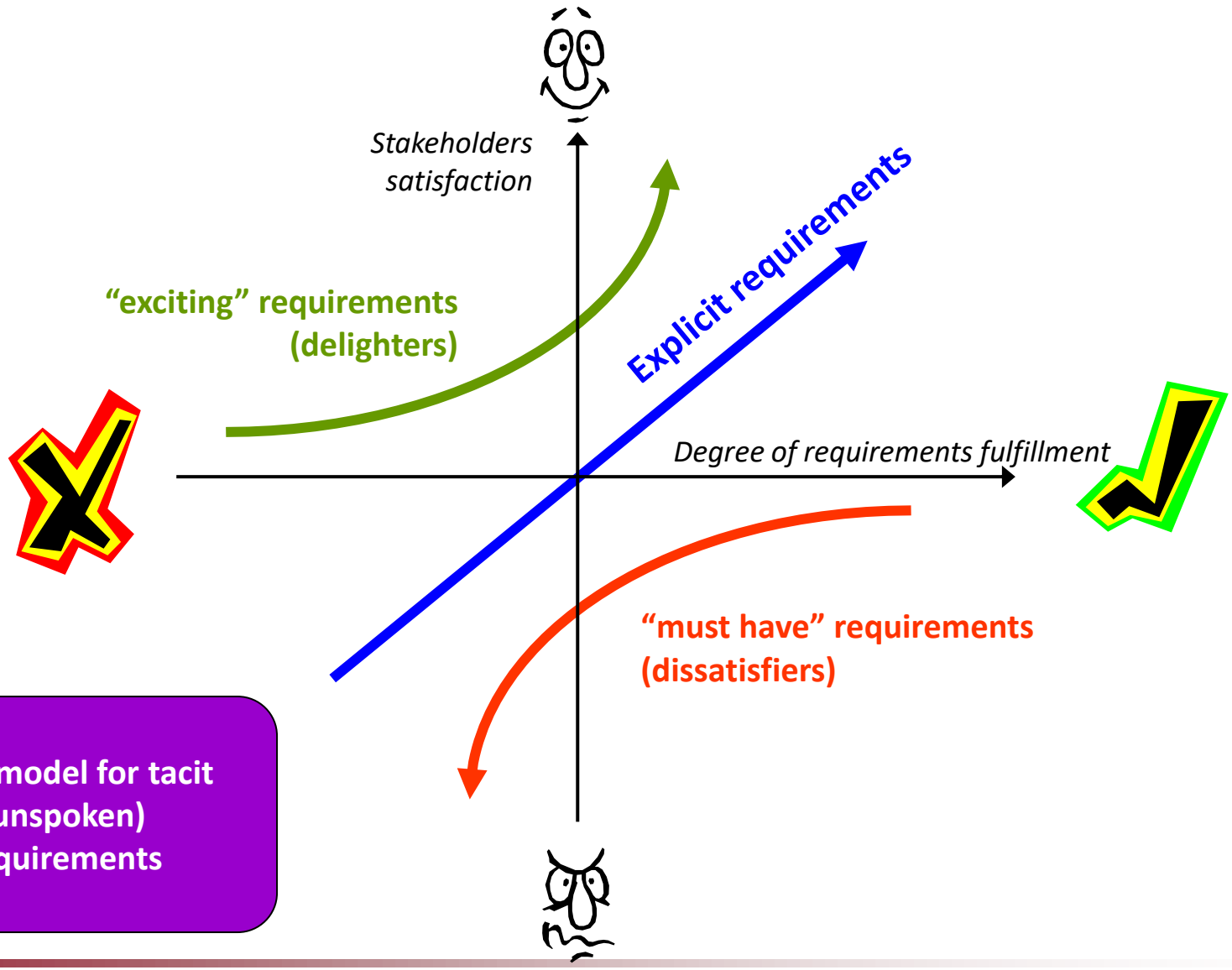
Whereas…

- The system architecture is driven mainly by **QAs**, e.g.
  - Using redundancy to improve availability
  - Using load-balancing to improve performance
  - Using fire walls to improve security
  - …

These do not affect the basic functionality,

but may impose additional functionality to provide quality

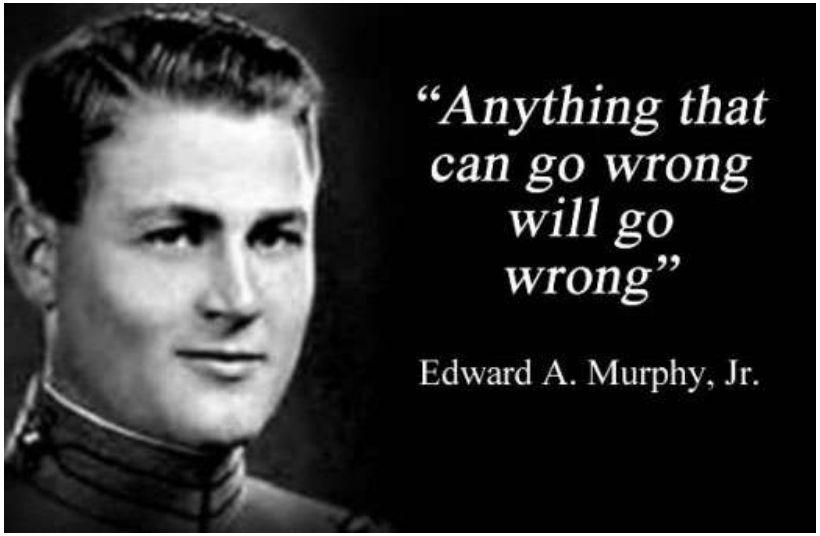
Many customer's QAs are unspoken ("must have")



Kano model for tacit (unspoken) requirements

Who is this person?

Edward Murphy – the speaker of the unspoken QAs



- Murphy is the one to say:
  - "The medical device will electrify the patient during treatment"
  - "The data server will crash at the most critical moment"
  - "A hacker will break into the system causing Denial of Service"
- So why shouldn't we hire him on our project?

Asking “how-well” (Murphy’s) questions

- The functional requirements may be investigated by asking questions related to various QAs
- For example,
  - A car has a BREAKING function to stop it from going
  - Questions that may be asked about this function
    - How long would it take to come to a full stop? [performance]
    - What is the probability that the break will not work? [reliability]
    - Are there alternative ways to stop the car [availability]
    - Is the breaking operation dangerous to the people in the car? [safety]
    - Can a hacker intervene wirelessly with the breaking function? [security]
    - and more ...

Quality Scenarios

- GPS's availability [i.e. to know your current location] is a key QA in car navigation systems, but...
  - Does it really matter in these cases?
    - Planning a future trip
    - Setting-up the system
    - Searching for restaurants near your destination
    - etc.
- QAs usually "dock their heads" when something goes wrong while the system is functioning, for example
  - While **navigating to destination** the **GPS signal is lost**
- Now, two things may happen
  - Either the GPS signal will resume, and we will **get to destination**,
  - Or not, and we will **not get to destination**

goal

fault

success

failure

Functional reaction to QAs (non-functional req's)

During a functional scenario

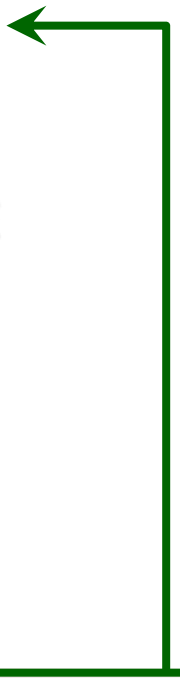
if a fault occurs

the system should **do something**

to prevent the **FAULT**

from becoming a **FAILURE**

FAULT = violation of a QA
FAILURE = not reaching the goal



The functionality behind non-functional requirements

A Case-study: A car navigation system

- Consider the following Main Success Scenario (MSS) of the "Navigate Along Route" use-case
- Quality attributes may be discovered in the functional scenarios by applying the "how well?" questions to the functions in the scenario

Is the driver aware? (usability)

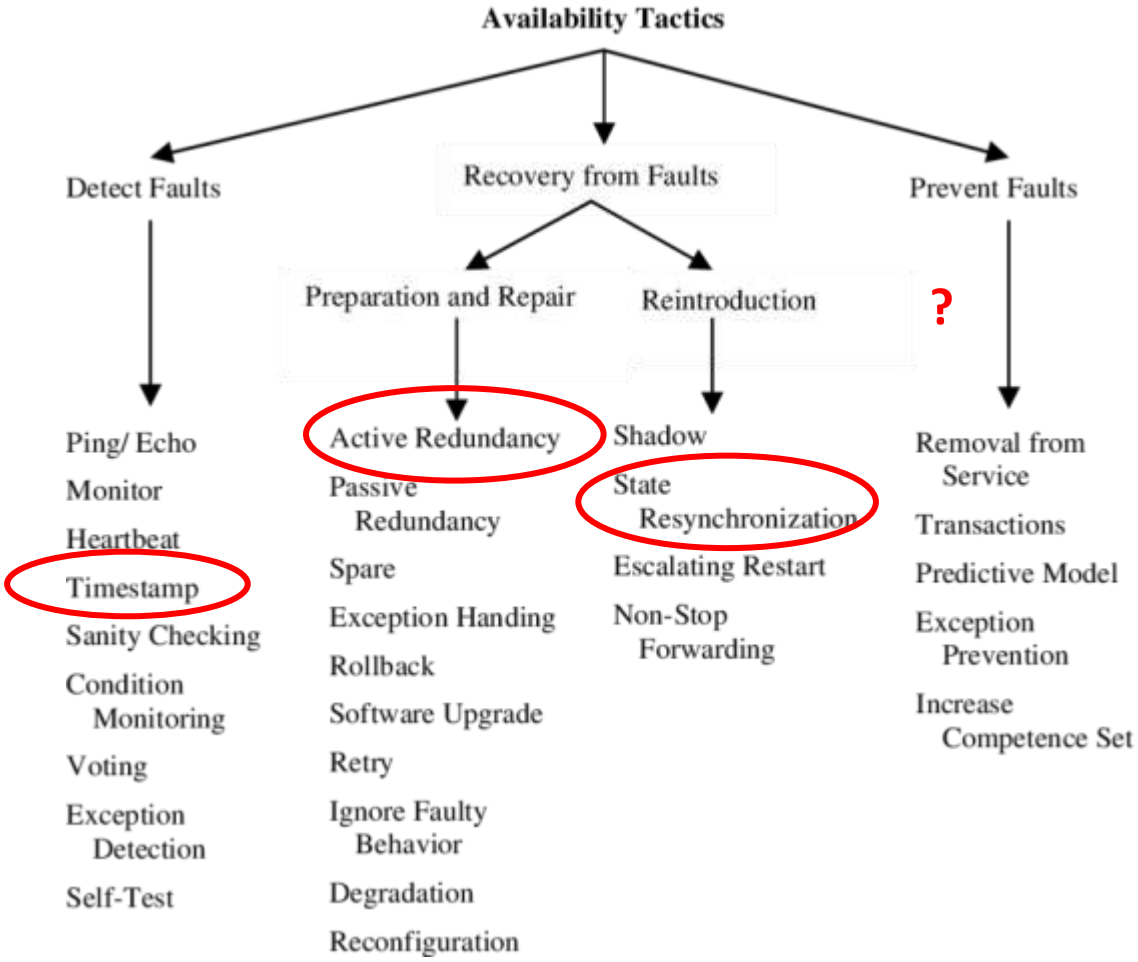
Trigger	**The driver** presses the "Start Navigation" button
Main Success Scenario (MSS)	**1.** **The system** enters "Navigate" Mode **2.** **The system** retrieves the pre-defined route **3.** **The system** retrieves a relevant map area 4. If the locally stored map data is missing – **the system** downloads map data from the server **5.** **The system** displays the map and route **6.** **The system** gets the current location from the GPS 7. If the location is off route – **the system** re-calculates the route (See "Calculate Route" UC) 8. Back to step 4

Does the server respond within TBD seconds? (availability)

Is the location precise? (performance)

Choosing tactics to deal with the server availability problem

- A hierarchy of availability tactics [BC&K, 2012]



Using the tactics in the quality scenario

- With “redundancy” in mind, the following functionality may be added to the scenario
  - “suggest an alternative server” is a new function

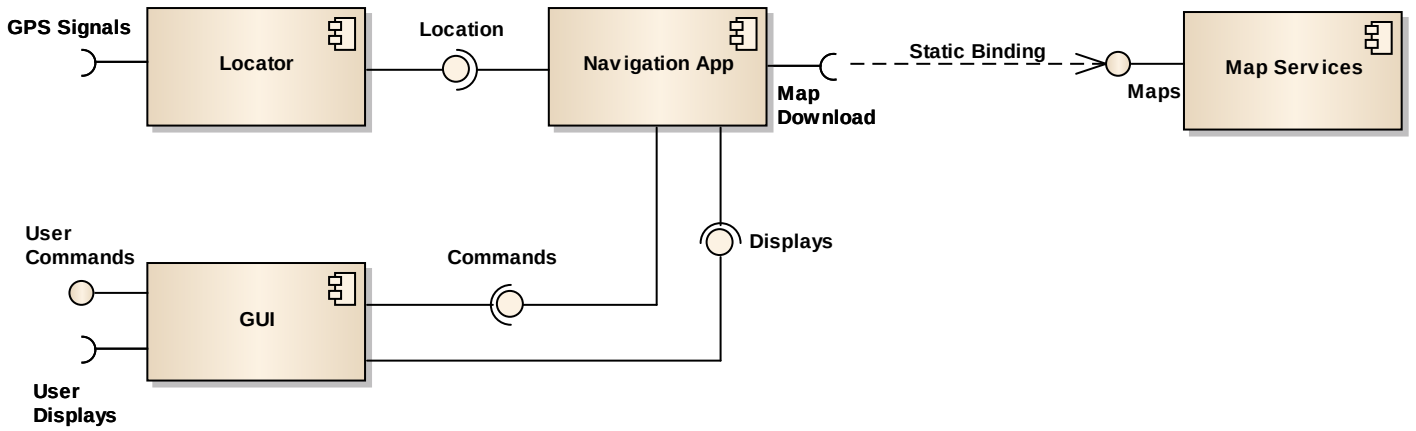
Trigger	**The driver** presses the “Start Navigation” button
Main Success Scenario (MSS)	**1.** **The system** enters “Navigate” Mode **2.** **The system** retrieves the pre-defined route (See “Define Trip” UC) **3.** **The system** retrieves a relevant map area 4. If the locally stored map data is missing – **the system** downloads map data from the server **5.** **The system** displays the map and route **6.** **The system** gets the current location from the GPS 7. If the location is off route – **the system** re-calculates the route (See “Calculate Route” UC) 8. Back to step 4
Branch A	**Alternative** in step 4 of MSS: The server did not respond within TBD seconds 4A1. **The system** suggests an alternative server 4A2. Back to step 4

A new function!

- But this is not the end of the story yet...

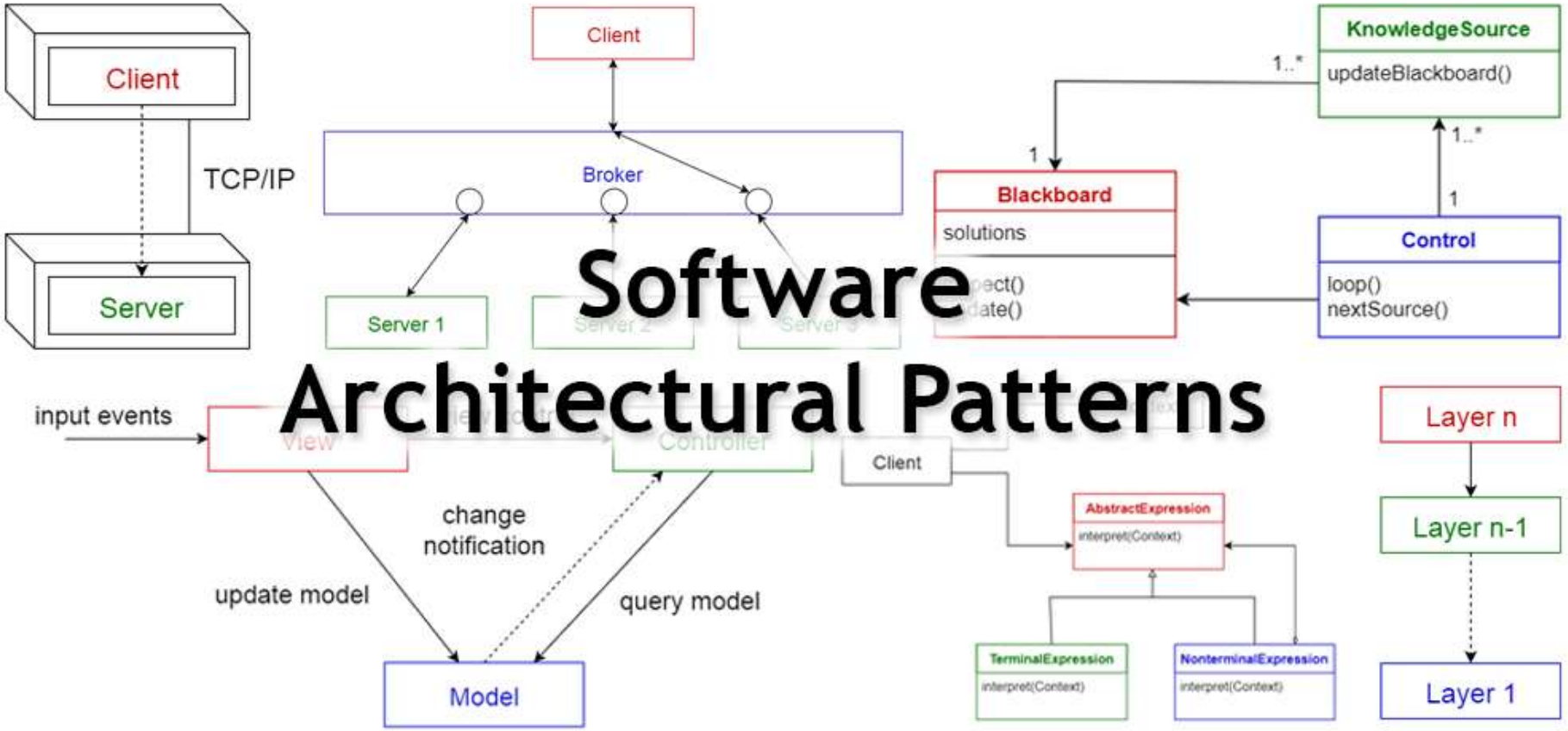
Architectural Impacts

- Supposing the following functional architecture of the system before the change:



- How can the new function be addressed?

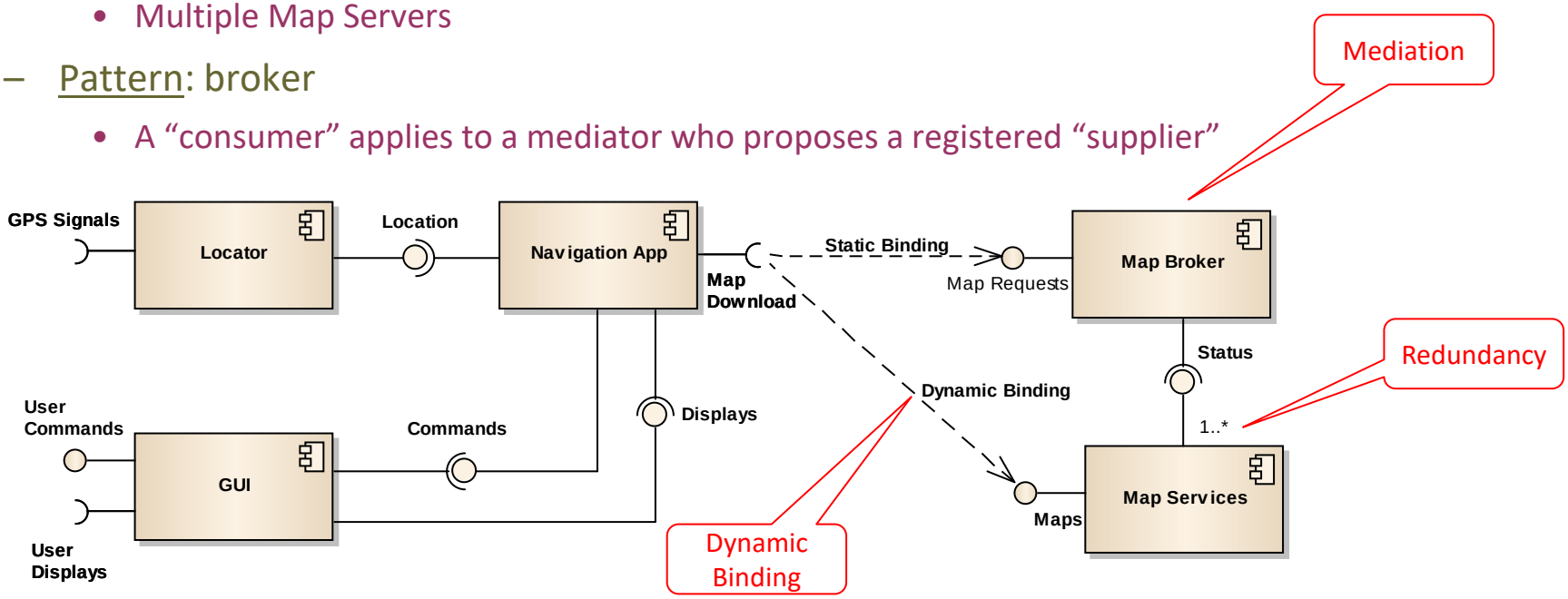
Choosing an architectural pattern



https://miro.medium.com/max/1400/1*M22DR3WPqbWXWidYIq2GwA.png

An architectural solution

- The following modified architecture applies 3 architecture decisions
  - Strategy: dynamic binding
    - The binding between a Navigation App and a Map Server may be changed during runtime
  - Tactics: redundancy
    - Multiple Map Servers
  - Pattern: broker
    - A “consumer” applies to a mediator who proposes a registered “supplier”



- However...

Functional Impact (examples)

- Each of the architectural decisions implies additional software:
  - Dynamic binding
    - Binding / unbinding times and durations
    - Binding multiplicity
  - Server redundancy
    - backup/synchronization
    - hand-off
  - Mediator/Broker
    - Register / unregister (service providers)
    - Match and connect
    - load-balancing

Conclusion

- At the end of a system's design, the system will have two kinds of functionality
  - The functionality imposed by the customer
    - "Mission functionality"
  - The functionality imposed by the engineers
    - "Quality functionality"

- Issues to be considered:
  - Workplan (WBS)
  - Schedule
  - Budget
  - Conflict of interest
  - and others...

A word on non-runtime quality attributes

- Non-runtime quality attributes are those referring to the development process
  - Which might be described as a “development scenario”
- Consider the following example:
  - Suppose a routine develop-deliver process in 6-months cycles
  - An event such as “we need to deliver more often” may be considered as a **fault** in the process which might cause a **failure** (the version is delivered too late)
- Cause of problem: “Big monolith” architecture
  - Process solution: DevOps
  - Structure solution: μServices

Resources

- Tomer, A., *Functional Angels and Quality Devils: Incorporating Quality Scenarios into Functional Scenarios for Software-intensive System Architecture*, International Journal of Computer and Software Engineering, Vol.4, No. 144, April 2019. [Tomer, 2019]
  - DOI: https://doi.org/10.15344/2456-4451/2019/144
- Bass, L., Clements, P. and Kazman, R., Software Architecture in Practice, 3rd Edition, Addison-Wesley, 2012 [BC&K, 2012]

More to come...

- Other tutorials by this author
  - Webinars
    - Making Use Cases great again (~1 hour)
    - Relay Race: The shared challenge of systems engineers and software architects (~1 hour)
  - Workshops
    - Software requirements and use cases (1 day)
    - Software engineering for systems engineers (2 days)
    - Software-intensive systems architecture (3 days)
- For details go to https://incoseil.org

Thank you for listening