Relay Race:
The Shared Challenge of Systems- and Software-Engineers



Picture source: https://pxhere.com/en/photo/360533

האיגוד הישראלי להנדסת מערכות
INCOSE IL
Israeli Society For Systems Engineering

© prof. Amir Tomer

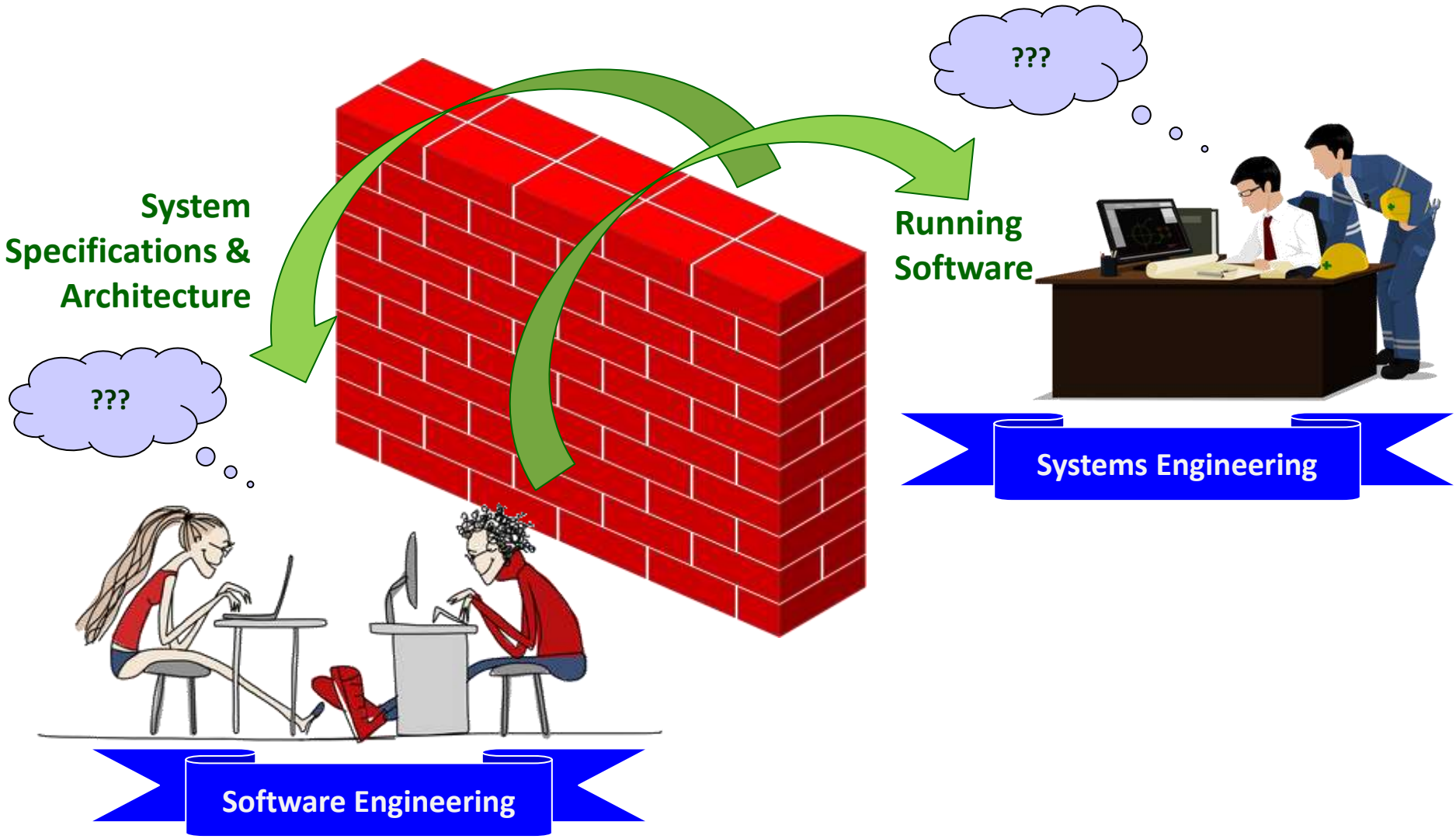
amir@amirtomer.com

Prof. Amir Tomer – short bio

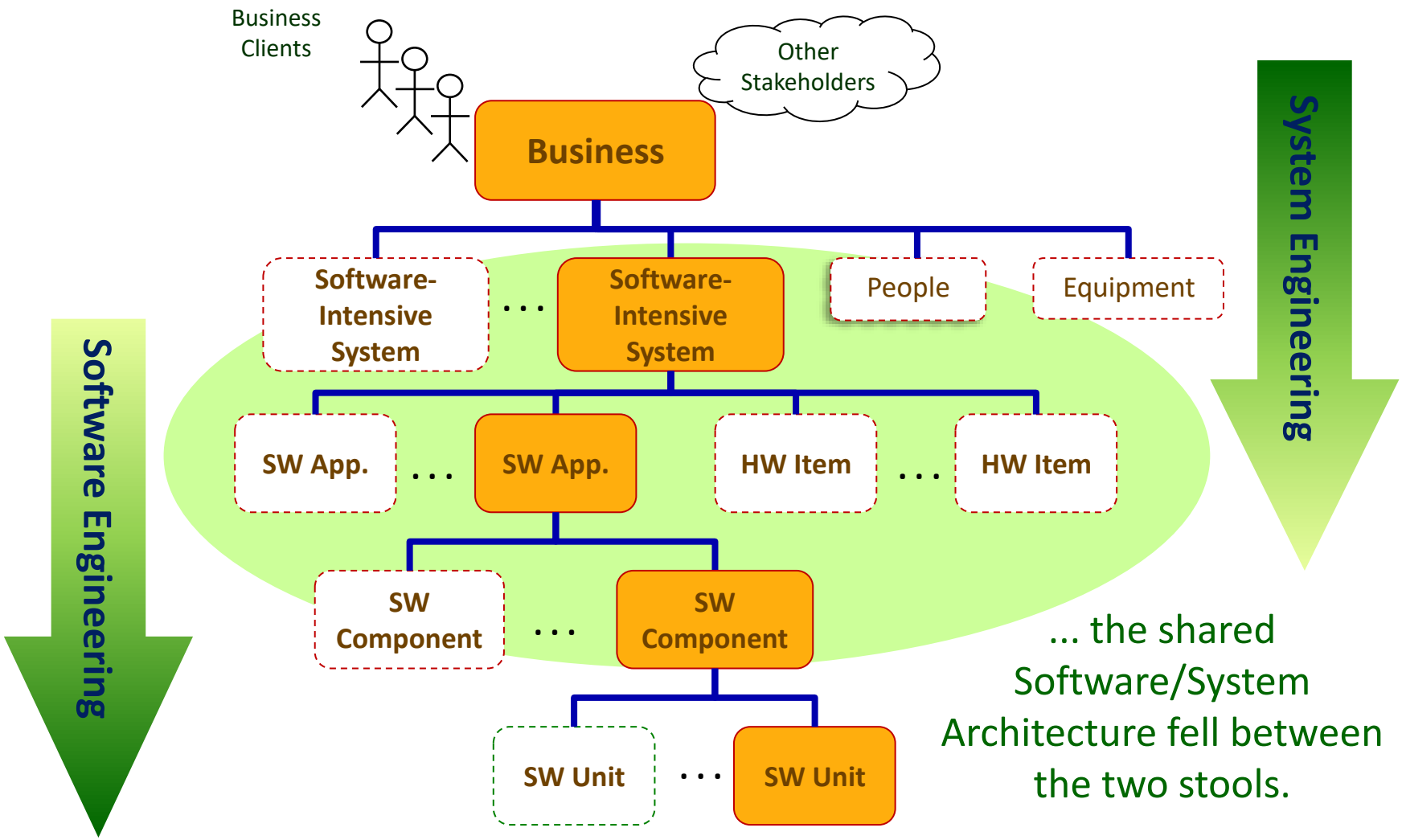
- Education
  - B.Sc. & M.Sc. in Computer Science – Technion, Israel
  - Ph.D. in Computing – Imperial College, London
- Industrial experience
  - 1982-2009: Rafael, Advanced Defense System
    - 1999-2006: Director of Software and Systems Engineering Processes
- Academic experience
  - 2009-now: Head 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



Once upon a Time...



... Although They Worked on Similar Systems...



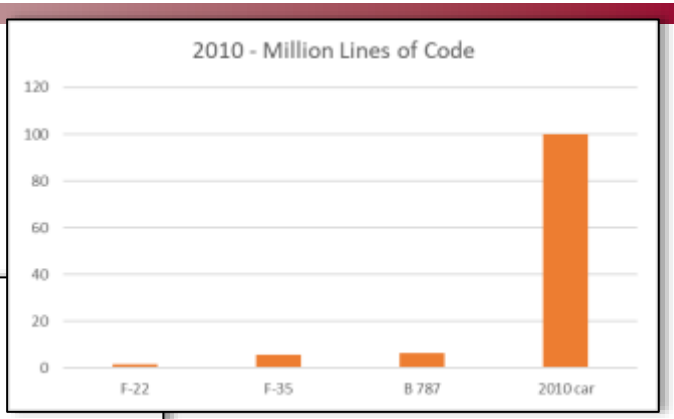
But Things have changed for Systems Engineers...

- New system capabilities
  - Adaptability
  - Intelligent behaviour
  - Interoperability
  - Relying upon multiple and large sources of data
  - Standard/commercial platforms (smartphone)
- New system architectures
  - Internet of Things (IoT)
  - Multiple sensors
  - Systems-of-systems
  - Human-Systems Integration (HSI)

… As well as for Software Engineers…

- New software techniques
  - Machine learning / deep learning
  - Big data
  - Standard GUI
  - NoSQL Databases
- New development Processes
  - Open-source
  - Agile software development
  - DevOps
  - Architecture frameworks

… And Systems Became More and More Software-intensive



IEEE SPECTRUM

FEATURE

This Car Runs on Code

It takes dozens of microprocessors running 100 million lines of code to get a premium car out of the driveway, and this software is only going to get more complex



Image: Daimler

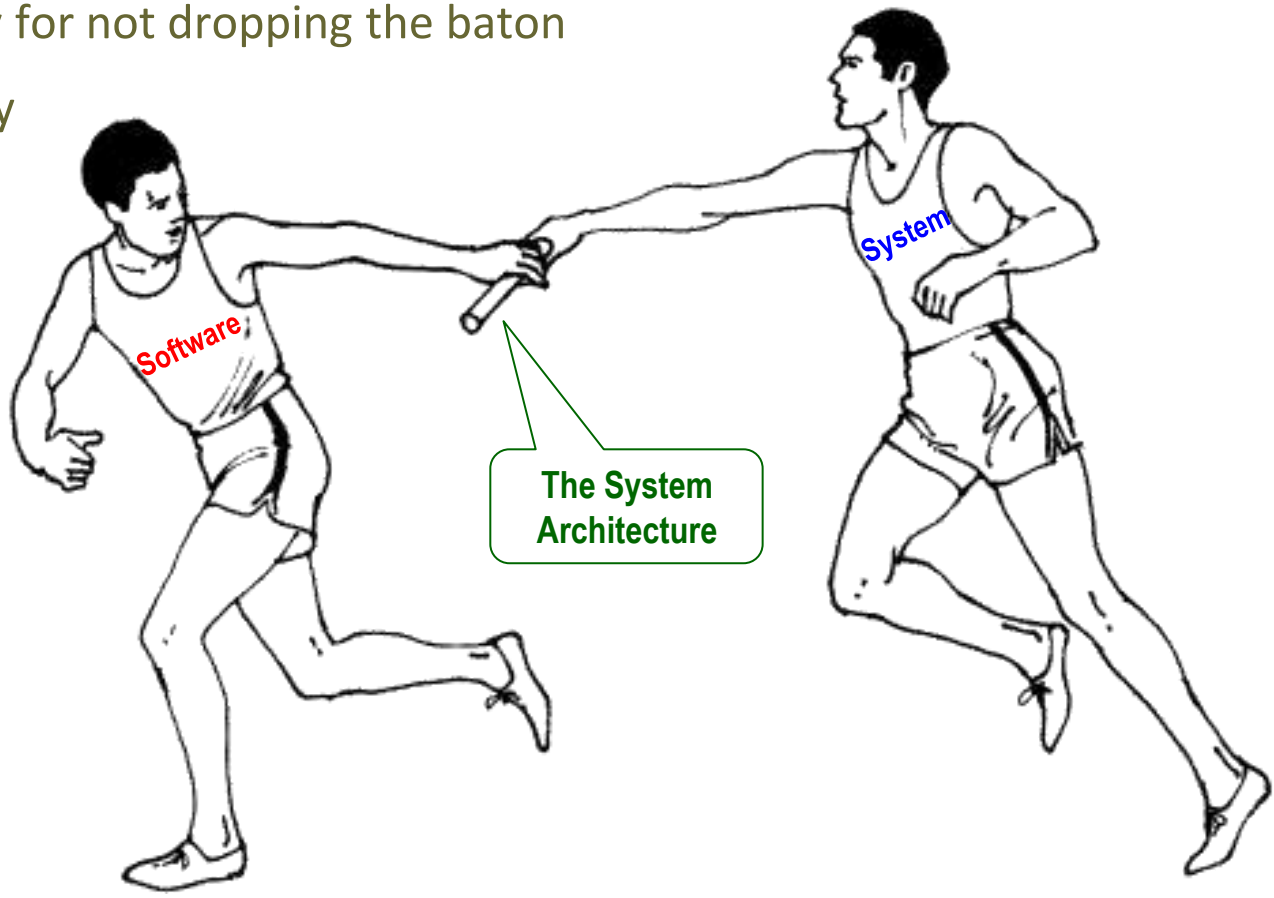
BY ROBERT N. CHARETTE // FEBRUARY 2009

The avionics system in the F-22 Raptor, the current U.S. Air Force frontline jet fighter, consists of about 1.7 million lines of software code. The F-35 Joint Strike Fighter, scheduled to become operational in 2010, will require about 5.7 million lines of code to operate its onboard systems. And Boeing's new 787 Dreamliner, scheduled to be delivered to customers in 2010, requires about 6.5 million lines of software code to operate its avionics and onboard support

… So They Realized they should Start Working Together on the Architecture in a Relay-Race style!

The Relay Race Principles:

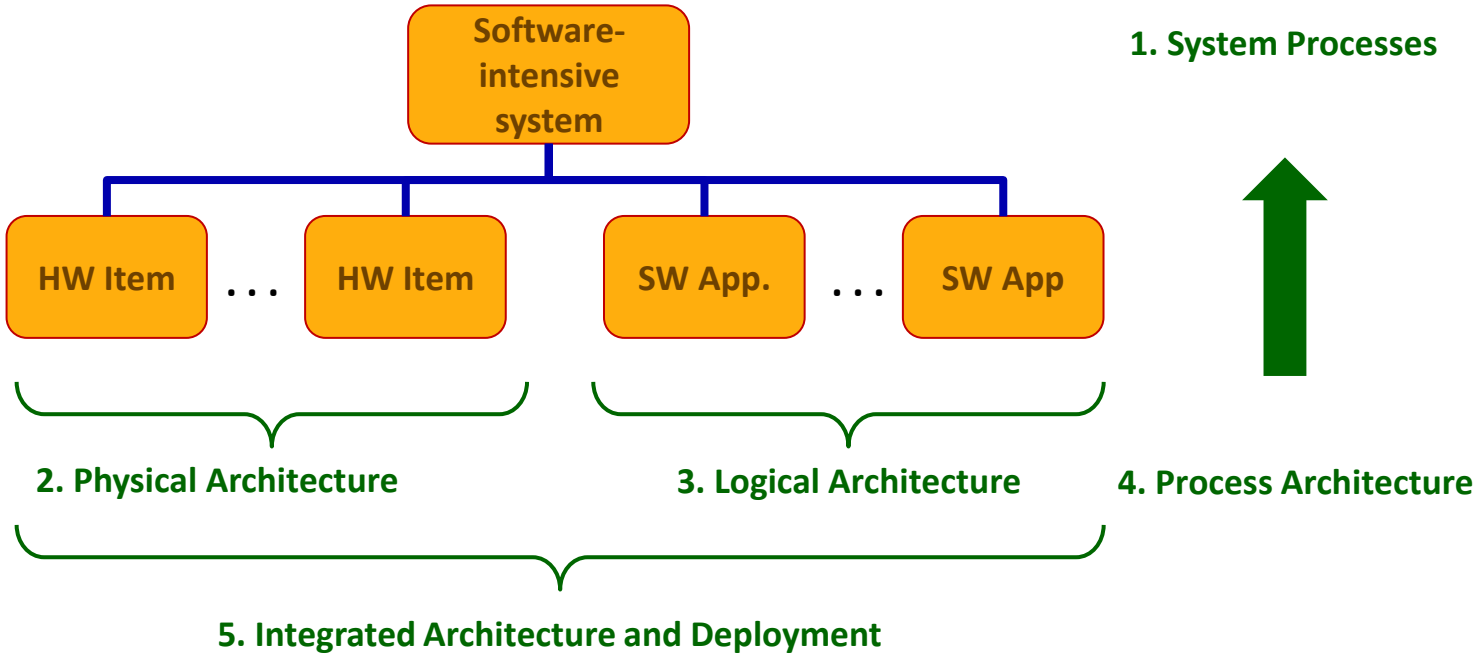
- Shared understanding of the mission (team success)
- Shared responsibility for not dropping the baton
- Running concurrently
- Guaranteed delivery



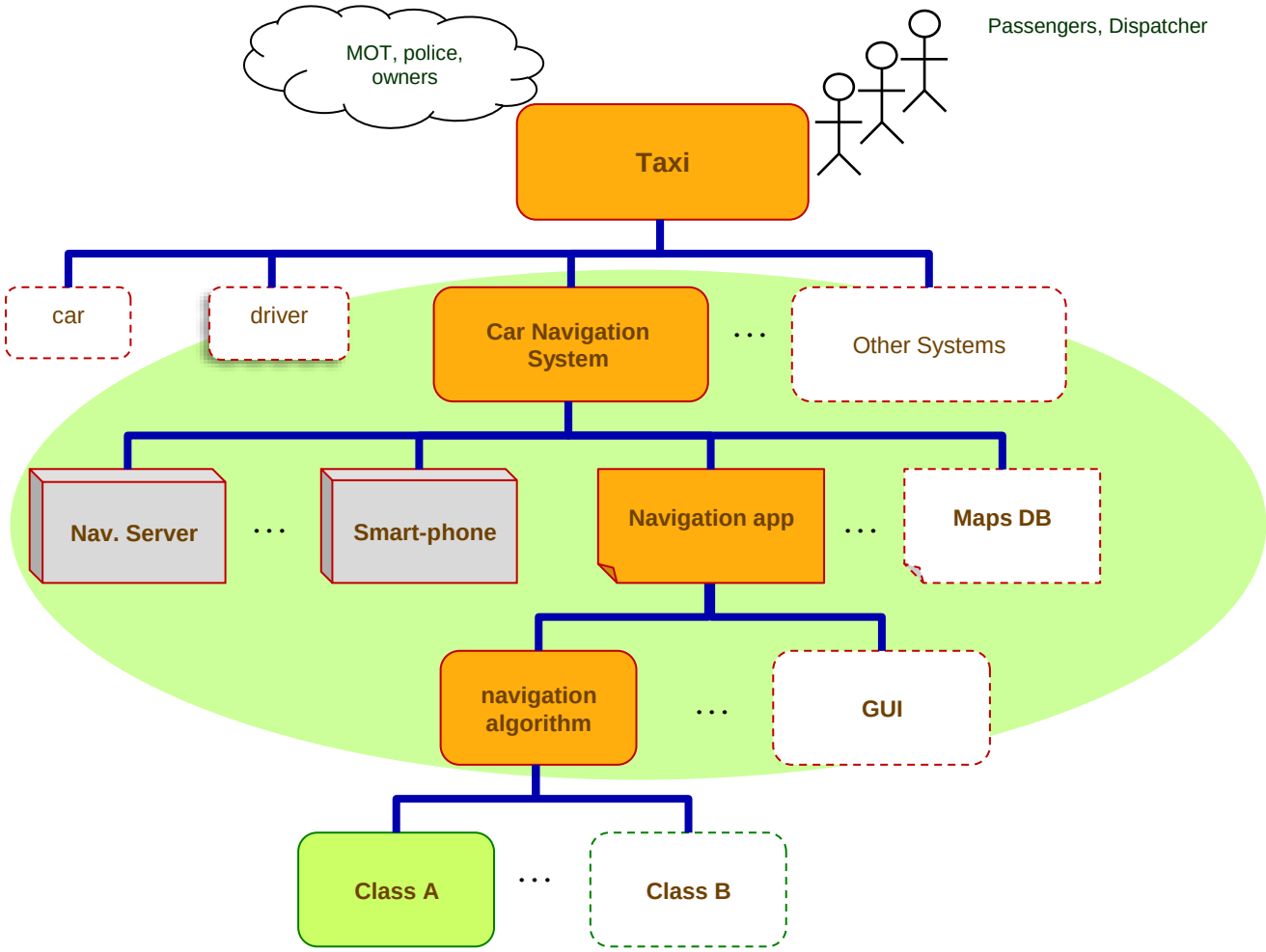
Picture source: https://teachermandc.com/2013/10/27/passing-the-baton-is-no-easy-feat/

System Architecture

- The architecture of a software-intensive system comprises:
  1. System Processes / Use Cases
  2. Physical architecture (hardware components and their physical links)
  3. Logical architecture (software components and their logical links)
  4. The process architecture (how system processes are performed by interactive components)
  5. The integrated architecture and software deployment



An Example: A Car Navigation System for a Taxi Driver



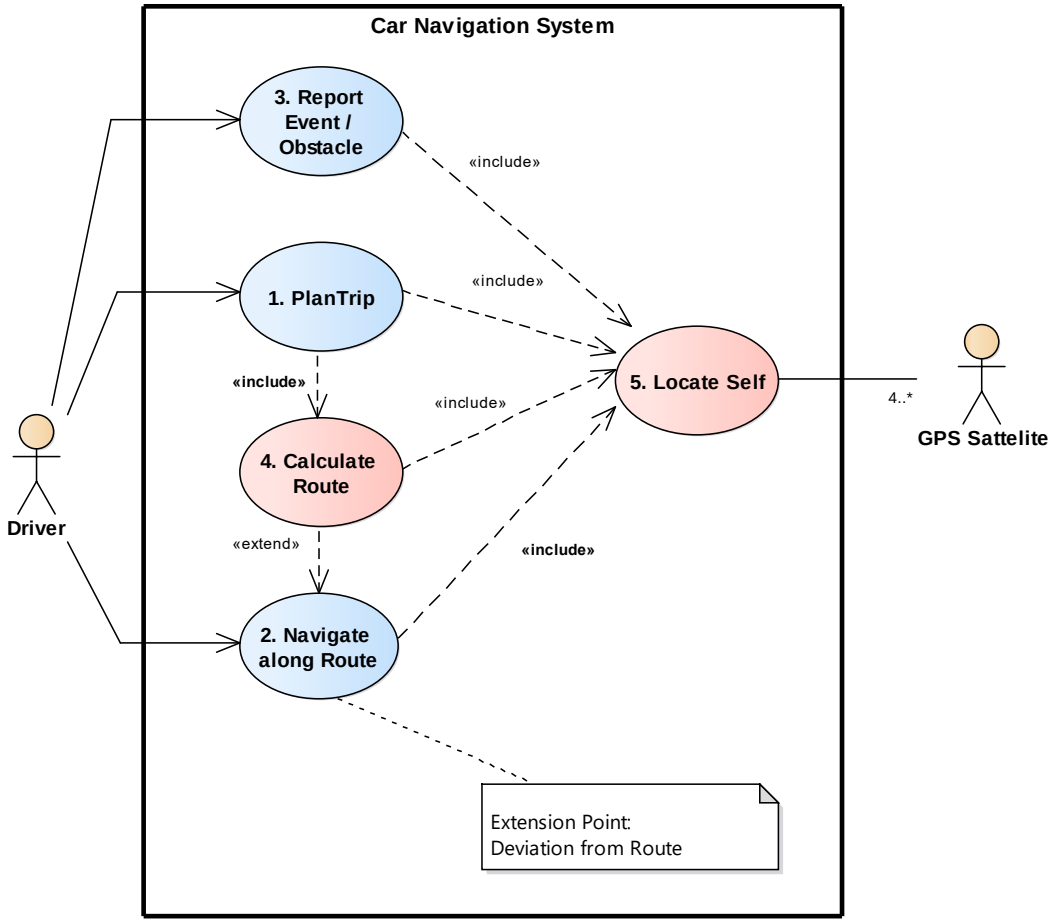
1. System Processes - Activity

Why start with the processes and not with the structure?

- Shared challenge
  - Establishing a functional process-base for the system
- System Engineering input (examples)
  - Business goals / needs
  - Operator/user capabilities
  - Interoperability requirements
  - Quality attributes
- Software Engineering input (examples)
  - Software capabilities
  - Software technologies (Database, Artificial Intelligence, GUI, ...)
- Shared effort
  - Writing Use-Case-based requirements specification

1. System Processes – Result # 1: Use Case Diagram

- The (entire) system in its environment



1. System Processes – Result # 1: Use Case Specifications

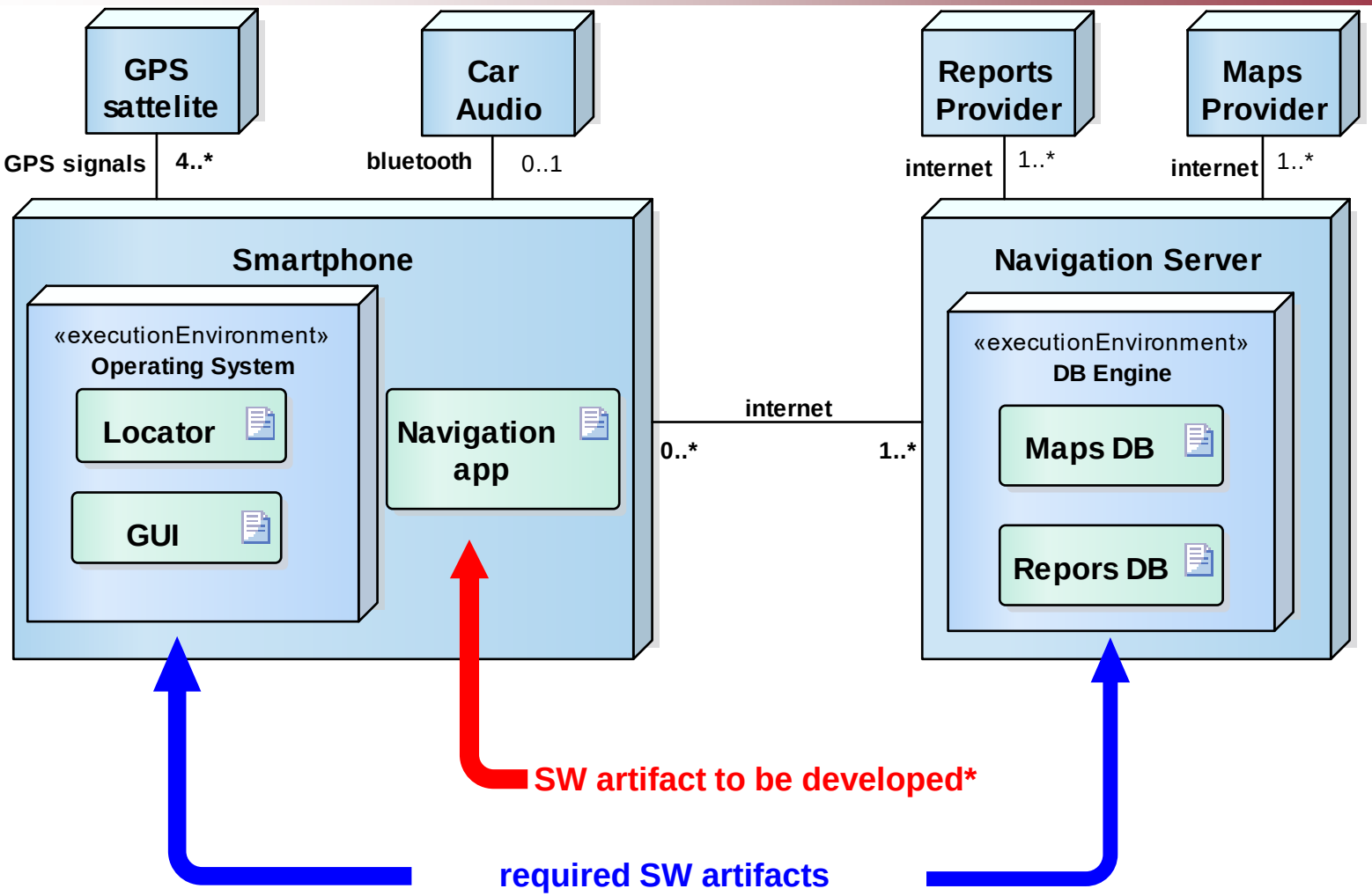
- E.g. 'Navigate along route' Use Case

Use Case # 2	Navigate along route
Actors	Driver: Primary actor. Goal: Get to destination GPS Satellites: Supporting actor
Other stakeholders and their interests	Business owners: passengers' satisfaction MOT, police: Use proper routes
Pre-Conditions	A route to destination exists (By UC#1: Plan Trip)
Post Conditions	Arrived at destination
Trigger	**The driver** presses the "GO" button
Main Success Scenario (MSS)	1. **The system** enters active mode 2. **The system** retrieves the existing 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 (inc. UC#5: Locate Self) 7. If the location is off route – **the system** re-calculates the route (ext. UC#4: Calc. Route) 8. Back to step 4
Branch A	**Exception in any step** of MSS: **The driver** presses the "Cancel" Key The scenario terminates

2. Physical Architecture – Activity

- Shared challenge
  - Establishing the computational platform of the system
- System Engineering input (examples)
  - The external environment
  - Computers and peripherals
  - Communication links/networks and protocols
  - COTS components (hardware)
- Software Engineering input (examples)
  - Operating systems
  - Execution environments (e.g. database engines, web browsers, ...)
  - Required dynamic libraries / other environment components
- Shared effort
  - Building a (partial) physical architecture

2. Physical Architecture – Result

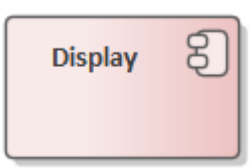


* Deployment and other decisions may be deferred to later stages

3. Logical (Software) Architecture – Activity

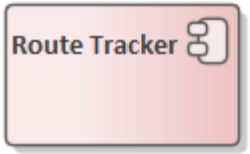
- Shared challenge
  - Establishing the software architecture of the system
    - functional components and their interfaces
- System Engineering input (examples)
  - Overall system functionality
  - Cohesion / coupling considerations
  - Synchronous / A-synchronous interaction
- Software Engineering input (examples)
  - Existing components (e.g. open-source, reuse, ...)
  - Software architecture frameworks
  - Logical data representation and protocols (e.g. HTTP, XML, Json, ...)
- Shared effort
  - Functional analysis and allocation
  - Logical interface design
  - Entire logical / functional connectivity

Functional Analysis and Allocation

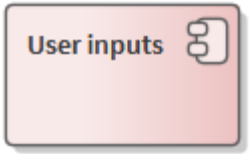




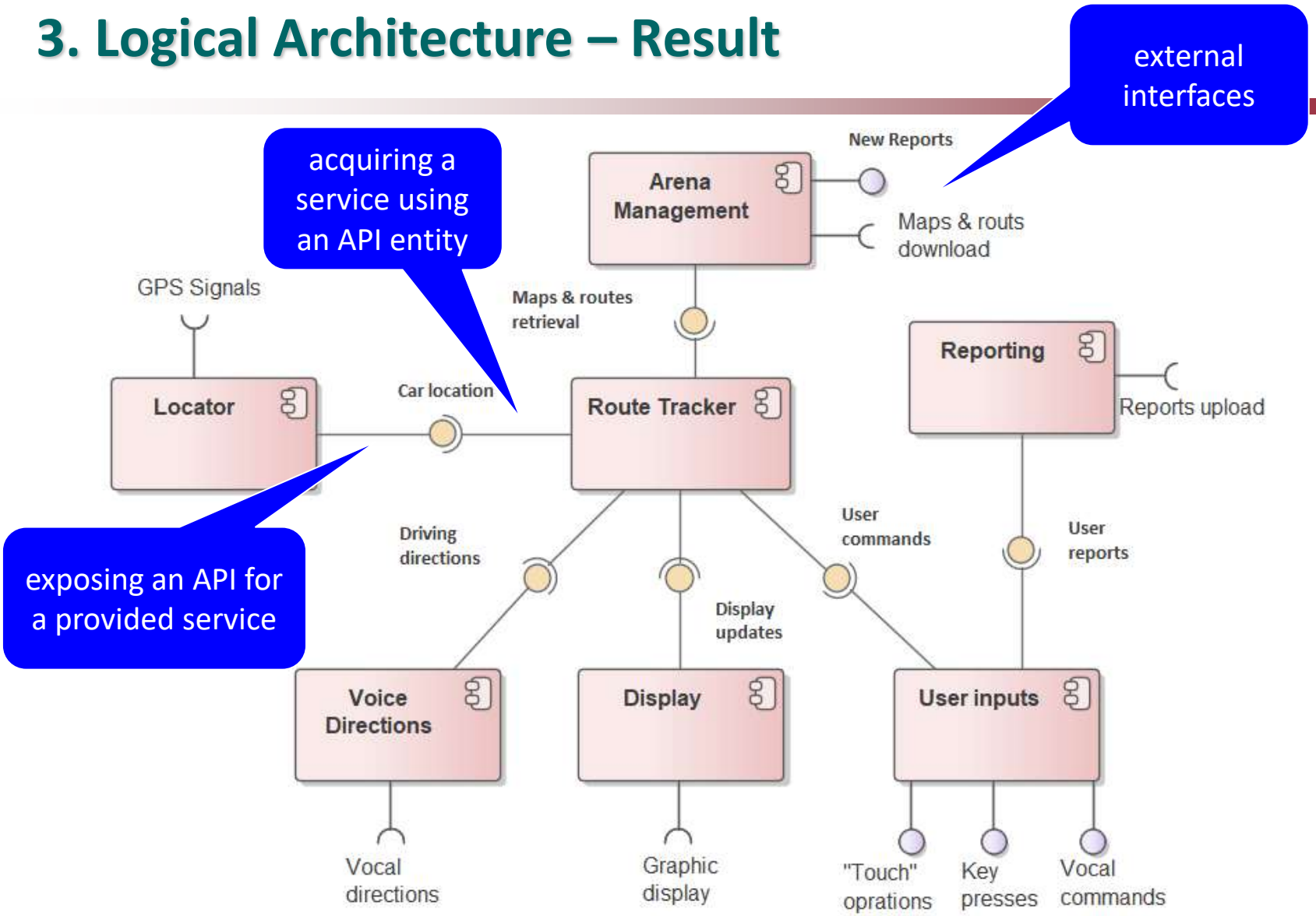
Main Success Scenario (MSS)	1. **The system** enters active mode 2. **The system** retrieves the existing route 3. **The system** retrieves a relevant map area 4. If the locally stored map data is missing – **the system** downloads map data from the server 5. **The system** displays the map and route 6. **The system** gets the current location from the GPS (inc. UC#5: Locate Self) 7. If the location is off route – **the system** re-calculates the route (ext. UC#4: Calc. Route) 8. Back to step 4
Branch A	**Exception in any step** of MSS: **The driver** presses the "Cancel" Key The scenario terminates







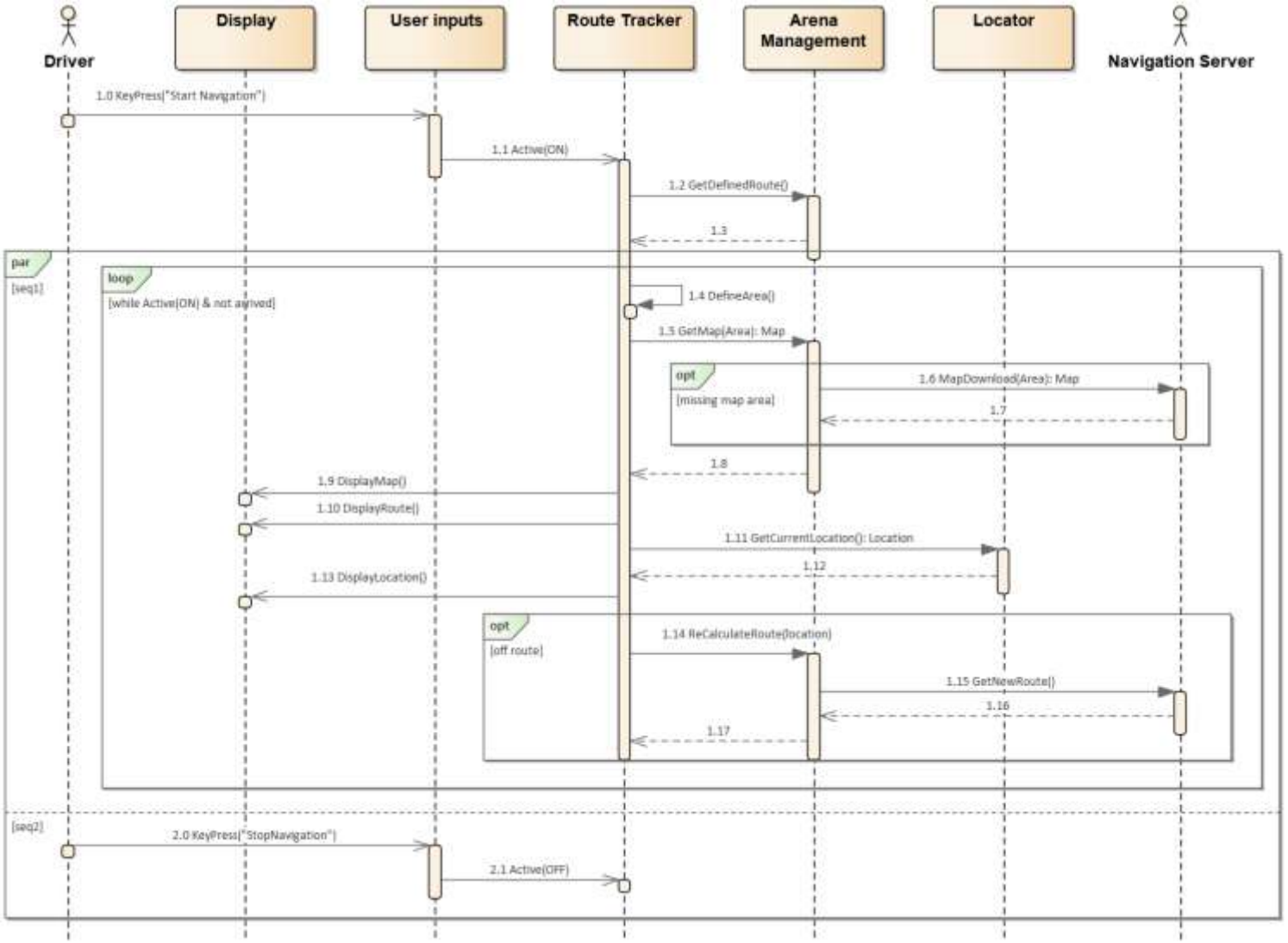
3. Logical Architecture – Result



4. Process Architecture – Activity

- Shared challenge
  - Transform the system processes (use cases) into component interaction
    - Using the components' functional allocation and designed interfaces
- System Engineering input (examples)
  - Communication capability / constrains
  - Computation / memory capabilities / constrains
  - Synchronous / A-synchronous interaction
  - Environment's interfaces
- Software Engineering input (examples)
  - Components' APIs
  - Standard APIs (e.g. SQL)
  - Software architecture patterns (e.g. broker, pipeline, MVC, map-reduce)
- Shared effort
  - Interaction diagrams

4. process Architecture – Result



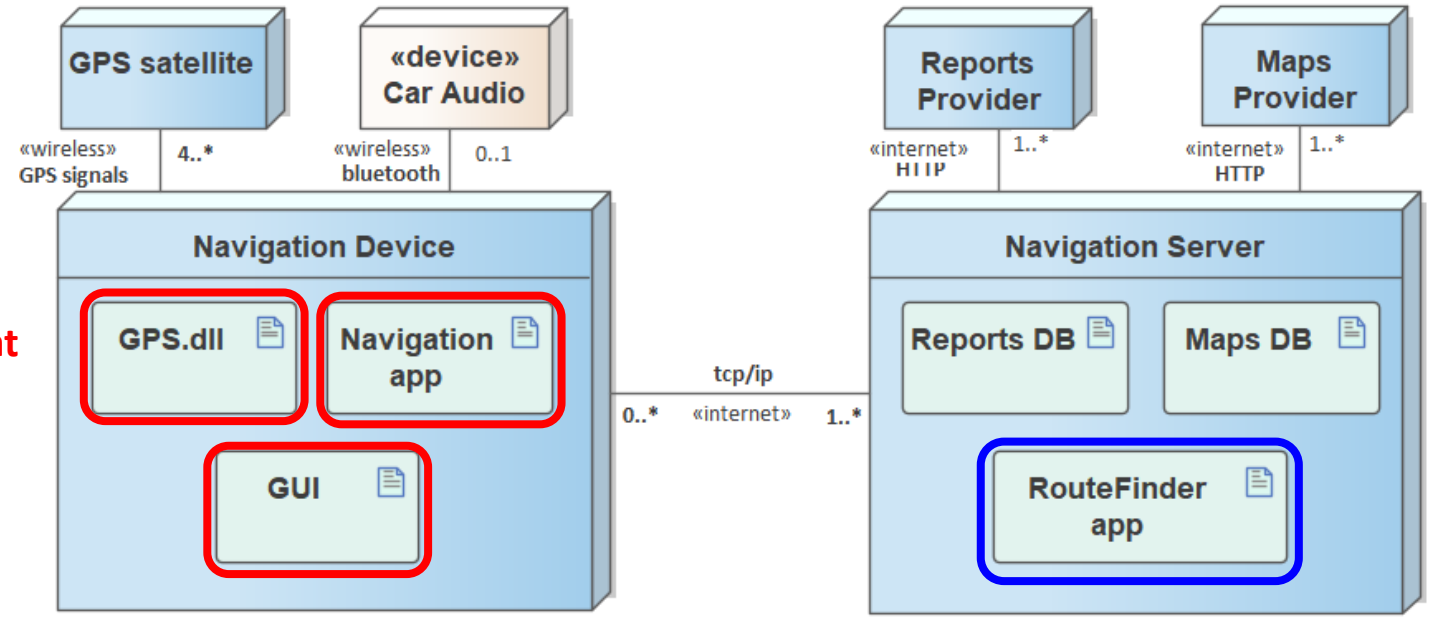
5. Integrated Architecture and Deployment – Activity

- Shared challenge
  - introduce an integrated software+hardware architecture / configuration
- System Engineering input (examples)
  - Communication implications (latency, rate, volume)
  - Hardware ports
  - Various system configurations [Tomer, 2012]
- Software Engineering input (examples)
  - Marshalling/unmarshalling messages
  - Build Scripts

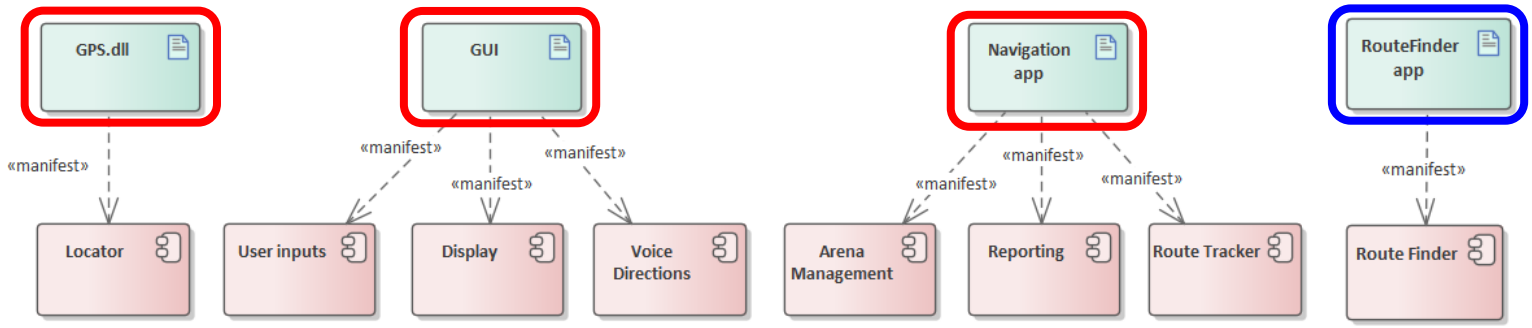
5. Build and Deployment – Distributed Architecture

- Build and deployment is physical architecture/configuration dependent

Deployment

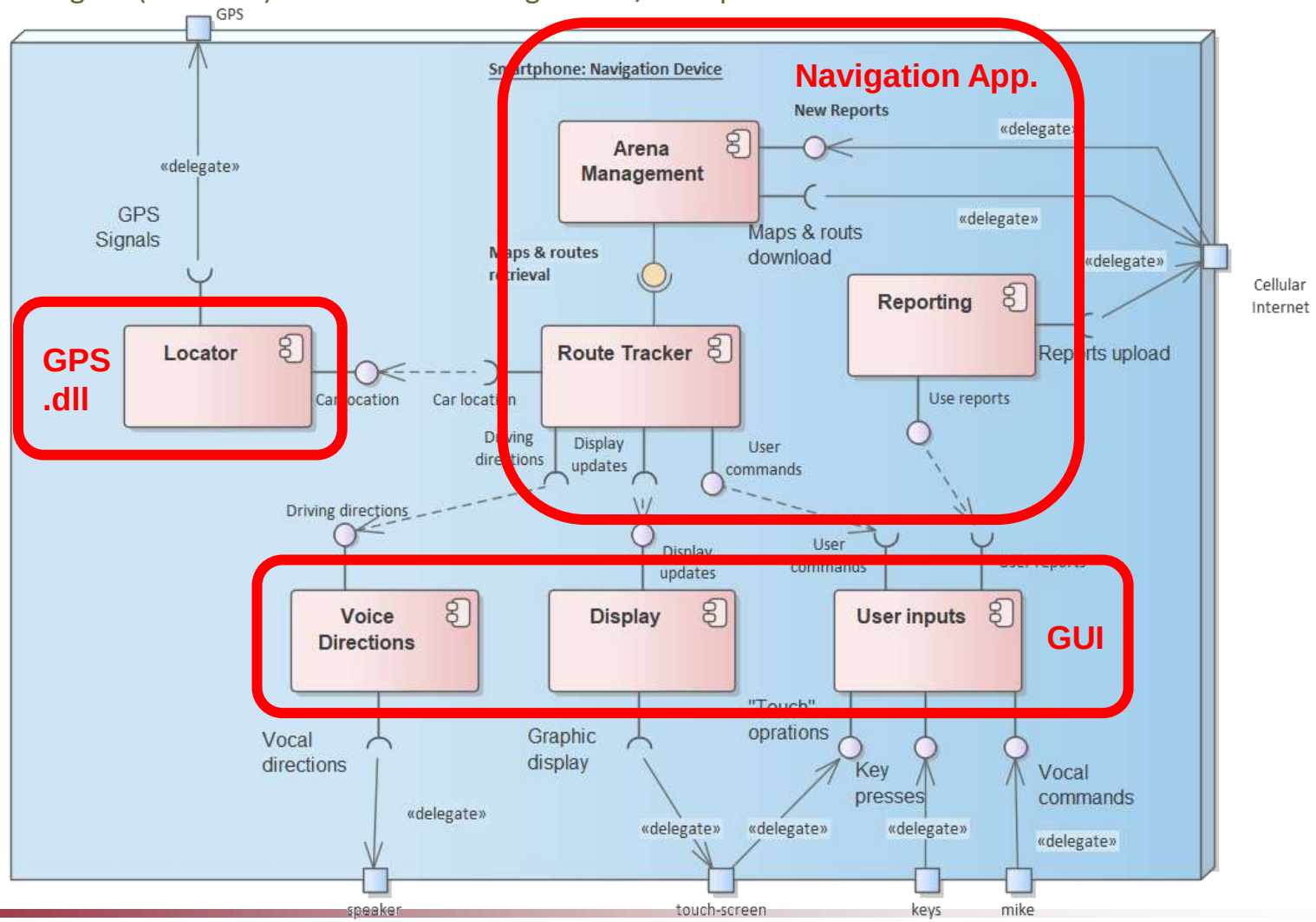


Build



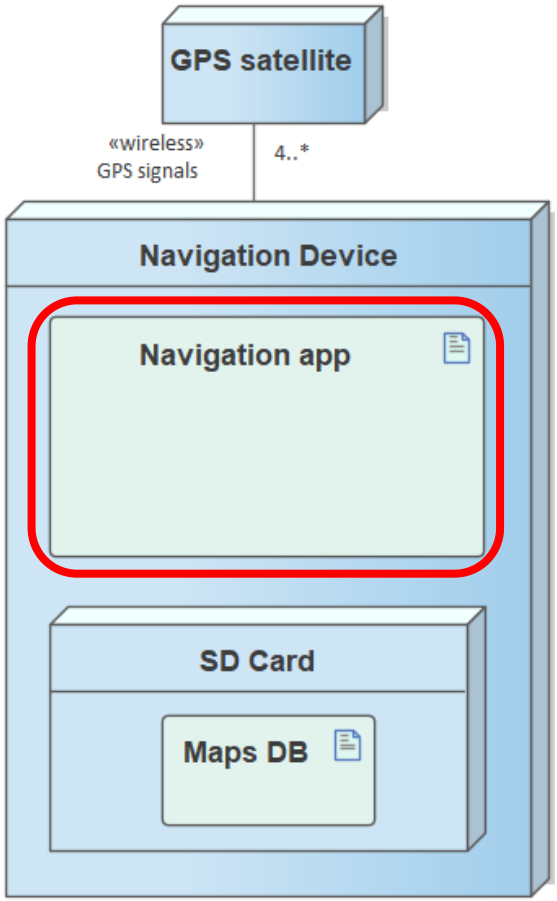
5. Integrated Architecture – Smartphone in Dist. Arch.

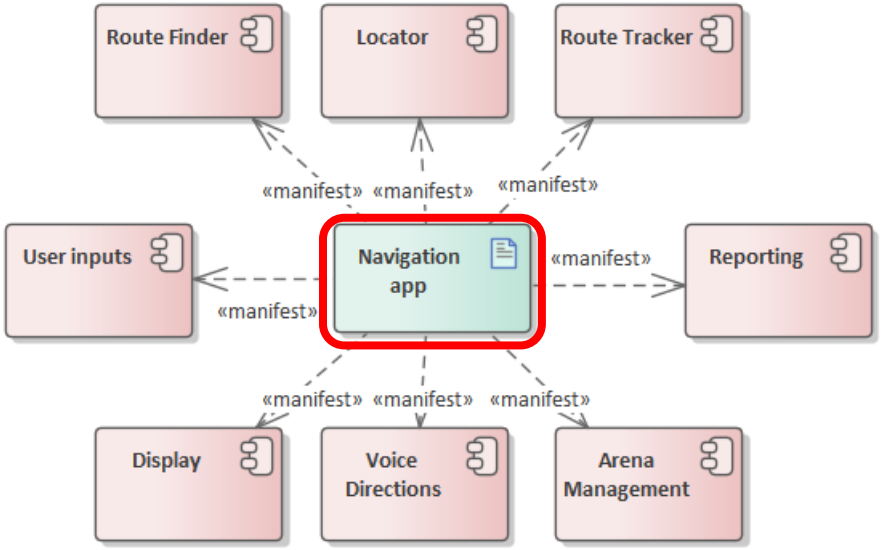
- Physical links are represented by ports
  - Logical (software) interfaces are delegated to/from ports



5. Build and Deployment – Standalone Architecture

- Build and deployment is physical architecture/configuration dependent





Digital Twins

- Definition:
  - A set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level
- Assumptions:
  1. The physical twin and the digital twin share the same architecture
     - I.e., there are two configurations derived from the same architecture
       I. A hardware-software configuration (the physical twin)
       II. A software-only configuration (the digital twin)
  2. The digital twin is developed upfront, to help validating the architecture
- Conclusion:
  - The common architecture is a shared challenge! [Tomer, 2012]

Final Note

If you are a systems engineer...

... or a software engineer / architect ...

... and you think that NOT EVERYTHING said in this presentation applies to you ...

... then ...

think again!

References

- Tomer, A., Ch. 5: Relay Race – The Shared Challenge of Systems and Software Engineering, in *Systems Engineering in the Fourth Industrial Revolution*, (Ed. Kenett, R. S., Swartz, R. S., Zonnenshain, A.), Wiley, December, 2019.
- Tomer, A., "Software Modeling from Life-Cycle Perspective", *Proceedings of the IEEE CS International Conference on Software, Science, Technology, and Engineering (SwSTE 2012)*, Hertzliyah, Israel, June 12-13, 2012.
  - DOI: 10.1109/SWSTE.2012.17

More to come...

- Other tutorials by this author
  - Webinars
    - Making Use Cases great again (~1 hour)
    - The Functionality behind the Non-functional Requirements (~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

