Goal-Oriented Requirements Engineering and Software Architecting

Lawrence Chung
Department of Computer Science
The University of Texas at Dallas
Outline

- Running Example: London Ambulance System
- Goal-Orientation
- Goal-Oriented Requirements Engineering
- Goal-Oriented Software Architecting
London Ambulance Service (LAS)
1992 Computer-Aided Dispatch (CAD) System Debacle
Nasima Begum’s death

Nasima Begum with liver condition

4 emergency calls

Call Taker

the only available ambulance sent to a non-emergency call

Died after waiting 53 minutes for an ambulance

Lived only 2 blocks from the hospital

Note: some source (Guy Fitzgeral’s “The Turnaround of the London Ambulance…” indicated this incident occurred after LAS went back to use the manual dispatch in June 1994 after the mishap in 1992 while some source cited this incident in the 1992 mishap (is it D. Dalcher’s “Disaster…”?).
London Ambulance 1992 dispatch system: What should have happened

Nasima Begum with liver condition

Request for ambulance

Call Taker

Incident info processed by Dispatch system

Dispatch System

Dispatch closest ambulance

Report current location and status

Ambulance & Crew

at station

Mobilize to scene

dispatched in 3 minutes from call

mobilized to scene in 11 minutes from dispatch
Investigative reports say

London Ambulance Service (LAS) deployed a new computer-aided dispatch system intended to comply with the new regulation that required an ambulance to be dispatched in 3 seconds and arrived at the scene in 11 seconds.

However, the new system not only failed to meet the timeliness requirements, it was also unreliable and completely crashed.

- Many died from not getting care in time:
  - An 11-year old girl died of a kidney condition after waiting for 53 mins
  - A man died of heart attack after waiting for 2 hours
- Multiple ambulances sent to the same incident
- Lost track of the ambulance status such that operators had to call the caller back to see if an ambulance had arrived

A workshop in software engineering concluded: **NFRs (non-functional requirements) were not considered early in the development process**, among other organizational and software engineering mistakes.
An architecture for the London ambulance dispatch system

How to develop a software architecture more systematically?
Outline

- Running Example: London Ambulance System
- Goal-Oriented Requirements Engineering
- Goal-Oriented Software Architecting
Goal!
Goal-Oriented Orientation

Goal-oriented analysis focuses on the description and evaluation of alternatives and their relationship to the organizational objectives.

- facilitates systematic exploration of, and selection among, requirements and then architectural design alternatives.

- establishes traceability and justifiability

- a “rational” approach
Goal-Oriented Process

Incremental, Iterative and Interleaving steps

Post goals – both functional and non-functional

Consider alternatives for operationalizations

Refine goals

Carry out tradeoff analysis among alternatives wrt. non-functional goals
What’s Operationalization?

Anna: Do I look happy?

Carlos: Yes, you look happy.

Anna: Why?

Carlos: Because you look happy.

Anna: Why?

Carlos: Because you look happy.

Anna: Why?

Carlos: Because you look happy.

Anna: Why?

Carlos: I’m hungry now.
What’s Operationalization?

Operationalization is the process of defining a fuzzy concept so as to make the concept measurable in form of variables consisting of specific observations. In a wider sense it refers to the process of specifying the extension of a concept.

Anna: Do I look happy?

Carlos: Yes, you look happy.

Anna: Why?

Carlos: Because you smile a lot, your voice is soft, and you buy me drinks these days.
Operationalization is the process of defining a fuzzy concept so as to make the concept measurable in form of variables consisting of specific observations. In a wider sense it refers to the process of specifying the extension of a concept.

Anna: Do I look happy?

Carlos: Yes, you look happy.

Anna: Why?

Carlos: Because you smile a lot, your voice is soft, and you buy me drinks these days.

Anna: Really?

Carlos: Yes, you smile every day, your voice is soft, and you buy me expensive drinks these days.
What’s Operationalization?

From Wikipedia, the free encyclopedia

Operationalization is the process of defining a fuzzy concept so as to make the concept measurable in form of variables consisting of specific observations. In a wider sense it refers to the process of specifying the extension of a concept.

Anna: Do I look happy?

Carlos: No, you look unhappy.

Anna: Why?

Carlos: Because you cry a lot, you don’t talk, and you don’t buy me drinks these days.
Goal-Oriented Requirements Engineering (GORE) &
Goal-Oriented Software Architecting (GOSA)

A software Architecture should help its intended stakeholders achieve their goals. More narrowly speaking, a software architecture should meet its intended software requirements.
A GORE-and-GOSA process

Incremental, Iterative and Interleaving steps

GORE

Step 1: Develop goal/softgoal models and choose operationalization alternatives
  1.1 Identify and refine goals
  1.2 Explore and trade-off goal operationalizations
  1.3 Identify and refine softgoals
  1.4 Explore and trade-off softgoal operationalizations

Step 2: Develop domain model and identify associated producer/consumer goals

GOSA

Step 3: Identify components based on goal-domain/entity relationships

Step 4: Identify component dependencies
  4.1 Identify process component - process component dependencies
  4.2 Identify process component - input component dependencies
  4.3 Identify process component - output component dependencies

Step 5: Select architectural styles/patterns by NFR softgoals

Step 6: Produce concrete connectors using the chosen style/pattern
Outline

• Running Example: London Ambulance System

• Goal-Orientation

Goal-Oriented Requirements Engineering

• Goal-Oriented Software Architecting
Step 1.1: **Identify and refine goals**

Diagrams developed using the RE-Tools (utdallas.edu/~supakkul/tools/re-tools)
Step 1.2: **Explore alternative tasks for goal operationalizations and carry out tradeoff analysis**

Task/goal operationalization

Softgoal

- Request handled
- Resource assigned
- Least used free resource
- Assign closest free resource
- Assign free resource from station owning the area
- Assign free resource from any station

Agent dependencies or agent assignment

- Call Taker
- Resource Allocator

Timeliness [Ambulance dispatched]

Trade-off goal operationalizations based on softgoal

Crew satisfaction

Time consuming to get back to station
Step 1.3: **Identify and refine softgoals**

Diagram showing the relationships between Timeliness, Ambulance services, and other domain entities.
Step 1.4: Explore softgoal operationalizations and carry out tradeoff analysis

Positive contribution

Negative contribution

Claim/argument

Timeliness [Resource.Status]

"Crew may forget to call"

Use mobile data terminal to report status

Ambulance crew calling to report status

Mobile Data Terminate (MDT)

Crew

Use automatic vehicle location system

Ambulance crew calling in to report location

Automatic Vehicle Location System (AVLS)

Agent dependency or agent assignment

Softgoal operationalization
Step 1: Develop goal and softgoal models and choose operationalization alternatives based on NFR softgoals.
Step 2: Develop domain model and identify associated producer and consumer goals

**Producer goal**: a goal whose fulfillment necessitates changes in a domain/entity

**Consumer goal**: a goal whose fulfillment necessitates use of information from a domain/entity
Outline

• Running Example: London Ambulance System

• Goal-Orientation

• Goal-Oriented Requirements Engineering

Goal-Oriented Software Architecting
A goal-oriented process

*Incremental, Iterative and Interleaving steps*

**GORE**

Step 1: Develop goal/softgoal models and choose operationalization alternatives
1.1 Identify and refine goals
1.2 Explore and trade-off goal operationalizations
1.3 Identify and refine softgoals
1.4 Explore and trade-off softgoal operationalizations

Step 2: Develop domain model and identify associated producer/consumer goals

**GOSA**

Step 3: Identify components based on goal-domain/entity relationships

Step 4: Identify component dependencies
4.1 Identify process component - process component dependencies
4.2 Identify process component - input component dependencies
4.3 Identify process component - output component dependencies

Step 5: Select architectural styles/patterns by NFR softgoals

Step 6: Produce concrete connectors using the chosen style/pattern
Step 3: **Identify components based on goal-entity relationships**

Derive a process component (using a convention e.g. “<domain>Manager”) for a domain/entity having consumer and producer goals.
Repeat Step 3 for components of other domains/entities

Goal+domain model

- Producer goal
  - Assign resource
  - Track resources
  - Handle requests

- Consumer goal
  - Notify resource
  - Select free resource
  - Assign resource

Sw architecture

- Process component
  - Dispatch Manager
    - assignResource()
    - notifyResource()
  - Resource Manager
    - trackResources()
    - selectFreeResource()
  - Request Manager
    - handleRequests()
    - assignResource()
Step 4.1: Identify process component - process component dependencies

If domain D1 depends on producer goal G1 for attribute A1 that is also a consumer goal of attribute A1 from domain D2, then the resulting component C1 (derived from D1) depends on C2 (derived from D2) for attribute A1.
Step 4.2: **Identify process – agent dependencies to identify process component – input component dependencies**

For each producer task/goal that depends on an agent, define an input component using a naming convention e.g. "{agent’s name}Interface" for an agent.
Step 4.2: **Identify process – agent dependencies to identify process component - output component dependencies**

For each consumer goal/task that is depended upon by an agent, define an output component using a convention e.g. `{agent}Interface`
Step 5: Select architectural styles/patterns by NFR softgoals

Modeling of styles using UML is adapted from [3]
Step 6: **Produce concrete connectors using the chosen style/pattern**

Connector pattern

Component dependency

Component concrete connector
Repeat for all component dependencies for the final architecture.
Compared well with an expert-produced architecture

**ACME [D. Garlan]**

**Similarities:**
- call_entry vs. Request Mgr
- incident_mgr vs. Incident Mgr
- Resource_mgr vs. ResourceMgr
- Dispatcher vs. DispatchMgr

**Differences:**
- Additional explicit input/output components
- Visual, UML component diagram

**Benefits:**
- More confident (why each alternative is chosen)
Different alternatives lead to different architectures

Your architecture is as good, or as bad, as your design decisions.
Goal-Oriented Requirements Engineering and Software Architecting

...thanks to

Goal-oriented analysis focuses on the description and evaluation of alternatives and their relationship to the organizational objectives.

- facilitates systematic exploration of, and selection among, requirements and then architectural design alternatives.

- establishes traceability and justifiability

- a “rational” approach
Challenges & Opportunities

- Architecture is the only thing that matters ???
- Transition from RE to SAD
- Goal-oriented software architecture (re-)confirmation
- Exploration of alternatives with more ontological concepts
- Applications to health-care, energy, education, cyber-physical systems, etc.
Thank You