Software Requirements Engineering

Part 1. Foundations

Emmanuel Letier
http://www.cs.ucl.ac.uk/staff/E.Letier/

Outline

1. What is requirements engineering?
2. A Theory of Requirements Engineering – Part I
3. A Theory of Requirements Engineering – Part II
4. Requirements Errors
5. Requirements in the Software Lifecycle
6. Requirements in the Wider Context

Lecture notes:
http://www.cs.ucl.ac.uk/staff/E.Letier/book/

1. What is Requirements Engineering?

A Working Definition

In software engineering, requirements engineering is a set of activities concerned with discovering, analysing, and communicating

- the stakeholder needs for a software system
- the contexts in which the software system is used, and
- the externally visible behaviours and qualities the software must posses to satisfy the stakeholder needs in a given context.

It involves finding suitable tradeoffs between conflicting stakeholder needs, and between stakeholder needs and the capabilities afforded by technologies. It also involves managing changes to stakeholder needs, contexts and the desired software behaviours and qualities.

A Working Definition

In software engineering, requirements engineering is a set of activities concerned with discovering, analysing, and communicating

- the stakeholder needs for a software system
- the contexts in which the software system is used, and
- the externally visible behaviours and qualities the software must posses to satisfy the stakeholder needs in a given context.

It involves finding suitable tradeoffs between conflicting stakeholder needs, and between stakeholder needs and the capabilities afforded by technologies. It also involves managing changes to stakeholder needs, contexts and the desired software behaviours and qualities.

Not the first phase of development!

A Working Definition

In software engineering, requirements engineering is a set of activities concerned with discovering, analysing, and communicating

- the stakeholder needs for a software system
- the contexts in which the software system is used, and
- the externally visible behaviours and qualities the software must posses to satisfy the stakeholder needs in a given context.

It involves finding suitable tradeoffs between conflicting stakeholder needs, and between stakeholder needs and the capabilities afforded by technologies. It also involves managing changes to stakeholder needs, contexts and the desired software behaviours and qualities.

stakeholders = all people affected by the software, not just the client and users.
stakeholder needs = people's wishes and expectations about the software impacts on their lives, work and businesses.

A Working Definition

In software engineering, requirements engineering is a set of activities concerned with discovering, analysing, and communicating

- the stakeholder needs for a software system
- the contexts in which the software system is used, and
- the externally visible behaviours and qualities the software must posses to satisfy the stakeholder needs in a given context.

It involves finding suitable tradeoffs between conflicting stakeholder needs, and between stakeholder needs and the capabilities afforded by technologies. It also involves managing changes to stakeholder needs, contexts and the desired software behaviours and qualities.

Understanding context of use is essential!

A Working Definition

In software engineering, requirements engineering is a set of activities concerned with discovering, analysing, and communicating

- the stakeholder needs for a software system
- the contexts in which the software system is used, and
- the externally visible behaviours and qualities the software must posses to satisfy the stakeholder needs in a given context.

It involves finding suitable tradeoffs between conflicting stakeholder needs, and between stakeholder needs and the capabilities afforded by technologies. It also involves managing changes to stakeholder needs, contexts and the desired software behaviours and qualities.

Every software has requirements; even if not written down or communicated clearly.

A Working Definition

In software engineering, requirements engineering is a set of activities concerned with discovering, analysing, and communicating

- the stakeholder needs for a software system
- the contexts in which the software system is used, and
- the externally visible behaviours and qualities the software must posses to satisfy the stakeholder needs in a given context.

It involves finding suitable tradeoffs between conflicting stakeholder needs, and between stakeholder needs and the capabilities afforded by technologies. It also involves managing changes to stakeholder needs, contexts and the desired software behaviours and qualities.

Understanding tradeoffs is essential

A Working Definition

In software engineering, requirements engineering is a set of activities concerned with discovering, analysing, and communicating

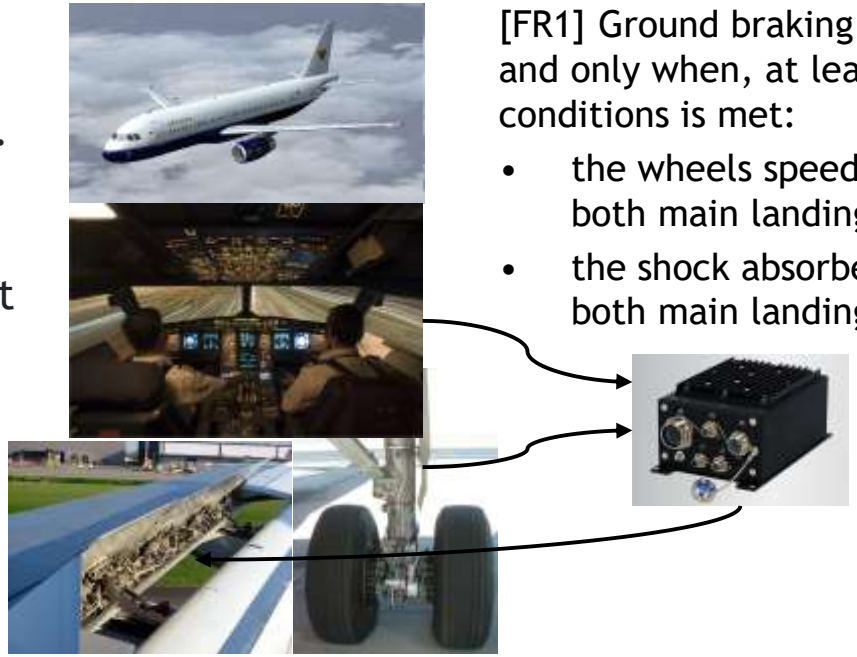
- the stakeholder needs for a software system
- the contexts in which the software system is used, and
- the externally visible behaviours and qualities the software must posses to satisfy the stakeholder needs in a given context.

It involves finding suitable tradeoffs between conflicting stakeholder needs, and between stakeholder needs and the capabilities afforded by technologies. It also involves managing changes to stakeholder needs, contexts and the desired software behaviours and qualities.

Changes across time and across variants in a product family

Example 1: Airplane Ground Braking System

1. The ground spoilers and reverse thrust should not be activated during flight.
2. Pilots must be able to activate the ground spoilers and reverse thrust immediately after the plane has landed.



[FR1] Ground braking must be enabled when, and only when, at least one of the following conditions is met:

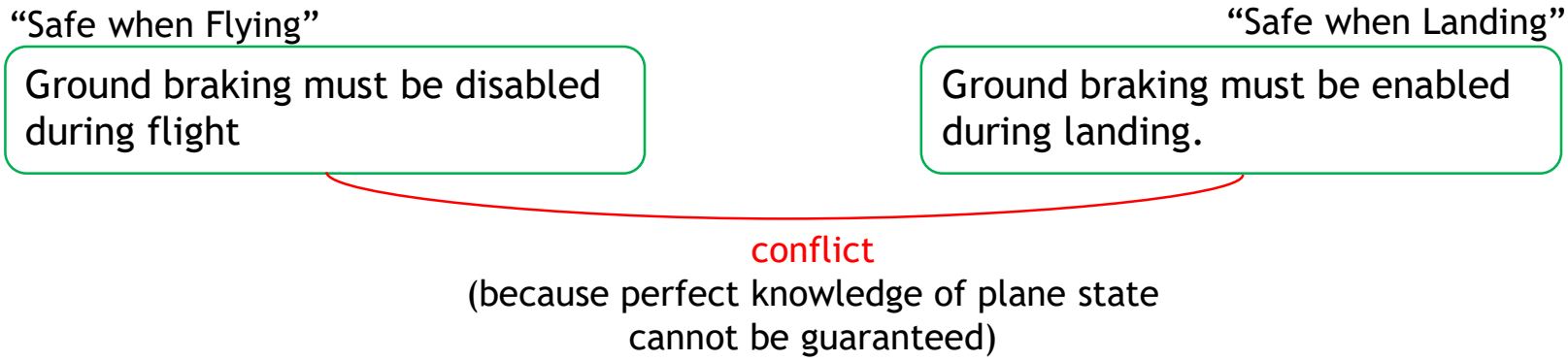
- the wheels speed is above 72 knots at both main landing gears
- the shock absorbers are compressed at both main landing gears

Stakeholder needs

Context

Ground Braking Safety Controller

Tradeoffs



Difficult tradeoff because we need to consider many scenarios for landing, takeoff and flight, including the possibility of human errors and sensor failures.

Managing Changes

The ground braking safety controller (our software) has many versions and variants.

- Versions = change over time due to
  - changes in the context (e.g. new types of sensors)
  - changes in stakeholder needs and priorities
  - new regulations on airplane safety
  - accidents or near misses
- Variants = change over members of a product family
  - for different airplane models
  - or, possibly, for different clients or regions of the world.

→ Need to keep track of many requirements for multiple software versions and variants.

Example 2: Ambulance Dispatch Software

Ambulances should arrive quickly in response to emergency calls

Stakeholder needs

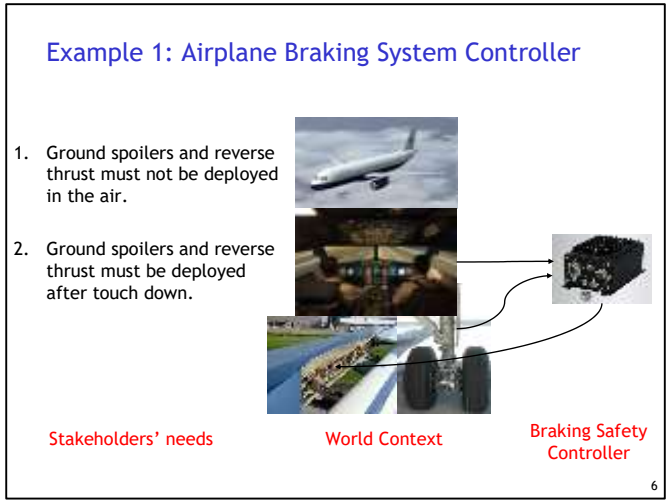
Context

Ambulance Dispatch Software

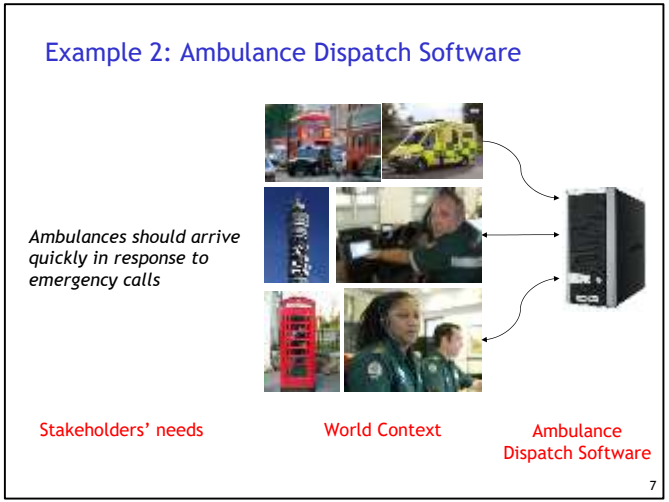
Required features:

- on-screen call taking
- automated caller location detection
- identification of duplicate calls
- automatic vehicle location through GPS
- allocation of nearest available ambulance
- communication of mobilisation orders to ambulance mobile data terminals
- …

Learning from failures



Airbus crash on Warsaw airport runway in 1993



London Ambulance Service Failure in 1992

Common Misconception #1

Requirements engineering is the first phase of development.

Requirements engineering is not a phase; it is a set of activities that occur throughout the software development lifecycle.

Some requirements engineering is done by developers during implementation!

Common Misconception #2

Requirements engineering is asking people what they want.

Requirements engineering ≠ writing a prioritized list of user stories, use cases, features, or “shall statements”.

Analogy: restaurant waiters vs. medical doctors.

Revision: True or False?

1. Requirements engineering is the first phase of software development.
2. Requirements engineering involves discovering, clarifying, analysing, and communicating the needs, context, and requirements for a software system.
3. Requirements engineering is about understanding the stakeholders' needs, not about defining how these needs can be met.
4. Requirements engineering deals with tradeoffs between conflicting needs.
5. Requirements must be defined by clients, not by software engineers.

2. A Theory of Requirements Engineering – Part I

Motivation

Theories help engineers understand and improve their practices.

This theory of requirements engineering will help you understand and be more skillful with *all requirements engineering methods*.

Key ideas:

- World phenomena vs machine phenomena
- The most important formula of requirements engineering

$$Req, Dom \vdash Goals$$

The World and the Machine

The Machine is the product to be developed or improved.
Synonyms: the product, the software-to-be, the system-to-be, etc.

We create concrete machines, not just lines of codes.
These machines have tangible effects on people and the world.

The World is the part of the real-world affected by the machine.
Synonyms: the application domain, the environment, the context.

The World is the scope of our requirements investigations.

Phenomena

A *phenomenon* is an observable state or event.

A *state* is a condition that something or someone is in at a specific time.

An *event* is something that occurs at a specific time.

Phenomena descriptions	Shorthand notations
the plane is flying	`Flying`
the plane is moving on the runway	`MovingOnRunway`
reverse thrust is deployed	`RevThrustDeployed`
the landing gear wheels are turning	`WheelsTurning`
the landing gear sensors signal that wheels are turning	`WheelsPulsesOn`
ground braking is enabled	`GrdBrakingEnabled`

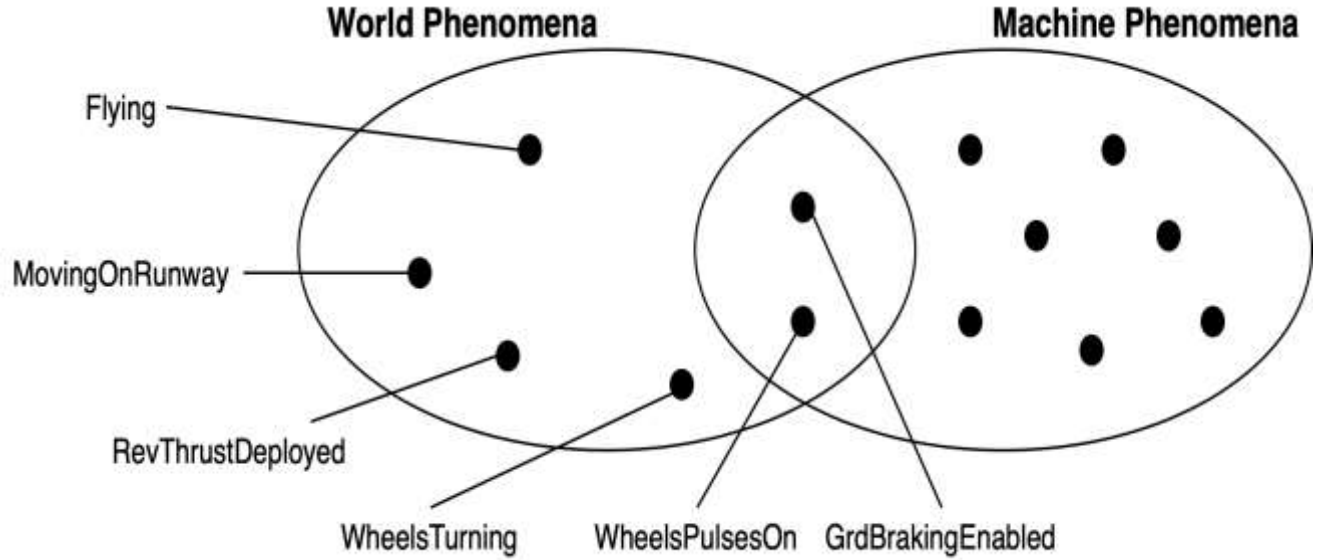
Actual phenomena

Phenomena location

A *World phenomenon* is a phenomenon located in the World.

A *Machine phenomenon* is a phenomenon located in the Machine.

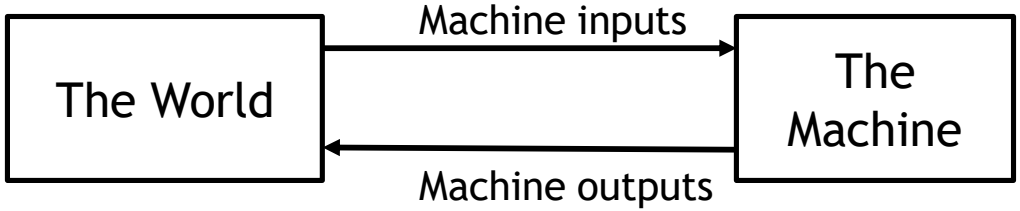
A *shared phenomenon* is a phenomenon located at the interface between the World and the Machine.



Machine Inputs and Outputs

A machine input is a shared phenomenon controlled by the World.

A machine output is a shared phenomenon controlled by the Machine.



Phenomena for the Ground Braking System

Cockpit_cmd

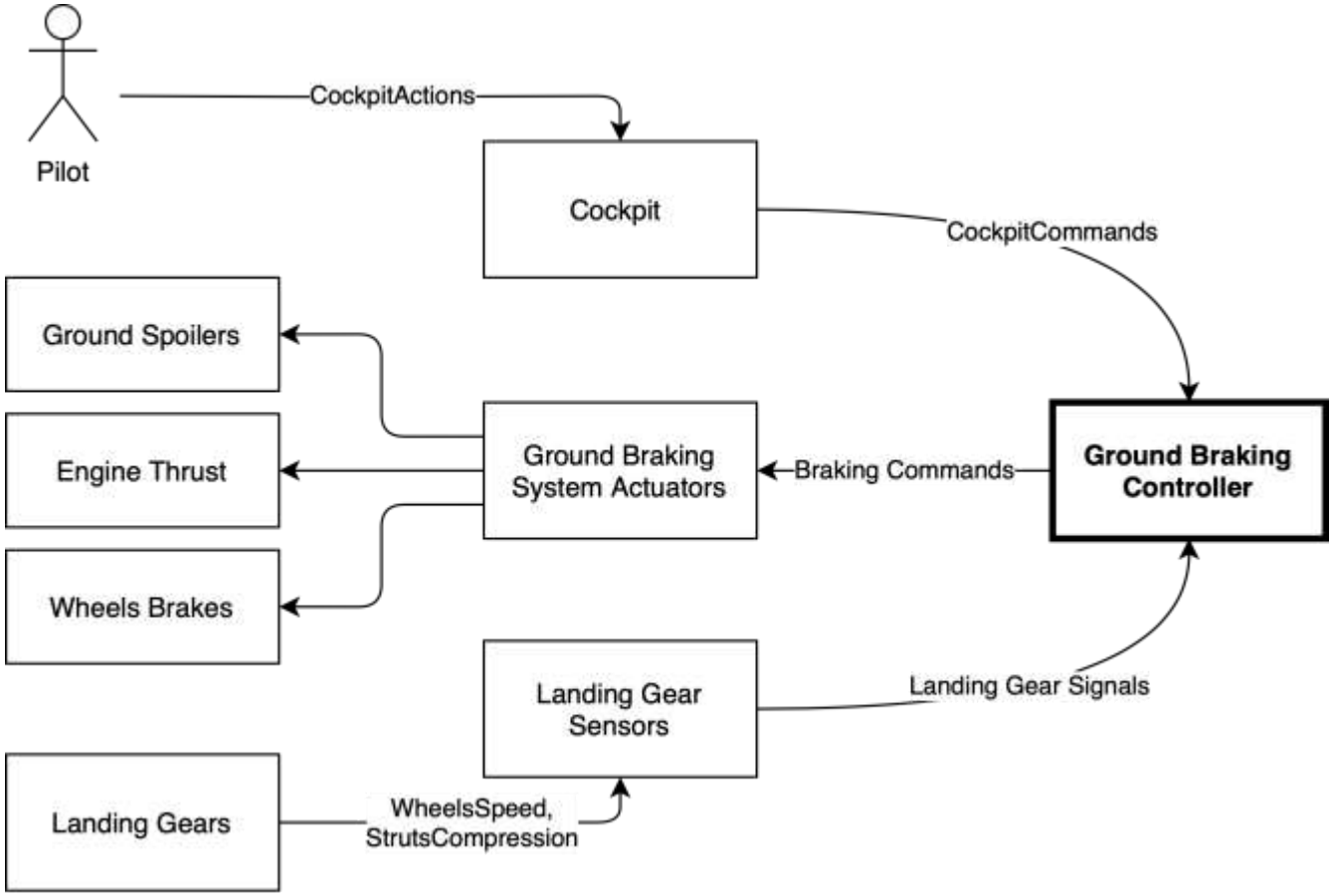
PlaneFlying
SpoilersDeployed
ReverseThrustDeployed

Braking_cmd

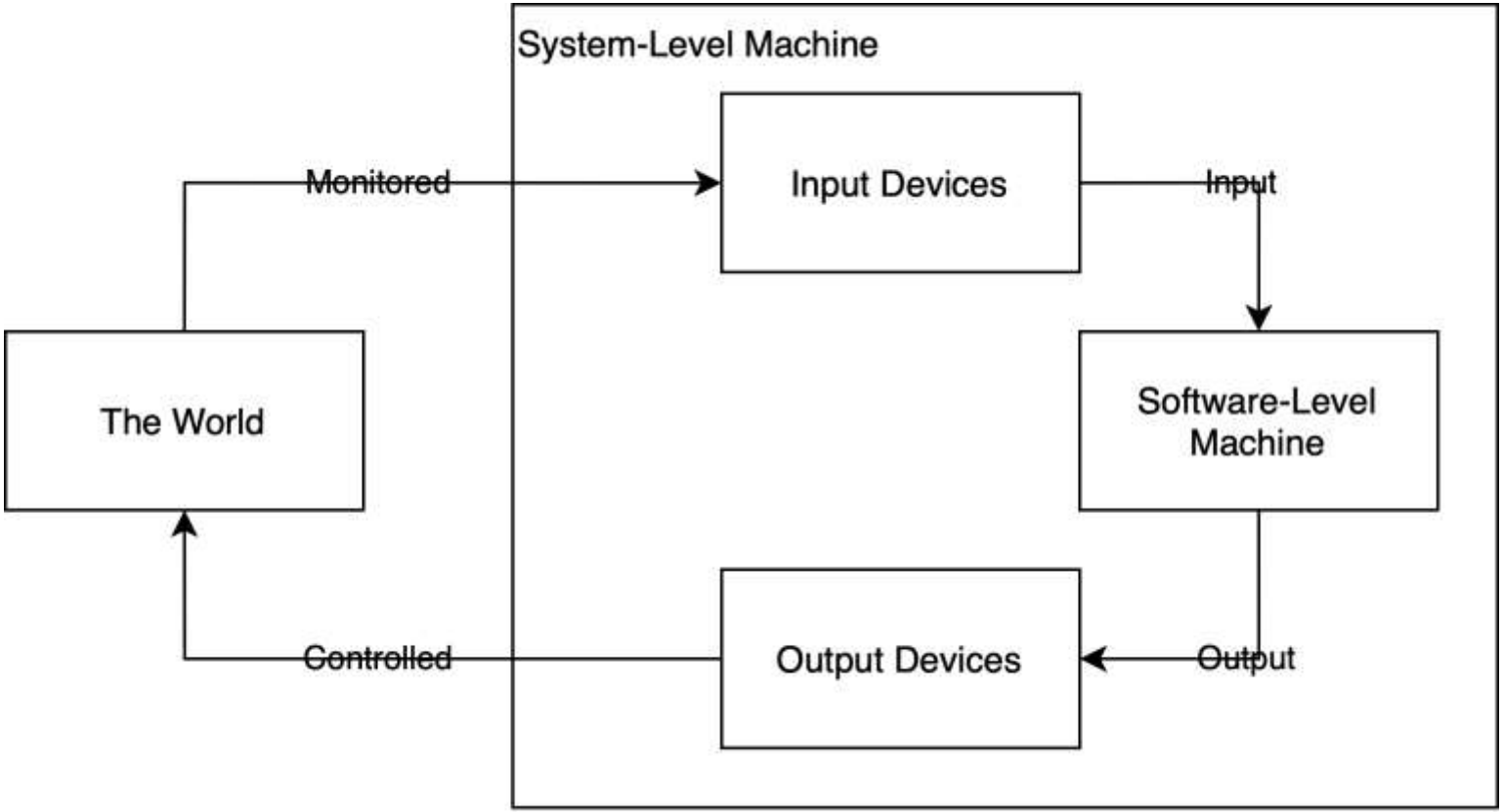
MovingOnRunway
WheelsTurning
StrutsCompressed

Landing_gears__signals

Context Diagrams: A Preview



Synonyms in Systems Engineering: the Four-Variable Model



Actors and Stakeholders

An *actor* is an entity that can perform actions in the World. An actor can initiate events or change some states in the World.

Synonyms: agent, component, element.

Can be a person, an organisation, a device (sensor or actuator) or a software system. The Machine is an actor.

A *stakeholder* is a person or group of persons who have an interest in, or are affected by the Machine.

Non-human actors are not stakeholders.

Not all stakeholders are actors in the system.

Question 1: phenomenon or not?

Which of the following items describe phenomena of the ambulance dispatching system?

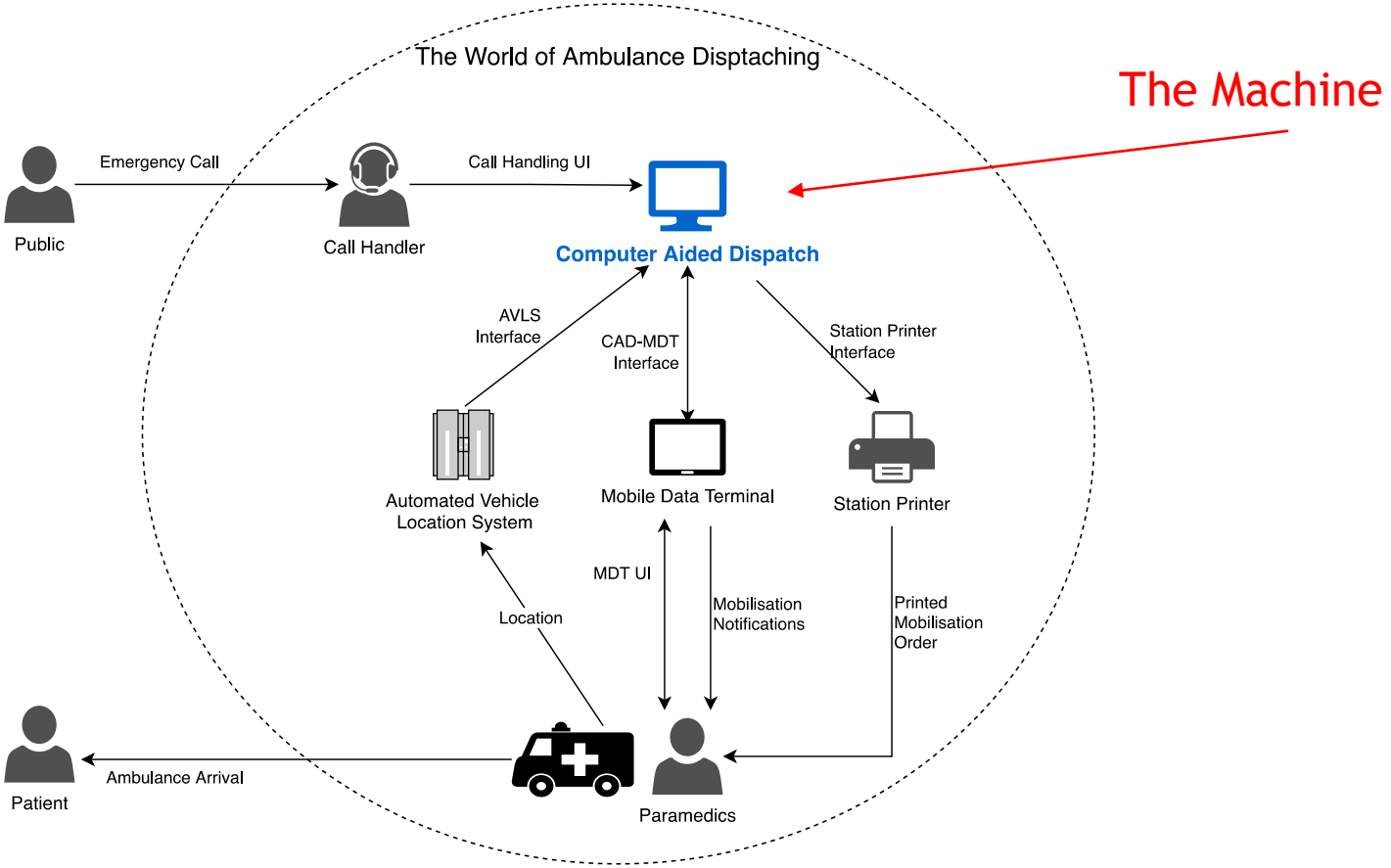
1. An incident occurs at 10 Downing Street.
2. A person calls the emergency service to report the incident.
3. An ambulance arrives at 10 Downing Street.
4. An ambulance.
5. The ambulance response time must be less than 14 minutes.

Use context diagram on next slide

Question 2: Classify the phenomena (World, Input, Output, Machine)

Id	Phenomena	Type (W, I, O, M)
1	An incident occurs at 10 Downing Street.	
2	A person calls the emergency service to report the incident.	
3	A call handler encodes the incident details into the CAD.	
4	Ambulance with ID 123 is at Buckingham Palace.	
5	Ambulance with ID 123 is available.	
6	The function `findNearestAvailableAmbulance()` returns the ambulance ID 123.	
7	The CAD sends mobilisation instructions to ambulance 123's Mobile Data Terminal.	
8	The ambulance crew for ambulance 123 accept the mobilisation on the Mobile Data Terminal.	
9	The ambulance with ID 123 arrives at 10 Downing Street.	

Ambulance Dispatching Context Diagram



2. A Theory of Requirements Engineering – Part II

Recap and what's next

In Part I

- The World and the Machine
- Phenomena
  - world vs machine phenomena
  - machine inputs and outputs
- Actors and Stakeholders

In Part II

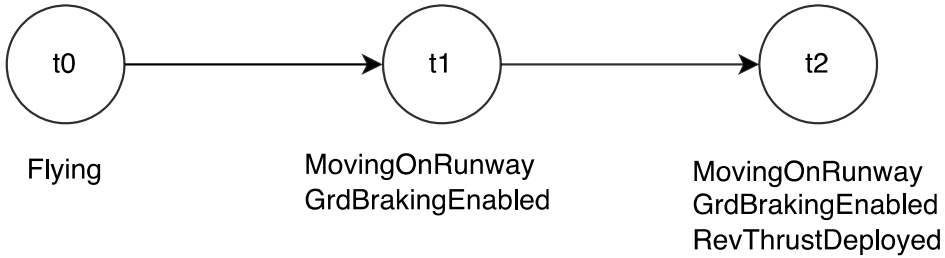
- Behaviours and properties
- Stakeholder goals, machine requirements and domain assumptions
- Requirements correctness: $Req, Dom \vdash Goals$
- Implications for practice

Behaviours

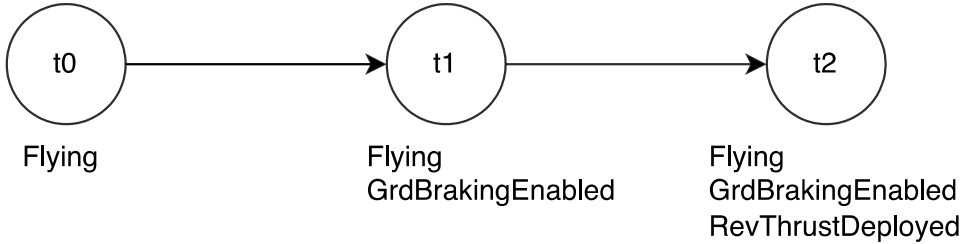
A behaviour is a temporal sequence of phenomena.

Synonym: scenario

A desirable behaviour



An undesirable behaviour



Behavioural Properties

We cannot list and classify all possible behaviours. A property characterize a whole set of behaviours as either desired or undesired.

A behavioural property is a condition on behaviours.

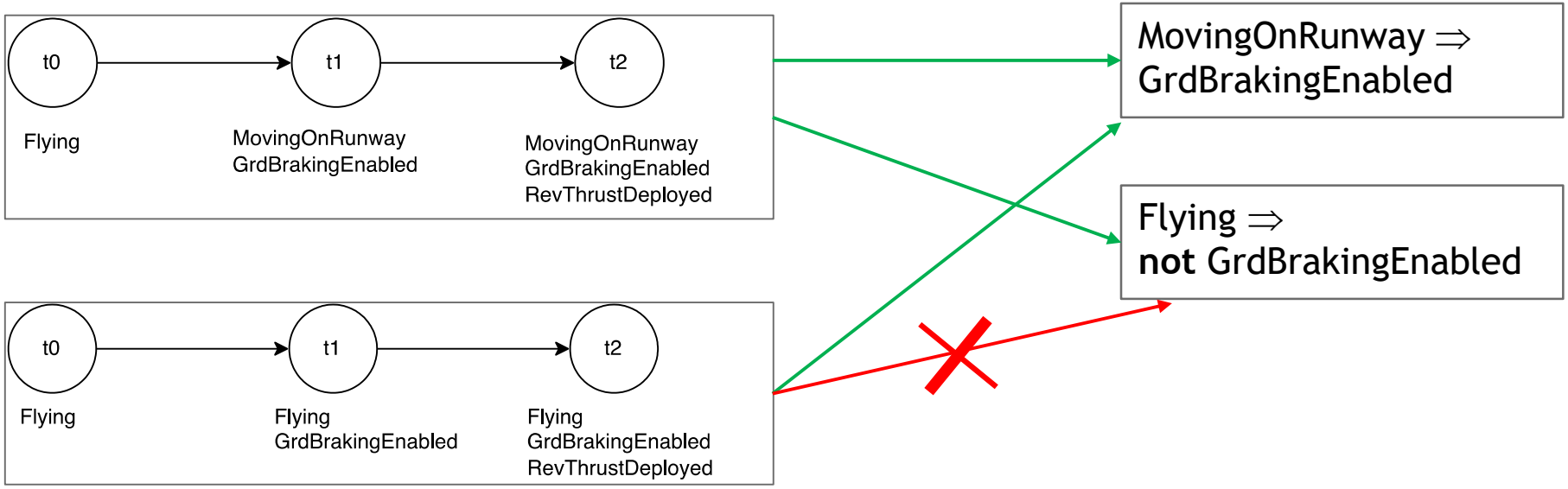
[G1] Ground braking must be enabled when the plane is moving on the runway.

MovingOnRunway $\Rightarrow$ GrdBrakingEnabled

[G2] Ground braking must be disabled when the plane is flying.

Flying $\Rightarrow$ **not** GrdBrakingEnabled

Relation between behaviours and properties



… other behaviours …

Natural Language and Formal Logic

Natural language is the primary language for describing properties.

Logical formulae are optional and usually not shown to stakeholders.

Benefits of logical formulae:

- shorter and easier to read than equivalent NL statement
- no ambiguity in sentence structure
- automated analysis: consistency checking, simulation, verification

Findings of formal analysis are communicated back to stakeholders in NL

Goals, Requirements and Assumptions

Three type of properties:

- Stakeholder goal = desired property of the World
- Machine requirement = desired property of the Machine at its interface with the World
- Domain assumption = property of the World that holds either as laws of nature or through the behavior of actors other than the Machine.

Stakeholder goals

A stakeholder goal is a desired property of the World.
Synonyms: stakeholder need, business goal, customer requirements, user need, etc.

Examples

[G1] Ground braking must be enabled when the plane is moving on the runway.

[G2] Ground braking must be disabled when the plane is flying.

For an ambulance dispatching system:

"An ambulance must arrive at the incident scene within 14 minutes after the first call reporting the incident."

Characteristics:

1. Formulated in terms of World phenomena.
2. Satisfying a goal may involve multiple actors, not just the machine.
3. Not all goals must be satisfied.

Machine requirements

A machine requirement is a desired property of the Machine at its interface with the World.

Synonyms: software requirements, software specification.

Examples

[R1] Ground braking must be enabled when the wheels sensor indicate that the wheels are turning.

$$\text{WheelsPulseOn} \Rightarrow \text{BrakingEnabled}$$

Characteristics:

1. Refer to shared phenomena only (machine inputs and outpus). Cannot refer to non-shared World phenomena.
2. Must be satisfied by the Machine alone. The Machine cannot rely on other World actors.

Domain assumptions

A domain assumption is a property of the World that holds either as a law of nature or because of the behaviours of World actors other than the Machine.

Synonyms: environment assumption, domain property.

Examples

[D1] When the plane is moving on the runway, its wheels are turning.

MovingOnRunway $\Rightarrow$ Wheels Turning

[D2] When the wheels are turning, the wheels sensor signal is pulsing.

WheelsTurning $\Rightarrow$ WheelsPulseOn

Requirements Correctness

A program is correct with respect to a set of requirements. Requirements are correct with respect to a set of stakeholder goals.

The machine requirements *Req* are **correct with respect to the stakeholder goals** *Goals* if, and only if, there exists valid domain assumptions *Dom* such that:

$$Req, Dom \vdash Goals.$$

If the Machine satisfies *Req* and the World satisfies *Dom*, then it can be logically deduced that the World satisfies *Goals*.

The symbol $\vdash$ denotes logical deduction: P $\vdash$ Q means Q can be deduced from P.

Example: “all men are mortal”, “Socrates is a man” $\vdash$ “Socrates is mortal”

Example

Argument that R1 is correct with respect to G1: D1, D2, R1 $\vdash$ G1

[D1] When the plane is moving on the runway, its wheels are turning.

MovingOnRunway $\Rightarrow$ Wheels Turning

[D2] When the wheels are turning, the wheels sensor signal is pulsing.

WheelsTurning $\Rightarrow$ WheelsPulseOn

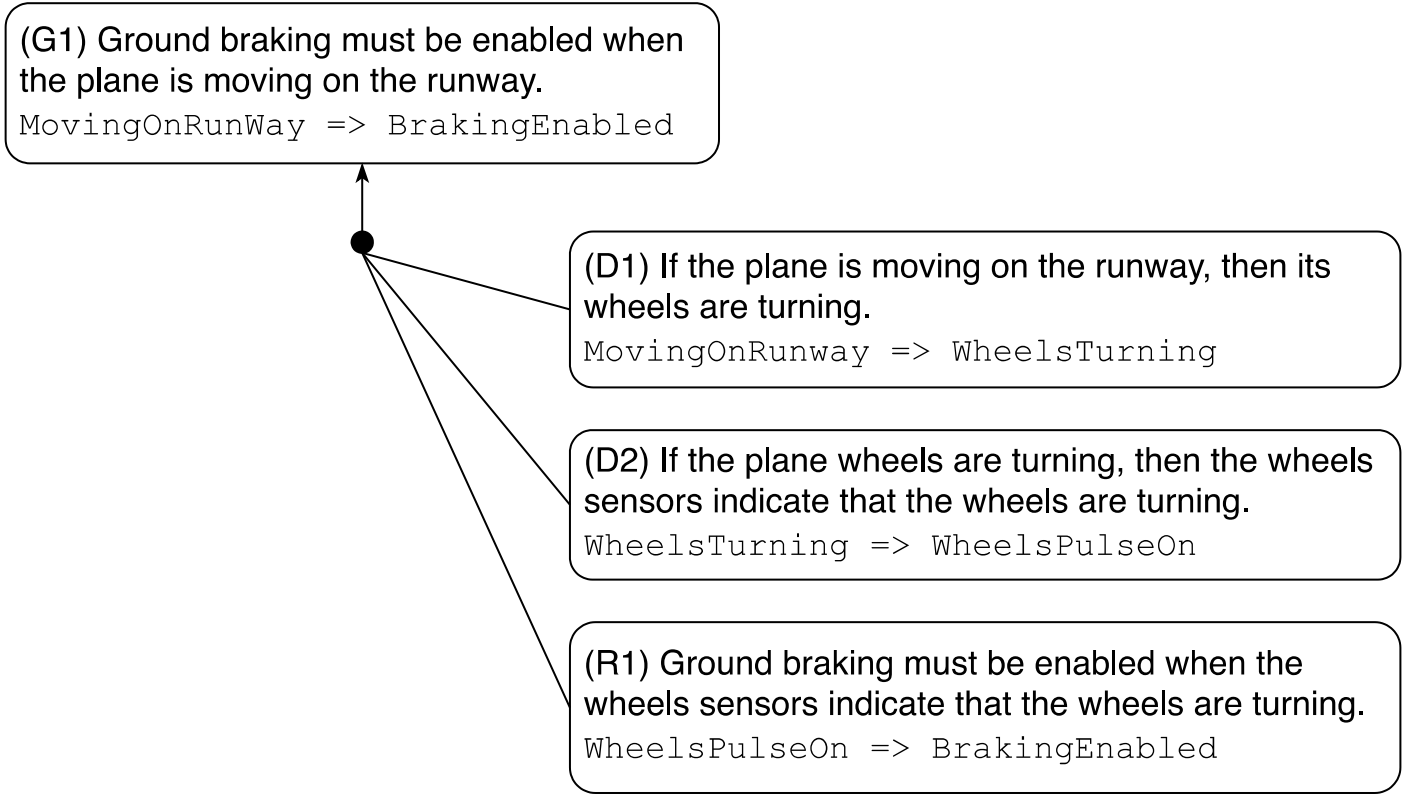
[R1] Ground braking must be enabled when the wheels sensor indicate that the wheels are turning.

WheelsPulseOn $\Rightarrow$ BrakingEnabled

[G1] Ground braking must be enabled when the plane is moving on the runway.

MovingOnRunway $\Rightarrow$ BrakingEnabled

Goal modelling: a preview



Implications for practice

Direct applications in a few RE methods: Problem Frames, KAOS, REVEAL.

The correctness formula suggests three important RE activities:

1. Identifying, formulating and agreeing a set of stakeholder goals (Goals)
2. Identifying valid domain assumptions (Dom)
3. Formulating machine requirements (Req) such that Req, Dom $\vdash$ Goals

These activities are performed concurrently rather than sequentially.

Wider Implications for all RE methods

1. Stakeholder goals vs. Machine requirements
   - What matters to stakeholders are stakeholder goals, not machine requirements.
   - Focusing on machine requirements (e.g. use cases, user stories) is not enough!
2. Machine requirements vs. implementation
   - Not “what” vs “how”. The distinction is about “where”!
   - Requirements are properties of the boundary between the world and the machine; Implementation is the internal working of the Machine.
   - Same idea as use cases and user stories; a radical shift of perspective from our usual coding mindset.
3. Importance of domain assumptions
   - Necessary to transform stakeholder goals into machine requirements.
   - Many errors due to incorrect assumptions (next lecture).

Beyond Requirements Correctness

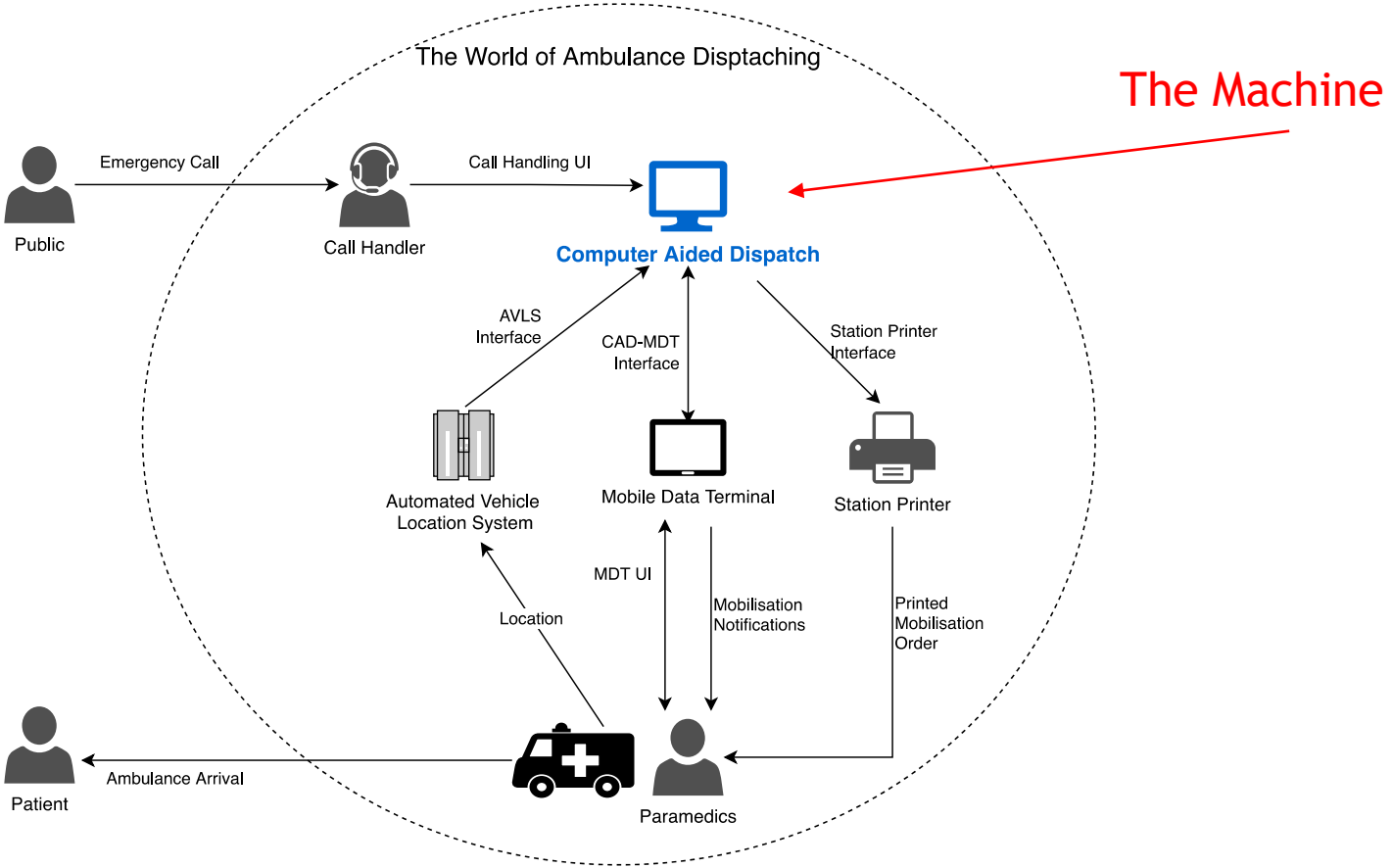
Not a complete theory of requirements engineering:

1. Formulating all requirements and proving their correctness is impossible. We need to stop when the requirements are “good enough”.
2. Stakeholder goals are often conflicting. The correctness formula does not tell us how to analyze tradeoffs between conflicting goals.
3. Goal satisfaction is usually not all-or-nothing (e.g. safety, security, privacy, etc.): we are interested in “how well” the goals are satisfied.
4. Requirements engineering involves exploring alternative ways to satisfy the stakeholder goals.
5. The impacts of alternatives on stakeholder goals are often uncertain.
6. Requirements must deal with technological, budget and time constraints.

Revision Exercise: Goal, Requirement or Domain Assumption?

Id	Property	Type (G, R, D)
1	An ambulance must arrive at incident scene within 14 minutes after the first call.	
2	When the ambulance service receives an emergency call, a call handler will encode the incident's details and location into the CAD.	
3	When a call handler submits a new incident form into the CAD, the CAD displays a list of the nearest available ambulances according to the information it holds about ambulances' status and location.	
4	The Automated Vehicle Location System sends correct up-to-date ambulance locations to the CAD.	
5	The paramedics signal their arrival at the incident scene on the ambulance's Mobile Data Terminal.	
6	The Mobile Data Terminals send updates about ambulance status to the CAD.	

Context Diagram



4. Requirements Errors

Motivation

- Understand requirements errors and how they differ from implementation errors
- Understand three broad types of requirements errors
  - misunderstanding stakeholder goals
  - invalid domain assumptions
  - inadequate requirements due to incorrect reasoning
- Study two system failures partly caused by requirements errors
  - The A320 crash in Warsaw in 1993
  - The 1992 failure of the London Ambulance Service

Engineering Terminology

- A failure is a situation where a product does not work as intended (e.g. the light does not turn on).
- A defect is a flaw in the product (e.g. the light bulb filament is broken).
- An error is a human mistake that contributes to a defect (e.g. you dropped the light bulb).

Machine failure and system failure

- A machine failure is a situation where the machine does not satisfy its requirements.
- A system failure is a situation where the machine and world actors do not satisfy some essential stakeholder goal.

- Machine failure does not necessarily lead to system failure.
- System failure can happen without machine failure: the machine satisfies its requirements but not the stakeholder goal.
- When machine requirements and stakeholder goals are vague, people may disagree about whether a failure has occurred or not.

Implemenation defects vs requirements defects

- An implementation defect (aka bug) is a characteristic of the software code that may cause a machine failure.
- A requirement defect is a characteristic of a requirement description that may cause a machine failure or system failure.

- Requirements that are incorrect with respect to stakeholder goals may cause system failures.
- Requirements descriptions that are vague, incomplete or hard to read can cause machine failures.
    - badly written requirements -> developer misunderstand the real requirement -> implementation defect -> machine failure (from the point of view of the requirements' authors).

Implementation errors vs requirements errors

- An implementation error is a programmer's mistake that causes an implementation defect that may lead to a machine failure.
- A requirement error is a mistake in the definition, communication or understanding of machine requirements that may cause a system failure.

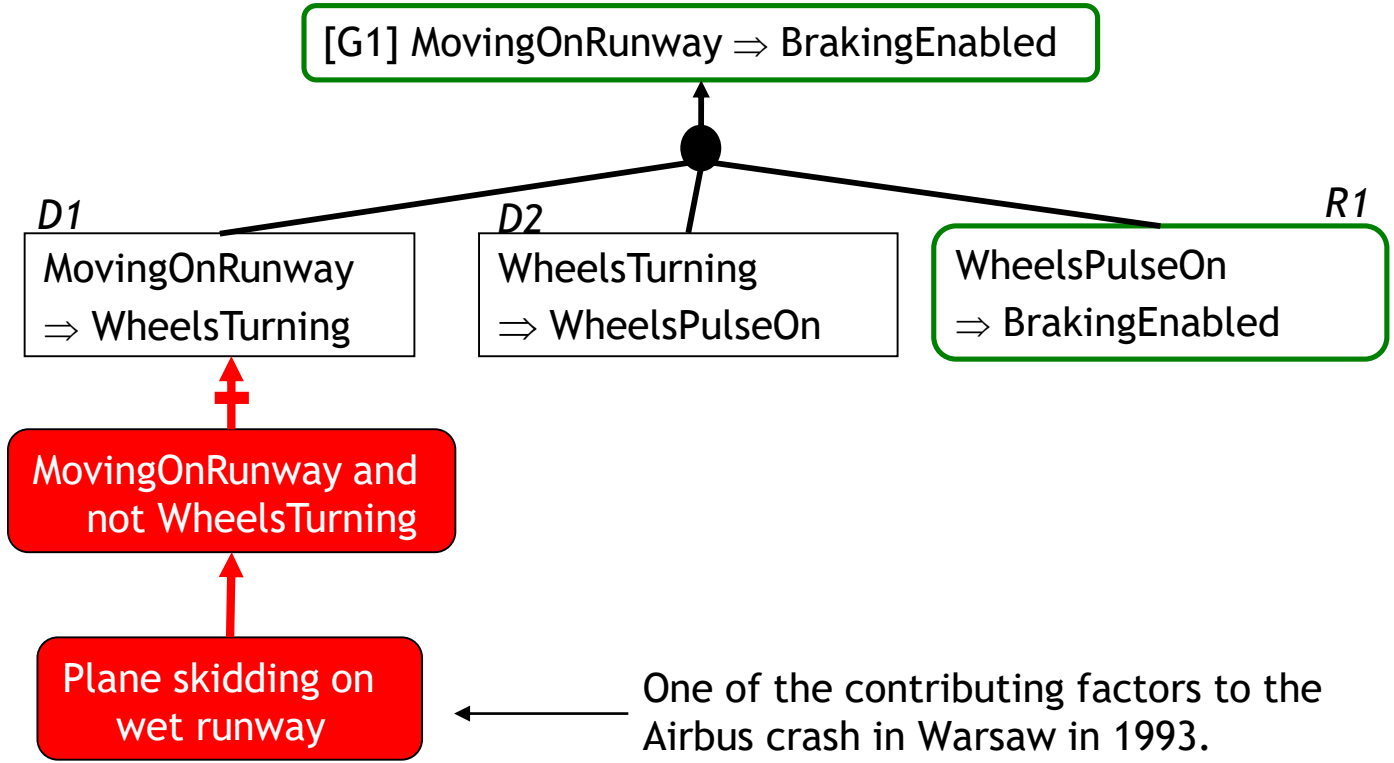
- Implementation errors are coding mistakes: the program does not behave as its developers wanted.
- Some requirements errors are made when discovering, analyzing and writing requirements.
- Some requirements errors are made when coding!
  - Descriptions of requirements are rarely complete and with no ambiguity
  - Developers fill the gaps based on their beliefs about stakeholder goals and context
  - Developers can make requirements errors in that process

Three types of requirements errors

The correctness formula R, D ⊢ G implies three types of requirements errors:

1. Misunderstanding stakeholder goals: G does not describe the essential and agreed stakeholder goals.
2. Invalid assumptions: some assumptions in D do not hold in the World.
3. Inadequate requirements due to incorrect reasoning: the requirements R are insufficient to satisfy G when D is true.

Example 1: Invalid domain assumption



See later lecture on obstacle analysis

Airbus A320 Crash in Warsaw in 1993

- Actual requirements more complex than our simplified example: different req for reverse thrust, ground spoiler, and wheels brakes.
- Software uses information about wheels speed and shock absorbers compression (weights on the wheels).
- Key domain assumption: During landing, both landing gear will be on the ground and their wheels turning
- What happened:
  - landing on a single gear due to pilot's expectation of side wind
  - wheels skidding on wet runway
  - ground spoilers and reverse thrust disabled for 9 seconds

The software satisfied its requirements but not its stakeholders' safety goal.

Example 2: 1992 Failure of the London Ambulance Service

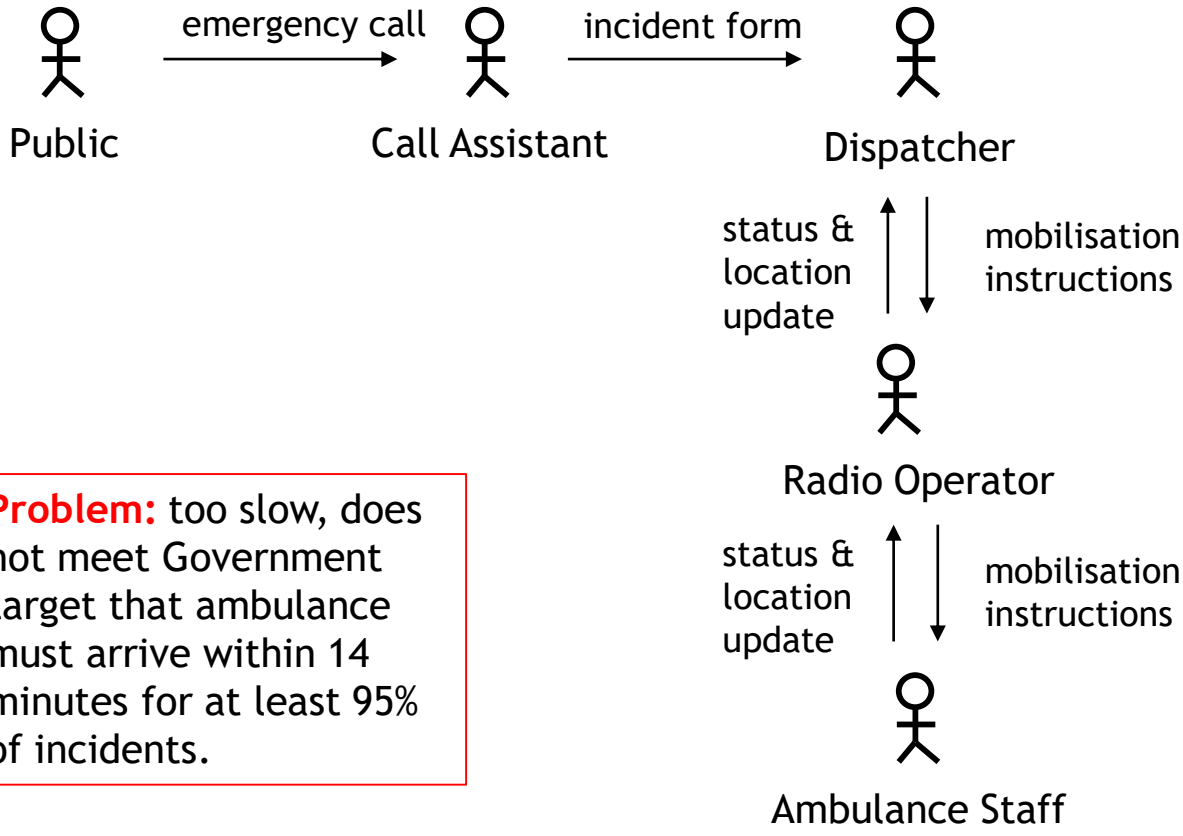
Ambulances should arrive quickly in response to emergency calls

Stakeholder needs

Context

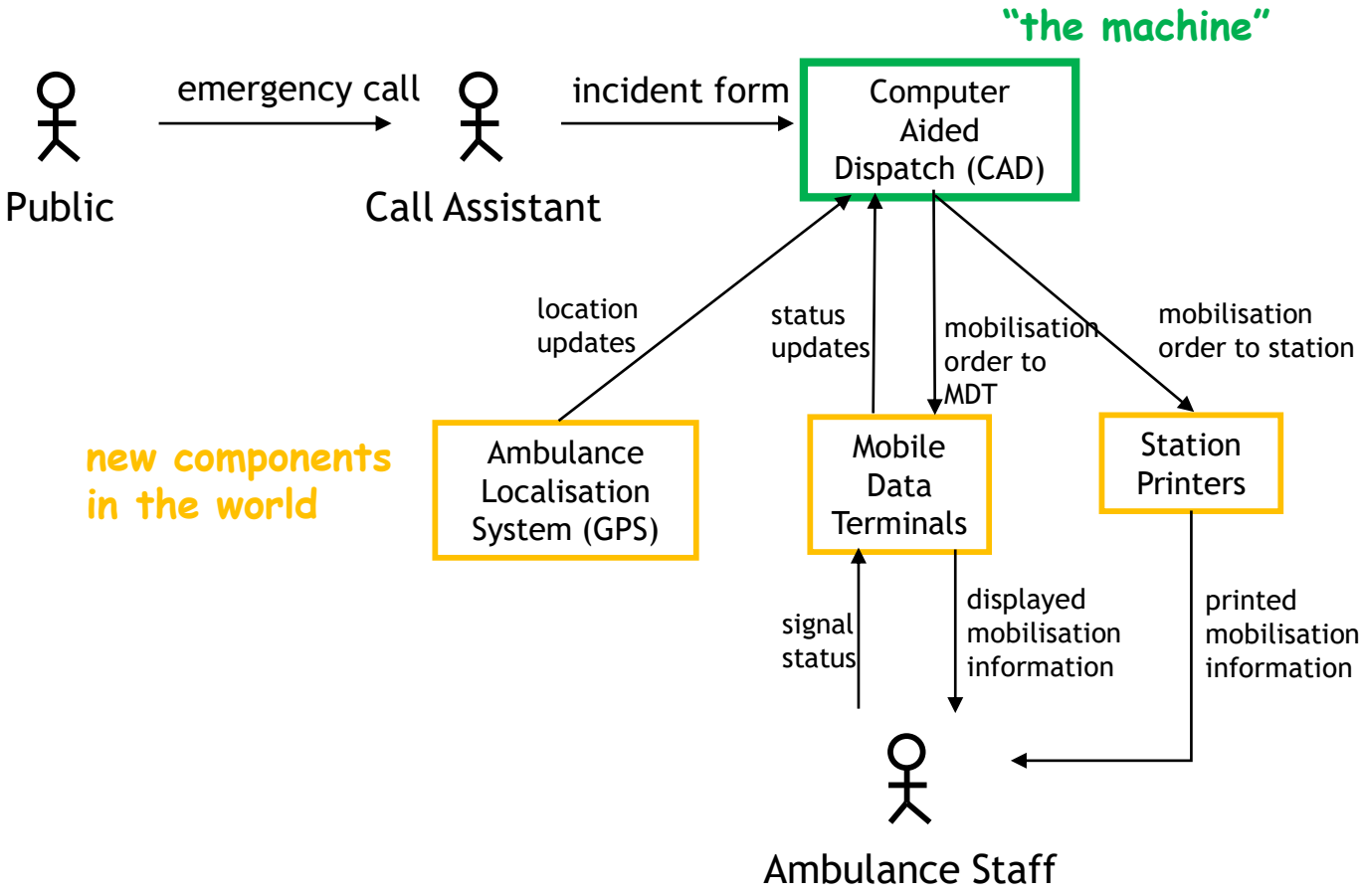
Ambulance Dispatch Software

1992 Paper-based system



Problem: too slow, does not meet Government target that ambulance must arrive within 14 minutes for at least 95% of incidents.

1992 New computer-based system



Goals, Requirements, Assumptions

Stakeholders' Goals

[Goal 1] An ambulance must arrive at incident scene within 14 minutes after the first call.

Domain Assumptions

[Dom1] Call takers encode incident's details and location.

[Dom2] GPS gives correct ambulances' locations.

[Dom3] When an ambulance is allocated to an incident, ambulance crew will drive ambulance to incident location.

[Dom4] When ambulance arrives at location, ambulance crew signal arrival on Mobile Data Terminal

...

Software Requirements

[Req1] When a call assistant submits a new incident form, the CAD identifies the nearest available ambulance based on ambulances' latest status and location information

...

Invalid domain assumptions

[Dom2] GPS gives correct ambulances' locations.

- Inaccurate or missing ambulance locations

[Dom3] When an ambulance is allocated to an incident, ambulance crew will drive ambulance to incident location.

- Ambulance crew take different ambulance than allocated

[Dom4] When ambulance arrives at location, ambulance crew signal arrival on Mobile Data Terminal

- Crew push wrong buttons or push buttons in wrong sequence

Problem: the software was designed with the assumption that it would have perfect information about ambulance status and location at all time.

Outcome: long delays in ambulance responses (up to 11 hours). LAS staff had to revert to manual operations.

Report of the Inquiry

> "On 26 and 27 October 1992 the computer system itself did not fail in a technical sense. Response times did on occasions become unacceptable, but overall the system did what it had been designed to do. However, much of the design had fatal flaws that would, and did, cumulatively lead to all of the symptoms of systems failure;"

The inquiry identified many other causes:

- system was overambitious and had impossible timeline
- LAS management ignored external advice about risks
- procurement rules that emphasize low cost over quality
- project management was inadequate
- misguided decision to deliver full implementation in one phase
- incomplete ownership of the system by its users
- system not tested to a satisfactory level of quality and resilience

Summary

- Software engineers must consider two types of failures
  - machine failures: the machine does not satisfy its requirements
  - system failures: the machine and world actors do not satisfy some essential stakeholder goals
- Three broad type of requirements errors
  1. misunderstanding stakeholder goals
  2. invalid domain assumptions
  3. inadequate machine requirements due to incorrect reasoning
- Invalid domain assumptions are common errors
  - Airbus Ground Braking System Failure 1993
  - London Ambulance Service Failure 1992

Revision Question 1

What type of requirement error contributed to the malfunctioning of the airplane braking system that contributed to the crash in Warsaw in 1993?

1. misunderstanding the stakeholder goals
2. invalid domain assumption
3. inadequate requirements due to incorrect reasoning

Revision Question 2

According the inquiry into the failure of the London Ambulance Service in 1992, which of the following factors contributed to the failure:

1. The system designers did not understand the stakeholders' needs
2. The system relied on invalid assumptions
3. The software requirements were too vague
4. The software had implementation errors
5. The proposed design was too ambitious for the given timeline
6. Risks were ignored

Revision Question 3

1. Imagine the following situation. The requirements for a software system are vague and undocumented. The development team correctly implemented their own understanding of the requirements, but this does not satisfy the stakeholder's needs. Does the situation describe an implementation error, a requirements error, a bit of both, or no error at all?

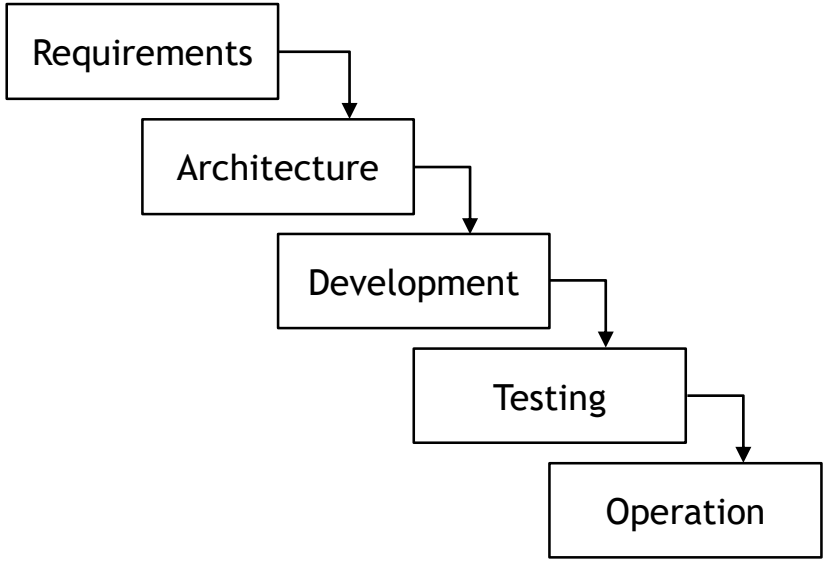
2. Imagine now that the software requirements are documented with precision but are hard to read. The development team misunderstood the requirements. They implemented their own understanding of the requirements without error, but not the requirements as documented. Does the situation describe an implementation error, a requirements error, a bit of both, or no error at all?

5. Requirements in the Software Life-cycle

Motivation

1. Understand how the focus of requirements engineering varies with development process: waterfall vs agile.
2. Understand the relations between
   - Requirements and architecture
   - Requirements and testing

Requirements in the Waterfall Model



“I believe in this concept, but the implementation described above is risky and invites failure.” – Winston W. Royce, 1970.

Waterfall: Assumptions and issues

Assumptions

- Goals, assumptions and requirements are clear from the start
- Nothing will change during development

Issues

- Clients and stakeholders must wait a long time before software is delivered
- Goals, requirements and assumptions are not validated until the whole system is operational
- Changes are difficult and expensive

Consequences for requirements engineers

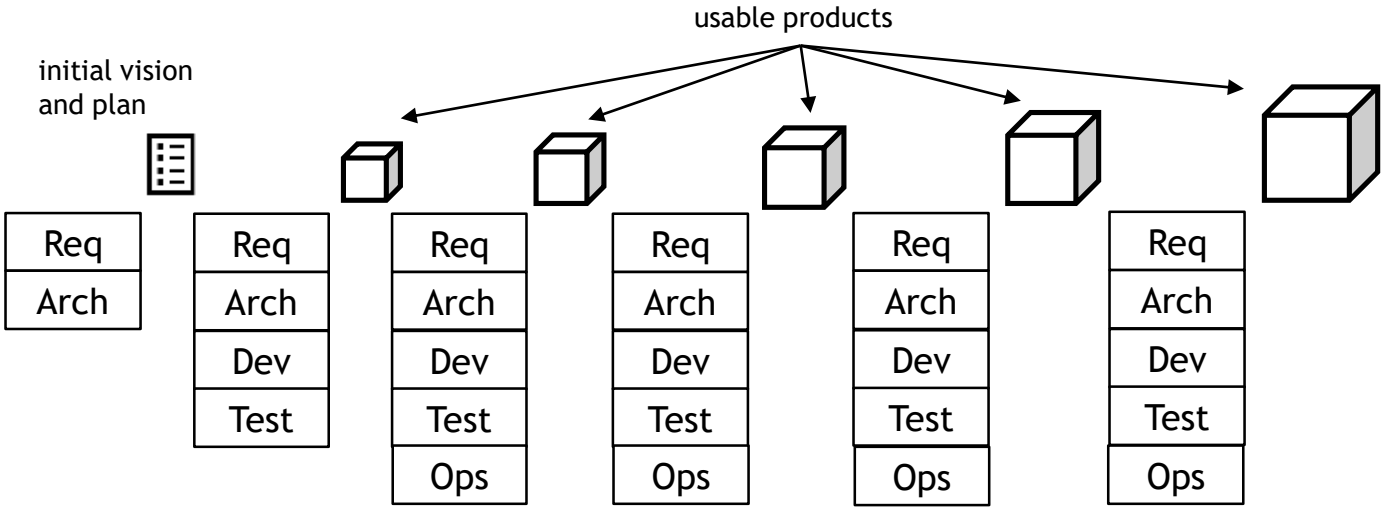
- Must write upfront detailed high-quality requirements descriptions (hard!)
- Later, must deal with changes by writing formal change requests

Requirements in Agile Projects

Agile = iterative + incremental

don't try to make each feature perfect at once; repeatedly improve existing features

don't build everything at once; repeatedly add new features



Agile: benefits and assumptions

Short iterations means

- earlier delivery of value to business and stakeholders
- earlier validation of requirements and assumptions
- easier to adapt to changes

Assumptions

- iterative and incremental delivery is possible and acceptable
  - possible to deliver the whole final system into small useful increments
  - risks of failure in first iterations are acceptable to business, safety and security
- good customer collaboration
- development team skilled in practices that enable frequent changes (continuous integration, automated testing, modularity, refactoring)

Different Focus

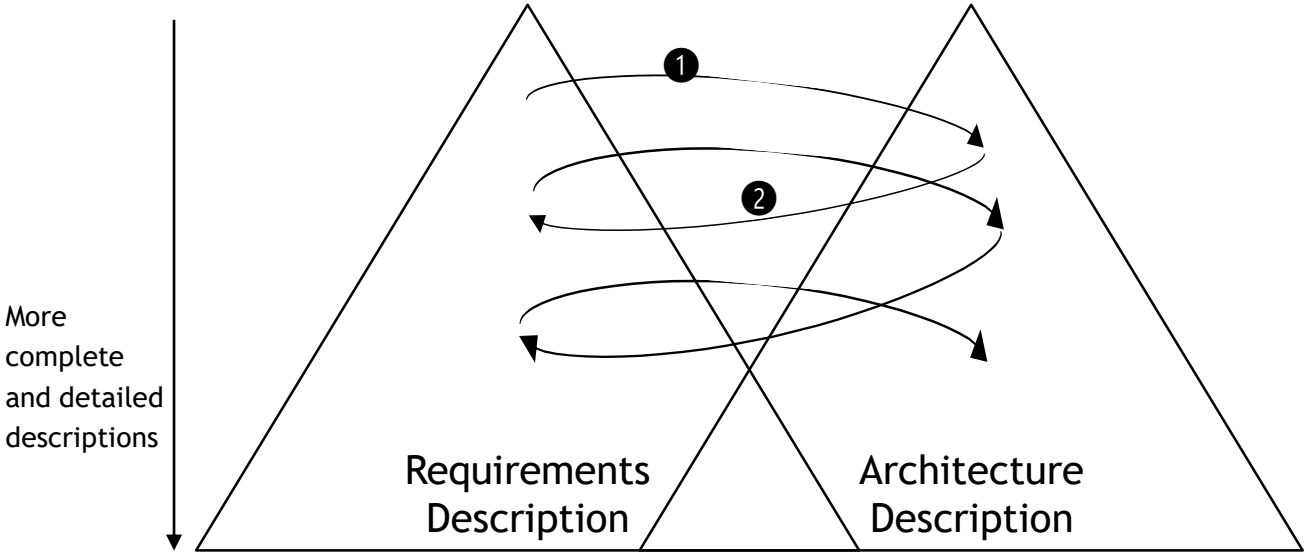
In waterfall

- writing high-quality requirements documents
- managing formal change requests

In agile

- prioritizing what to build next
- just-in-time understanding, analysis and communication of testable requirements

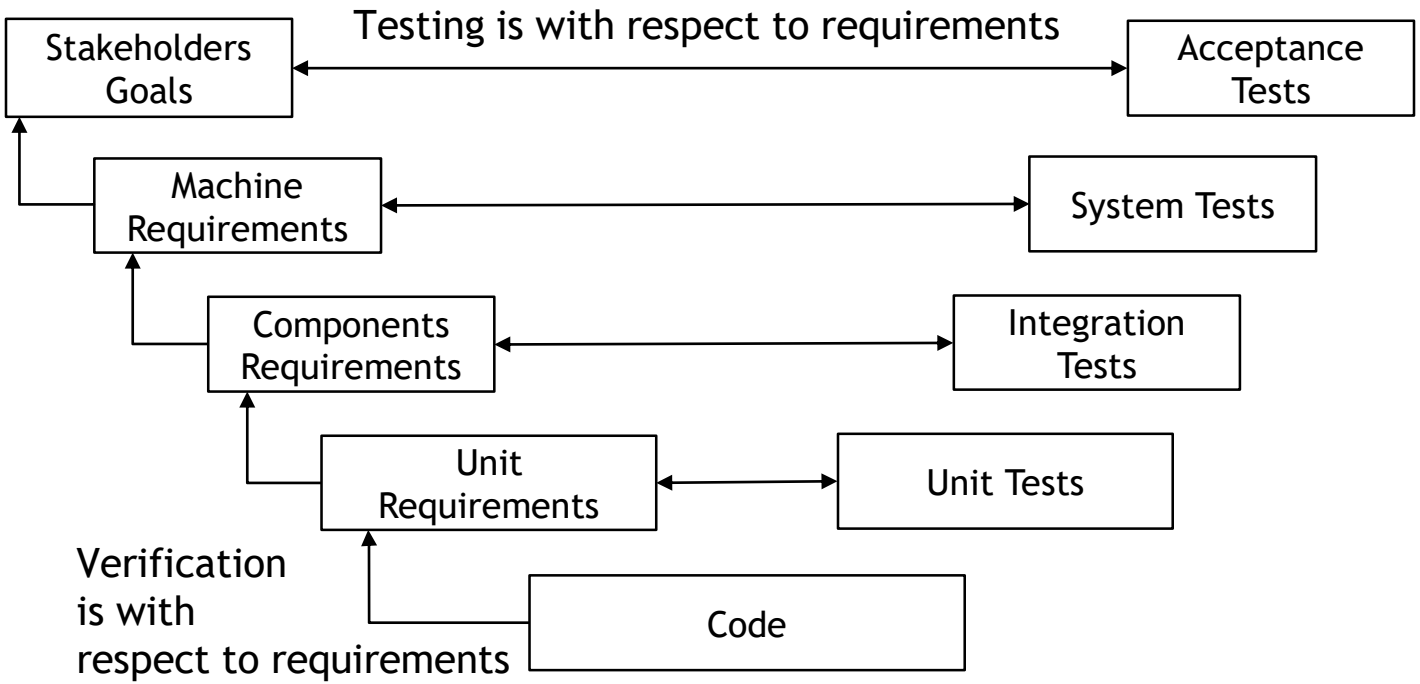
Requirements and Architecture: the Twin Peaks Model



❶ Requirements inform definition of architecture

❷ The architecture reveals costs, tradeoffs, omissions and new opportunities for the envisioned requirements

Requirements and Quality Assurance: the V-Model



Summary

- Requirements in waterfall
  - Importance of high-quality requirements descriptions and controlling change requests
- Requirements in agile development
  - Focus on stakeholder goals through incremental delivery, learning and changes
  - Just-in-time specification of detailed testable machine requirements
- Requirements and Architecture
  - An intertwined process (the Twin Peaks model)
- Requirements and quality assurance
  - Stakeholder goals = ultimate criteria for software quality
  - Machine requirements essential for software testing and verification

Revision Exercises

1. In this course, what do we mean by agile software development?
   A. Writing code without following a plan
   B. Any iterative and incremental development process
2. Imagine you are starting to write a book that will have several chapters. Which of these activities is iterative, which is incremental?
   A. Adding a new chapter
   B. Improving the text of an existing chapter
3. What does the Twin Peaks model mean?
   A. Requirements must be defined entirely before defining the architecture
   B. Requirements and architecture must be defined together
   C. Requirements and architecture must be still like mountains and flow like rivers
4. In this lecture, what did I claim was the ultimate criteria for evaluating the quality of a software system?
   A. How fast it runs compared to other systems
   B. The number of bugs it has in production
   C. How much money it brings to its developers or the organization they work for
   D. The extent to which it meets its stakeholders' goals

   Do you agree with this claim? Why or why not?

6. Requirements in the Wider Context

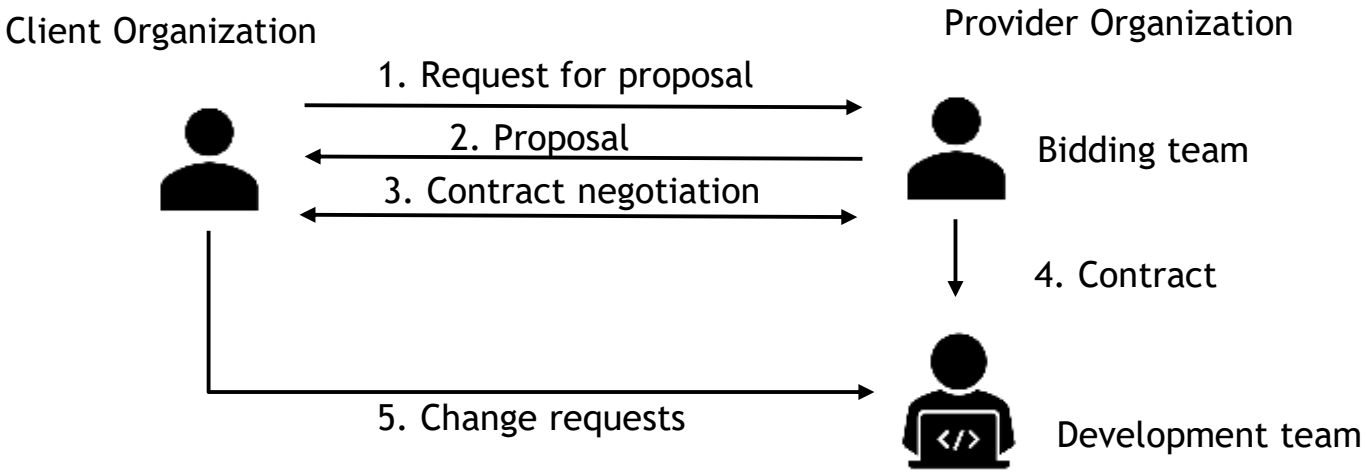
Motivation

Understand how the commercial, legal and social contexts influence requirements engineering activities.

- Client project vs product development
- Greenfield vs brownfield projects
- Regulated environments
- Social responsibility
- Requirements and AI

Requirements in Client Projects

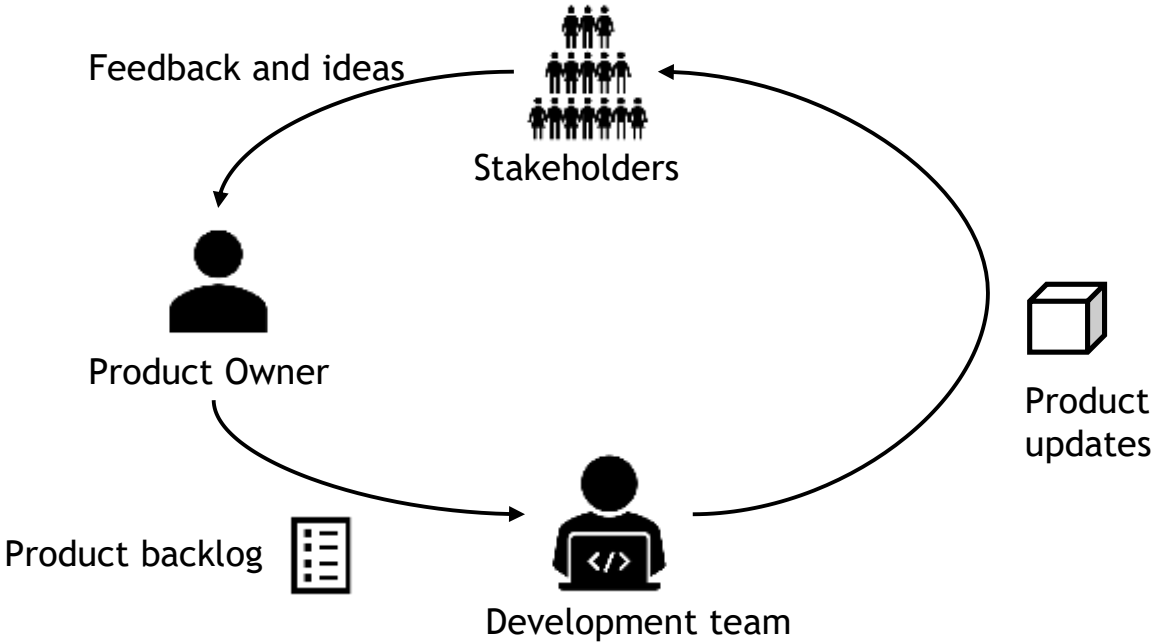
- Software is developed for a specific client
- Client often selects provider through competitive tender
- Project is defined in a legally binding contract
  - Main types: fixed-scope and price vs. time-and-materials
  - With fixed-scope contract, agile development is possible but constrained



Requirements in Product Development

Software is developed for a market

- usually designed and developed internally
- aligns more easily with agile development than client projects



Greenfield vs Brownfield Projects

- Greenfield project = a new software is developed from scratch
- Brownfield project = an existing software system must be modified or replaced

Requirements engineering in brownfield project requires deeper analysis of constraints imposed by the legacy application and the surrounding systems.

Requirements in Regulated Environments

In some industries, software is audited for regulatory compliance (e.g. aerospace, nuclear power plants, medical devices, financial systems).

Imposes the need for comprehensive requirements descriptions and robust traceability between regulations items, software requirements, code, tests and test results.

Requirements and Social Responsibility

Social responsibility = businesses and individuals have a duty to act in the best interest of society as a whole.

Important concerns include

Safety	avoiding physical harms to people and infrastructure
Security	protecting people and assets against harms caused by others
Privacy	protecting people's freedom not to be observed or disturbed.
Environmental Sustainability	protecting the environment by conserving natural resources, avoiding pollution, and reducing contributions to climate change
Fairness	avoiding discrimination, notably based on race, religion, and gender

Ethical standards:

- Professional code of ethics, national and international laws
- Personal code of ethics

A Wicked Problem

Designing socially responsible systems is a “wicked problem”.

A wicked problem = no definitive, universally agreed formulation of the problem to be solved.

- people’s concerns are vague (e.g. privacy, fairness, safety, …)
- people have different perspectives and interests: a positive impact for some people is seen as negative by others
- impacts of software on society are hard to predict and measure

Requirements engineering is the set of activities where software engineers engage with these issues.

Requirements and AI

Artificial Intelligence (AI) is an umbrella term that covers a range of computing techniques that imitate aspects of human intelligence: machine learning, optimization and planning, knowledge representation and formal reasoning.

An *AI system* is a software system where core features are implemented using one or more AI techniques.

Examples:

- self-driving cars, drones and other autonomous vehicles
- virtual assistants on mobile phones
- recommender systems in online stores, social media and streaming services
- credit scoring systems for loan applications
- risk assessment tools in policing and the justice system
- diagnosis systems on medical imaging
- etc.

Requirements Engineering for AI

- Essential for building successful AI systems
  - For guiding initial exploration when business goals are not clear.
  - For understanding all requirements of operational AI system.
- Requirements Tradeoffs and Risk Analysis
  - More important than in other systems because of uncertainty and non-determinism of AI components.
- Fairness, Accountability, Transparency
  - More important than in other system because of wider societal impacts.
- The AI Alignment Problem
  - Risks due to misalignment between AI system's objective function and actual stakeholder goals.

AI for Requirements Engineering

A long-standing ambition: using AI to support or automate the RE process.

Examples of use of AI in RE:

- tools for analyzing quality of requirements statements.
- tools for analyzing user feedback in app review or social media platforms.
- process mining tools to analyze real workflows (event logs).

Current use is marginal, but active research could lead to major changes.

Requirements in Software Engineering with AI

Software engineering increasingly supported by AI for code generation, testing, debugging and optimization.

This might have huge impacts on what software engineers do:

- Probably, less time spent fiddling with code and much more time spent in understanding stakeholder goals, and providing the right “prompts” to the AI coding assistant?
- The adequate specification of functional requirements, acceptance test suites, qualities to be optimized (objective functions) will be essential.

Summary

- Requirements in Client Projects
  - Call for proposals needs good descriptions of goals, context and features
  - Requirements are part of the contract between client and supplier
- Requirements in Product Development
  - Ideal for iterative and incremental approaches
- Requirements in Brownfield projects
  - Importance of understanding the context and constraints
- Requirements in regulated environment
  - Imposes comprehensive requirements documentation and traceability
- Requirements and Social Responsibility
  - Importance of considering all impacts on all members of society
- Requirements and AI
  - Importance of tradeoffs and risks analysis

7. Summary

Working Definition: 6 key ideas

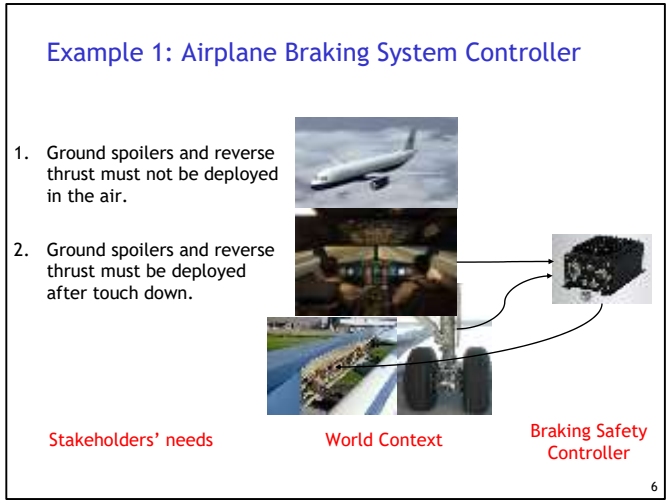
In software engineering, requirements engineering is a set of activities concerned with discovering, analysing, and communicating

- the stakeholders' needs for a software system
- the contexts in which the software system will be used, and
- the externally visible behaviours and quality properties the software must posses to satisfy the stakeholders' needs in a given context.

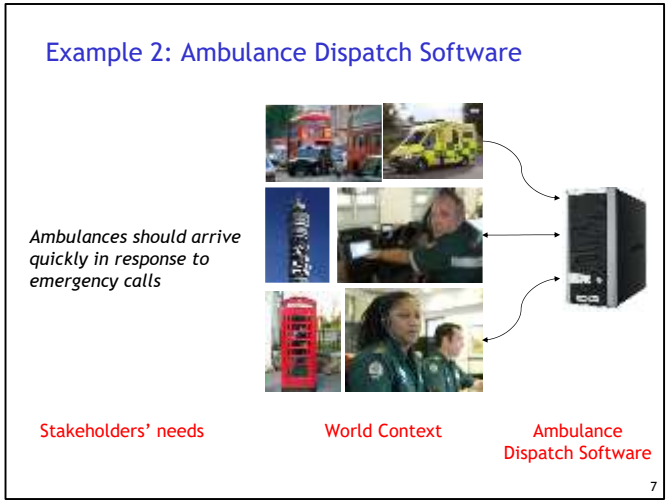
It involves finding suitable tradeoffs between conflicting stakeholder needs, and between stakeholder needs and the capabilities afforded by technologies. It also involves managing changes to stakeholder needs, context and the desired software behaviours and qualities.

Requirements correctness
Req, Dom ⊢ Goals

Learning from failures



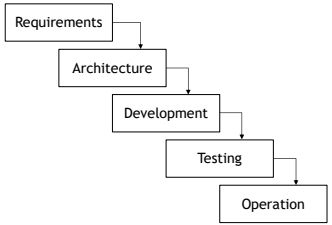
Airbus crash on Warsaw airport runway in 1993



London Ambulance Service Failure in 1992

Requirements in the lifecycle

Requirements in the Waterfall Model



"I believe in this concept, but the implementation described above is risky and invites failure." – Winston W. Royce, 1970.

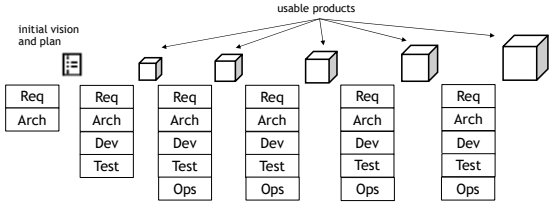
3

Requirements in Agile Projects

Agile = iterative + incremental

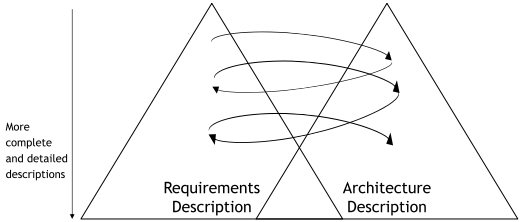
repeatedly improve existing features; don't try to make each feature perfect at once

don't build everything at once; repeatedly add new features



5

Requirements and Architecture: the Twin Peaks Model

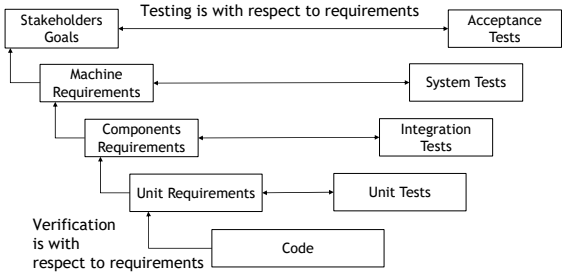


❶ Requirements inform definition of architecture

❷ The architecture reveals costs, tradeoffs, omissions and new opportunities for the envisioned requirements

8

Requirements and Quality Assurance: the V-Model

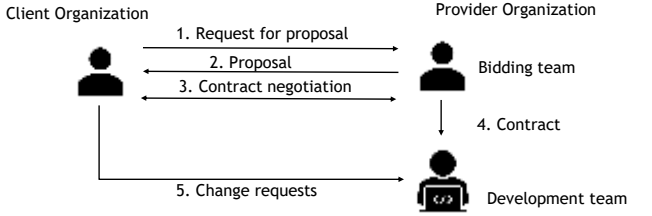


9

Requirements in the Wider Context

Requirements in Client-driven Projects

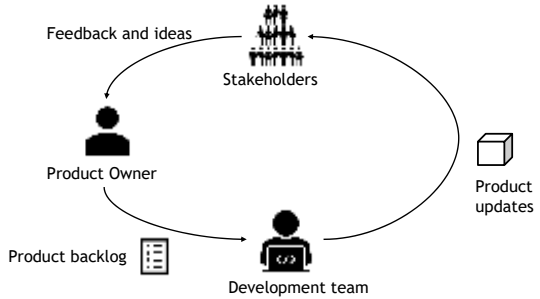
- Software is developed for a specific client
- Client often selects provider through competitive tender
- Project is defined in a legally binding contract
  - Main types: fixed-scope and price vs. time-and-material
  - With fixed-scope contract, agile development is possible but constrained



3

Requirements in Product Development

Software is developed for a market
- usually designed and developed internally
- context is more favourable to agile development



4

Requirements Engineering in 6 Phases