Week:
11 of 12

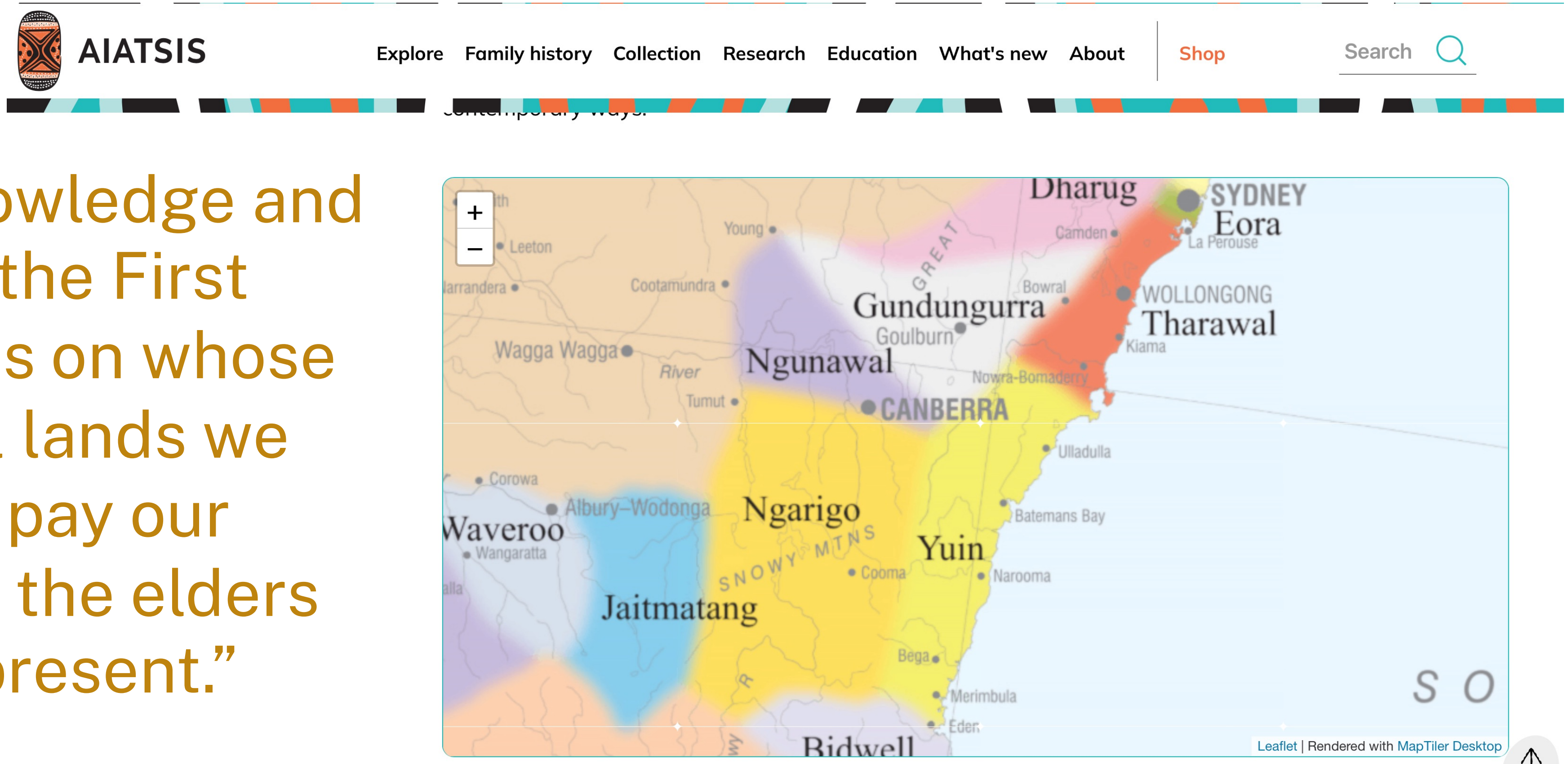
COMP 2120 / COMP 6120

ARCHITECTURE

A/Prof Alex Potanin

ANU Acknowledgment of Country

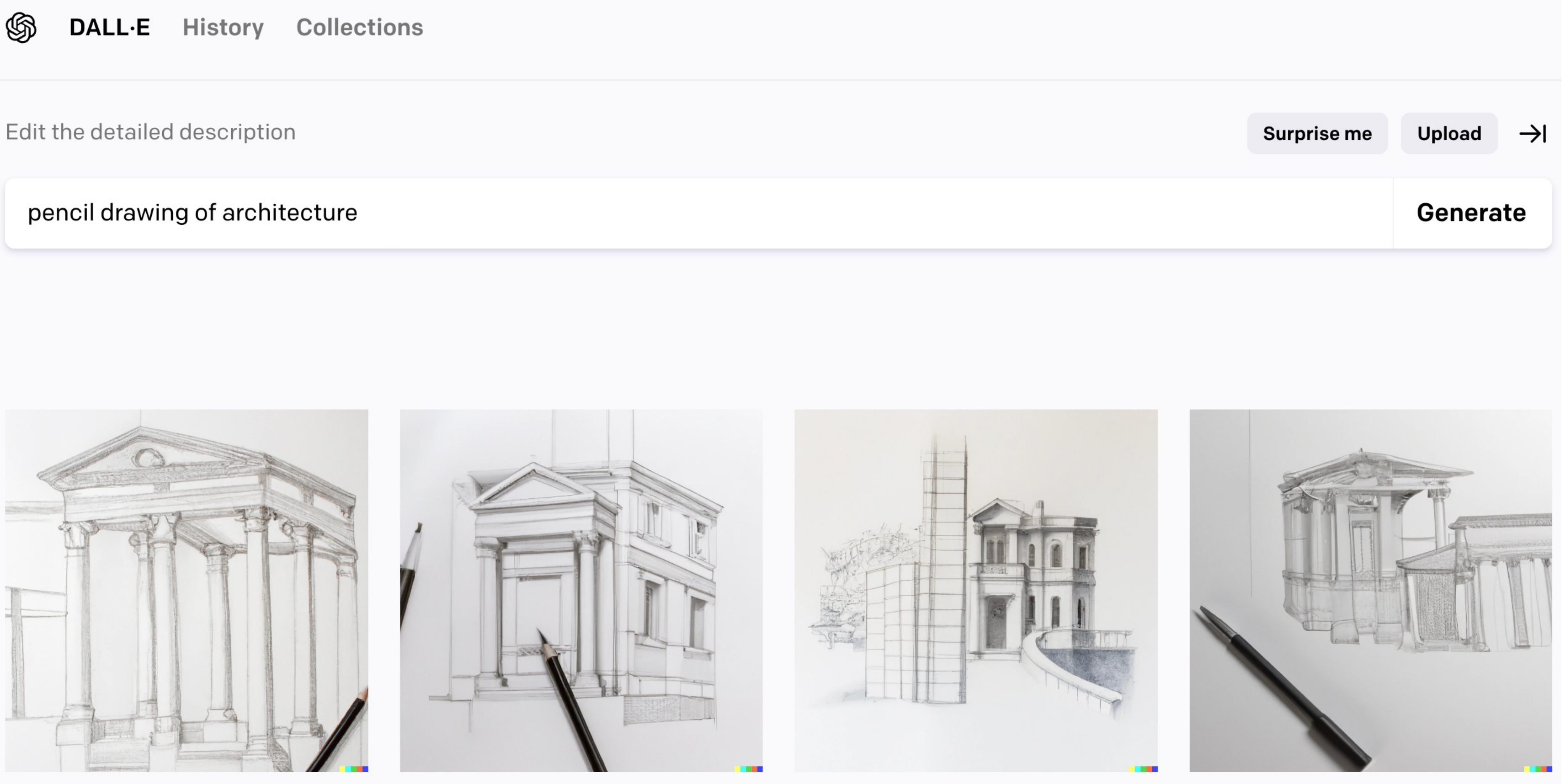
"We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present."



https://aiatsis.gov.au/explore/map-indigenous-austral

Today

- Architecture
- Security
- Wyvern

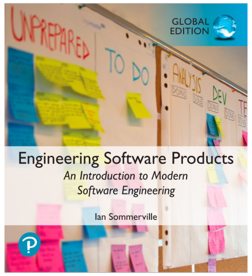


Edit the detailed description

pencil drawing of architecture

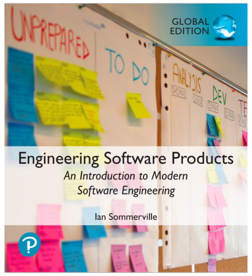
Architecture

Software architecture


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- To create a reliable, secure and efficient product, you need to pay attention to architectural design which includes:
  - its overall organization,
  - how the software is decomposed into components,
  - the server organization
  - the technologies that you use to build the software. The architecture of a software product affects its performance, usability, security, reliability and maintainability.
- There are many different interpretations of the term ‘software architecture’.
  - Some focus on ‘architecture’ as a noun - the structure of a system, and others consider ‘architecture’ to be a verb - the process of defining these structures.

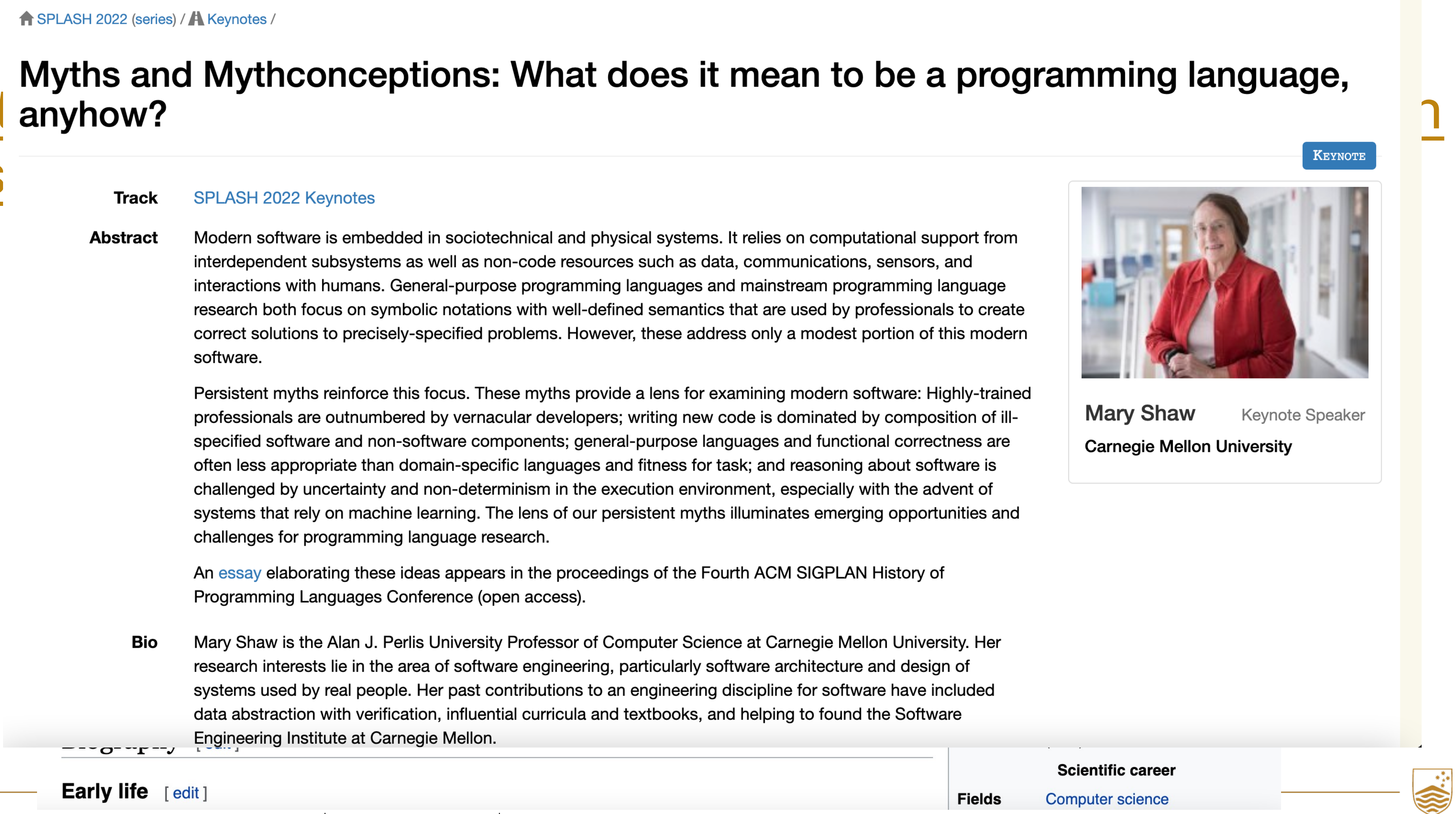
The IEEE definition of software architecture


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Architecture is the fundamental organization of a software system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

Mary Shaw

- htt n tis



SPLASH 2022 (series) / Keynotes /

Myths and Mythconceptions: What does it mean to be a programming language, anyhow?

Track SPLASH 2022 Keynotes

Abstract Modern software is embedded in sociotechnical and physical systems. It relies on computational support from interdependent subsystems as well as non-code resources such as data, communications, sensors, and interactions with humans. General-purpose programming languages and mainstream programming language research both focus on symbolic notations with well-defined semantics that are used by professionals to create correct solutions to precisely-specified problems. However, these address only a modest portion of this modern software.

Persistent myths reinforce this focus. These myths provide a lens for examining modern software: Highly-trained professionals are outnumbered by vernacular developers; writing new code is dominated by composition of ill-specified software and non-software components; general-purpose languages and functional correctness are often less appropriate than domain-specific languages and fitness for task; and reasoning about software is challenged by uncertainty and non-determinism in the execution environment, especially with the advent of systems that rely on machine learning. The lens of our persistent myths illuminates emerging opportunities and challenges for programming language research.

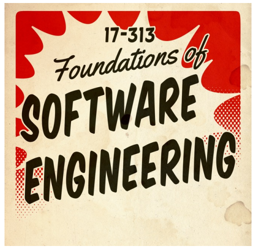
An essay elaborating these ideas appears in the proceedings of the Fourth ACM SIGPLAN History of Programming Languages Conference (open access).

Bio Mary Shaw is the Alan J. Perlis University Professor of Computer Science at Carnegie Mellon University. Her research interests lie in the area of software engineering, particularly software architecture and design of systems used by real people. Her past contributions to an engineering discipline for software have included data abstraction with verification, influential curricula and textbooks, and helping to found the Software Engineering Institute at Carnegie Mellon.

Mary Shaw Keynote Speaker
Carnegie Mellon University

Early life [edit]

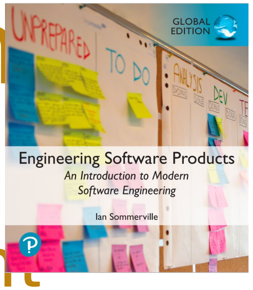
Scientific career	
Fields	Computer science



Quotes from this wonderful paper suggested by Mary Shaw:

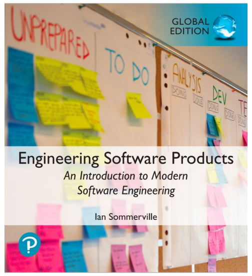
- https://web.engr.oregonstate.edu/~burnett/Reprints/vlhcc16-ML-trialsTribulations.pdf

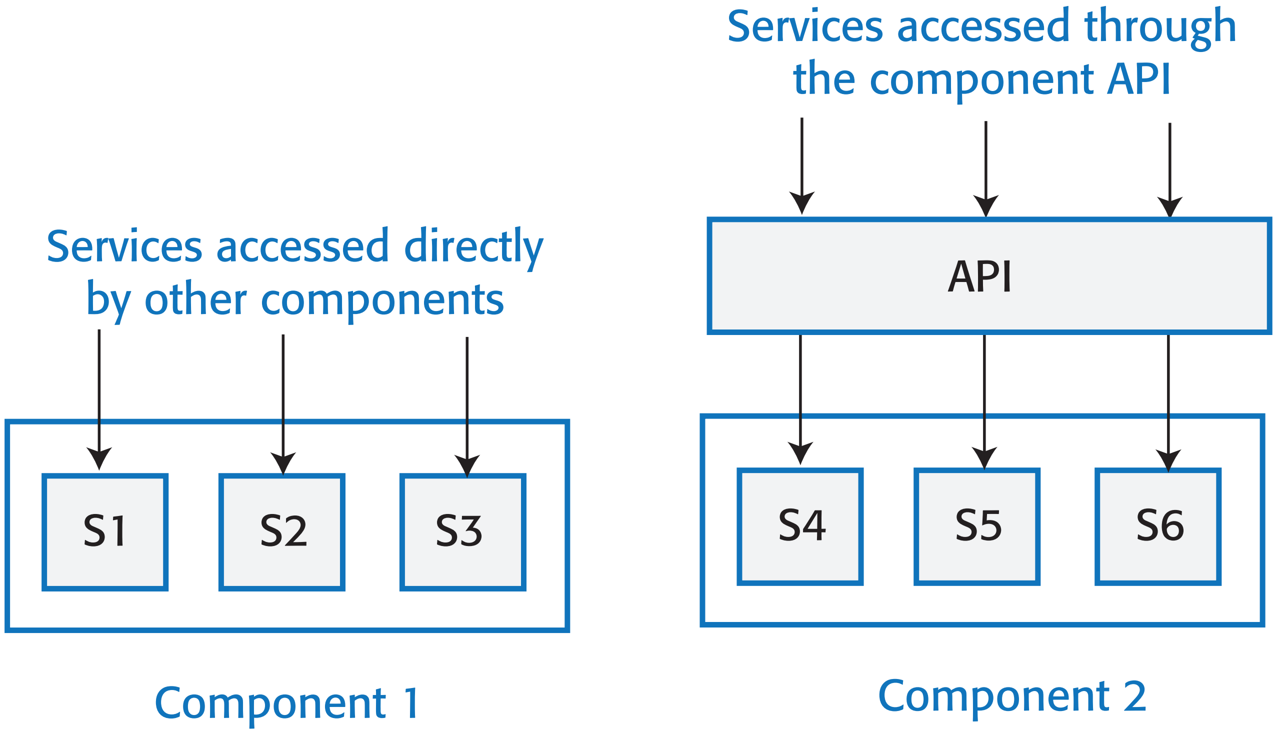
Software architecture and components



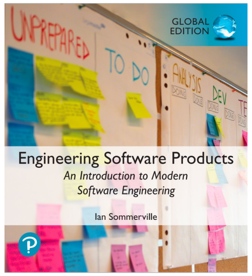
- A component is an element that implements a coherent set of functionality or features.
- Software component can be considered as a collection of one or more services that may be used by other components.
- When designing software architecture, you don't have to decide how an architectural element or component is to be implemented.
- Rather, you design the component interface and leave the implementation of that interface to a later stage of the development process.

Access to services provided by software components



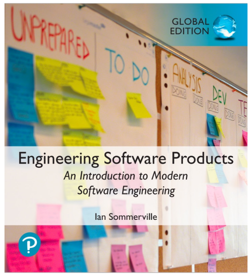

Services accessed through the component API
API
Services accessed directly by other components
S1
S2
S3
S4
S5
S6
Component 1
Component 2

Why is architecture important?



- Architecture is important because the architecture of a system has a fundamental influence on the non-functional system properties (see the next slide).
- Architectural design involves understanding the issues that affect the architecture of your product and creating an architectural description that shows the critical components and their relationships.
- Minimizing complexity should be an important goal for architectural designers.
  - The more complex a system, the more difficult and expensive it is to understand and change.
  - Programmers are more likely to make mistakes and introduce bugs and security vulnerabilities when they are modifying or extending a complex system.

Non-functional system quality attributes


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- *Responsiveness*
  Does the system return results to users in a reasonable time?

- *Reliability*
  Do the system features behave as expected by both developers and users?

- *Availability*
  Can the system deliver its services when requested by users?

- *Security*
  Does the system protect itself and users' data from unauthorized attacks and intrusions?

- *Usability*
  Can system users access the features that they need and use them quickly and without errors?

- *Maintainability*
  Can the system be readily updated and new features added without undue costs?

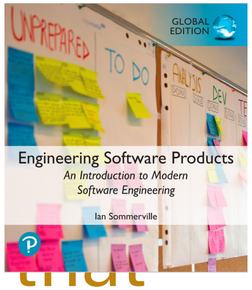
- *Resilience*
  Can the system continue to deliver user services in the event of partial failure or external attack?

The influence on architecture of syste security

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- *A centralized security architecture*
  In the Star Wars prequel Rogue One (https://en.wikipedia.org/wiki/Rogue_One), the evil Empire have stored the plans for all of their equipment in a single, highly secure, well-guarded, remote location. This is called a centralized security architecture. It is based on the principle that if you maintain all of your information in one place, then you can apply lots of resources to protect that information and ensure that intruders can't get hold of it.
- Unfortunately (for the Empire), the rebels managed to breach their security. They stole the plans for the Death Star, an event which underpins the whole Star Wars saga. In trying to stop them, the Empire destroyed their entire archive of system documentation with who knows what resultant costs. Had the Empire chosen a distributed security architecture, with different parts of the Death Star plans stored in different locations, then stealing the plans would have been more difficult. The rebels would have had to breach security in all locations to steal the complete Death Star blueprints.

Centralized security architectures

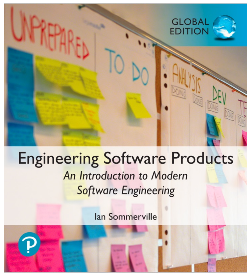

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

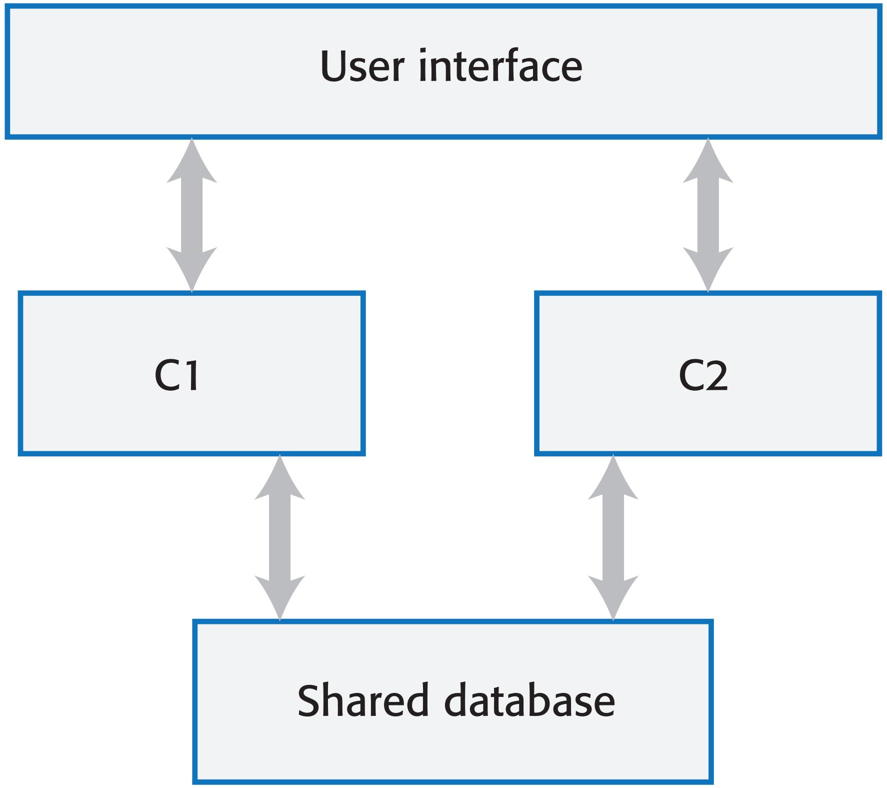
- The benefits of a centralized security architecture are that it is easier to design and build protection and that the protected information can be accessed more efficiently.
- However, if your security is breached, you lose everything.
- If you distribute information, it takes longer to access all of the information and costs more to protect it.
- If security is breached in one location, you only lose the information that you have stored there.

Maintainability and performance

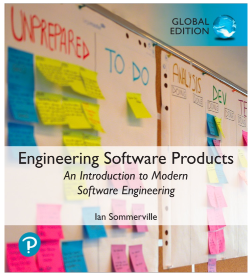
- Next slide shows a system with two components (C1 and C2) that share a common database.
  - Assume C1 runs slowly because it has to reorganize the information in the database before using it.
  - The only way to make C1 faster might be to change the database. This means that C2 also has to be changed, which may, potentially, affect its response time.
- The slide after the next shows a different architecture is used where each component has its own copy of the parts of the database that it needs.
  - If one component needs to change the database organization, this does not affect the other component.
- However, a multi-database architecture may run more slowly and may cost more to implement and change.
  - A multi-database architecture needs a mechanism (component C3) to ensure that the data shared by C1 and C2 is kept consistent when it is changed.

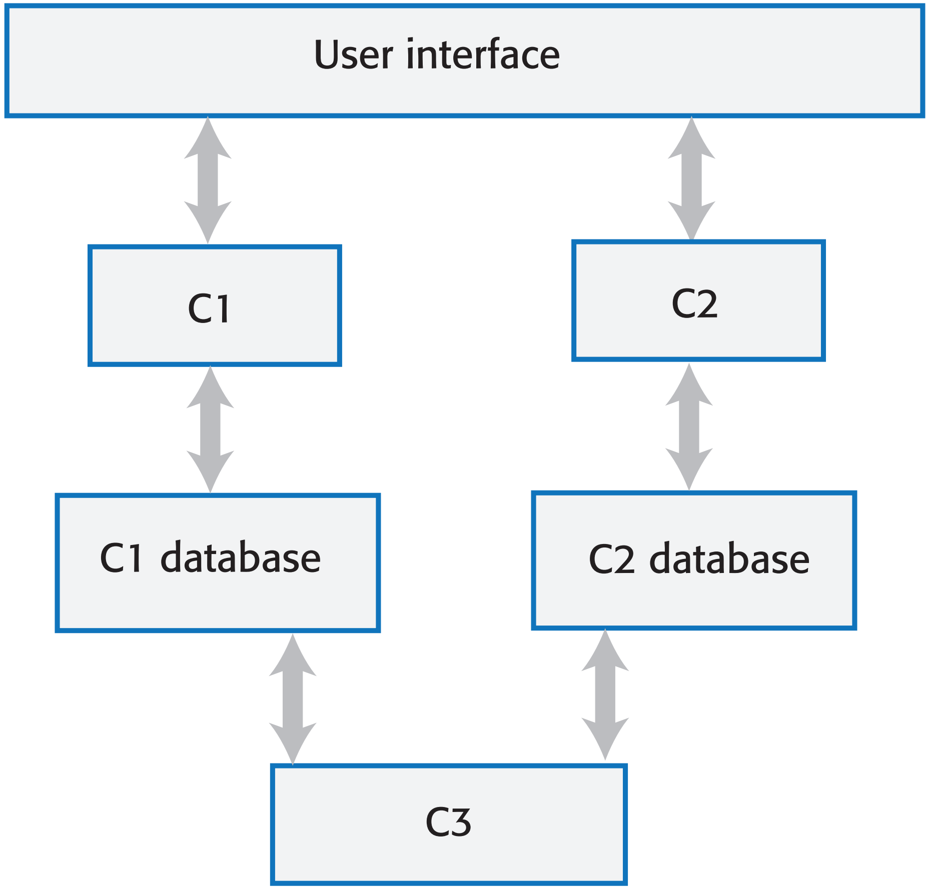
Shared database architecture





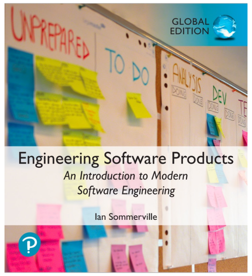
Multiple database architecture

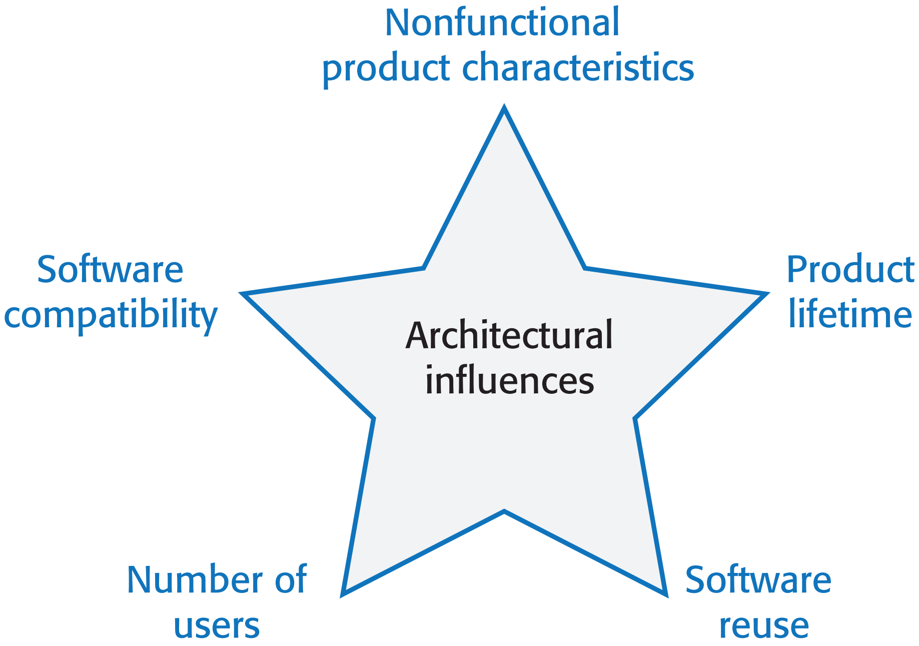

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville



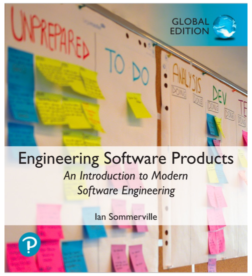
Database reconciliation

Issues that influence architectural decisions





The importance of architectural design issues



- *Nonfunctional product characteristics*
  Nonfunctional product characteristics such as security and performance affect all users. If you get these wrong, your product will is unlikely to be a commercial success. Unfortunately, some characteristics are opposing, so you can only optimize the most important.

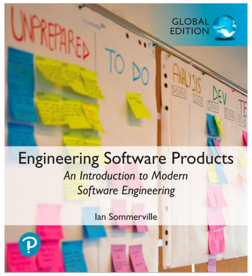
- *Product lifetime*
  If you anticipate a long product lifetime, you will need to create regular product revisions. You therefore need an architecture that is evolvable, so that it can be adapted to accommodate new features and technology.

- *Software reuse*
  You can save a lot of time and effort, if you can reuse large components from other products or open-source software. However, this constrains your architectural choices because you must fit your design around the software that is being reused.

- *Number of users*
  If you are developing consumer software delivered over the Internet, the number of users can change very quickly. This can lead to serious performance degradation unless you design your architecture so that your system can be quickly scaled up and down.

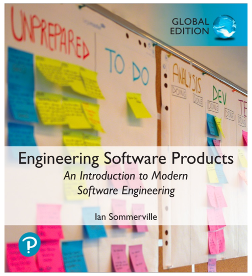
- *Software compatibility*
  For some products, it is important to maintain compatibility with other software so that users can adopt your product and use data prepared using a different system. This may limit architectural choices, such as the database software that you can use.

Trade off: Maintainability vs performance


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- System maintainability is an attribute that reflects how difficult and expensive it is to make changes to a system after it has been released to customers.
  - You improve maintainability by building a system from small self-contained parts, each of which can be replaced or enhanced if changes are required.
- In architectural terms, this means that the system should be decomposed into fine-grain components, each of which does one thing and one thing only.
  - However, it takes time for components to communicate with each other. Consequently, if many components are involved in implementing a product feature, the software will be slower.

Trade off: Security vs usability


GLOBAL EDITION

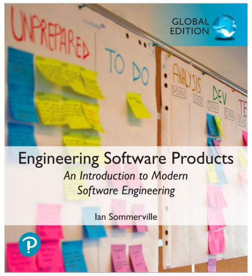
Engineering Software Products

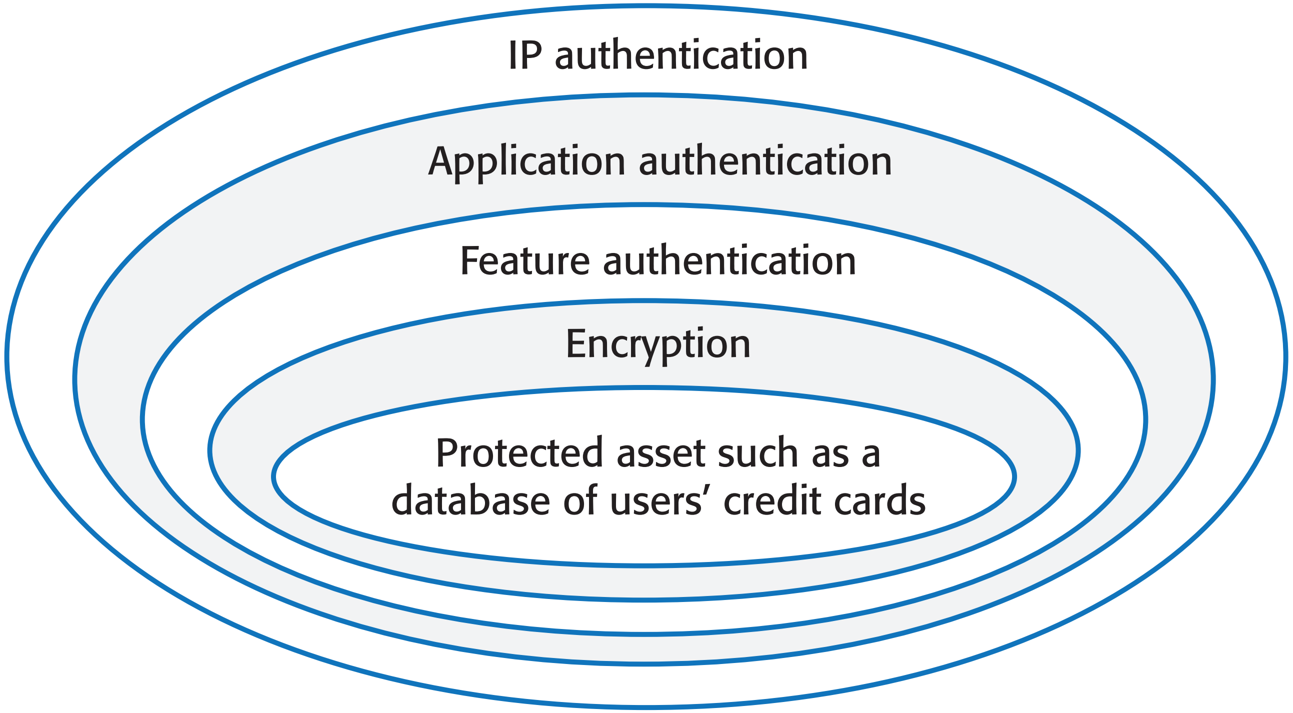
An Introduction to Modern Software Engineering

Ian Sommerville

- You can achieve security by designing the system protection as a series of layers (next slide).
    - An attacker has to penetrate all of those layers before the system is compromised.
- Layers might include system authentication layers, a separate critical feature authentication layer, an encryption layer and so on.
- Architecturally, you can implement each of these layers as separate components so that if one of these components is compromised by an attacker, then the other layers

Authentication layers





IP authentication

Application authentication

Feature authentication

Encryption

Protected asset such as a database of users' credit cards

Usability issues

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

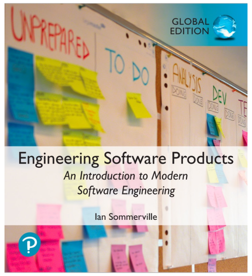
- A layered approach to security affects the usability of the software.
  - Users have to remember information, like passwords, that is needed to penetrate a security layer. Their interaction with the system is inevitably slowed down by its security features.
  - Many users find this irritating and often look for work-arounds so that they do not have to re-authenticate to access system features or data.
- To avoid this, you need an architecture:
  - that doesn’t have too many security layers,
  - that doesn’t enforce unnecessary security,
  - that provides helper components that reduce the load on users.

Trade off: Availability vs time-to-mark

GLOBAL EDITION
UNPREPARED
TO DO
ANALYSIS
DEV
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Availability is particularly important in enterprise products, such as products for the finance industry, where 24/7 operation is expected.
- The availability of a system is a measure of the amount of 'uptime' of that system.
  - Availability is normally expressed as a percentage of the time that a system is available to deliver user services.
- Architecturally, you achieve availability by having redundant components in a system.
  - To make use of redundancy, you include sensor components that detect failure, and switching components that switch operation to a redundant component when a failure is detected.
- Implementing extra components takes time and increases the cost of system development. It adds complexity to the system and therefore increases the chances of introducing bugs and vulnerabilities

Architectural design questions


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- How should the system be organized as a set of architectural components, where each of these components provides a subset of the overall system functionality?
  - The organization should deliver the system security, reliability and performance that you need.
- How should these architectural components be distributed and communicate with each other?
- What technologies should you use in building the system and what components should be reused?

Component organization

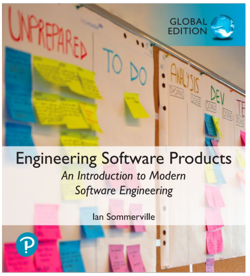
Engineering Software Products

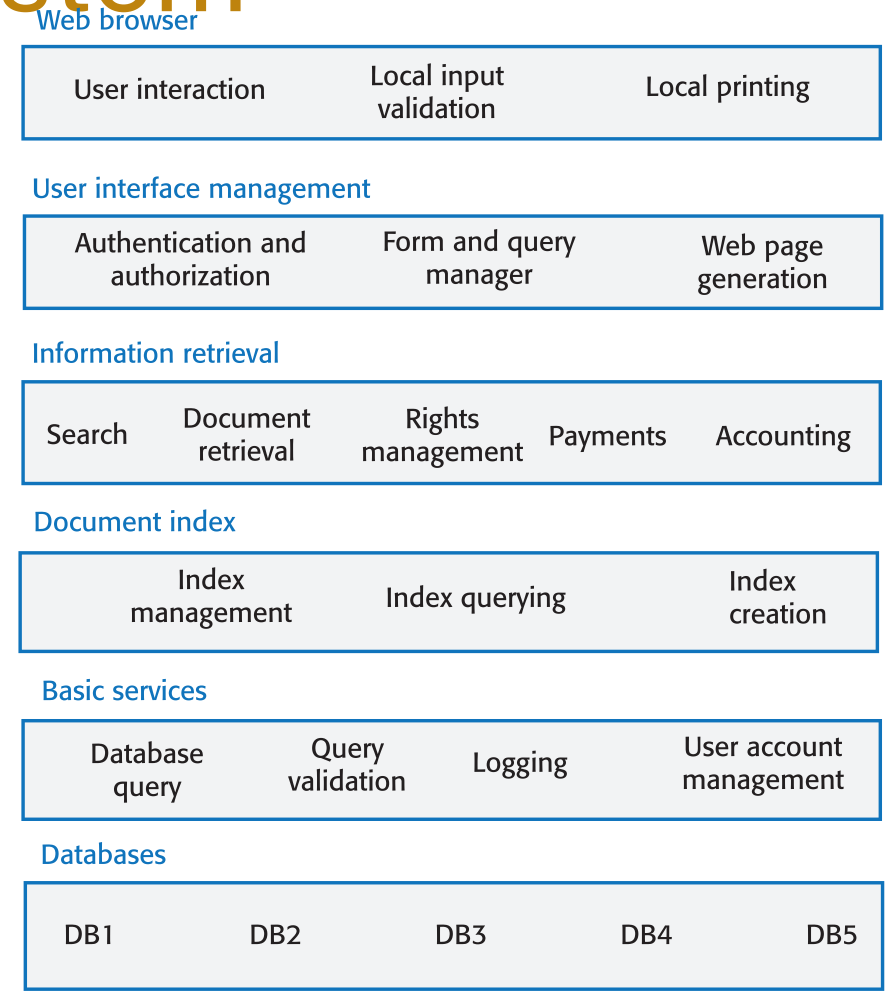
An Introduction to Modern Software Engineering

Ian Sommerville

- Abstraction in software design means that you focus on the essential elements of a system or software component without concern for its details.
- At the architectural level, your concern should be on large-scale architectural components.
- Decomposition involves analysing these large-scale components and representing them as a set of finer-grain components.
- Layered models are often used to illustrate how a system is composed of components.

An architectural model of a documen retrieval system


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville



Web browser

User interaction | Local input validation | Local printing

User interface management

Authentication and authorization | Form and query manager | Web page generation

Information retrieval

Search | Document retrieval | Rights management | Payments | Accounting

Document index

Index management | Index querying | Index creation

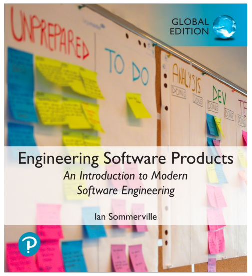
Basic services

Database query | Query validation | Logging | User account management

Databases

DB1 | DB2 | DB3 | DB4 | DB5

Architectural complexity


GLOBAL EDITION

Engineering Software Products

An Introduction to Modern Software Engineering

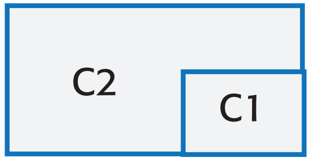
Ian Sommerville

- Complexity in a system architecture arises because of the number and the nature of the relationships between components in that system.
- When decomposing a system into components, you should try to avoid unnecessary software complexity.
  - *Localize relationships*
    If there are relationships between components A and B, these are easier to understand if A and B are defined in the same module.
  - *Reduce shared dependencies*
    Where components A and B depend on some other component or data, complexity increases because changes to the shared component mean you have to understand how these changes affect both A and B.
- It is always preferable to use local data wherever possible and to avoid sharing data if you can.

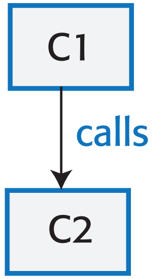
Examples of component relationships

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

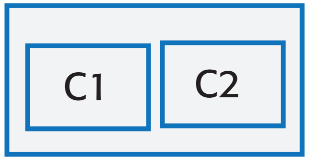
C1 is-part-of C2



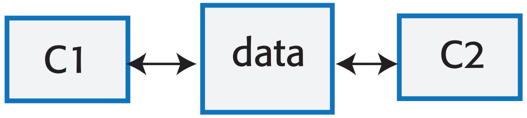
C1 uses C2



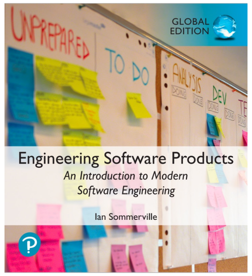
C1 is-located-with C2



C1 shares-data-with C2



Architectural design guidelines

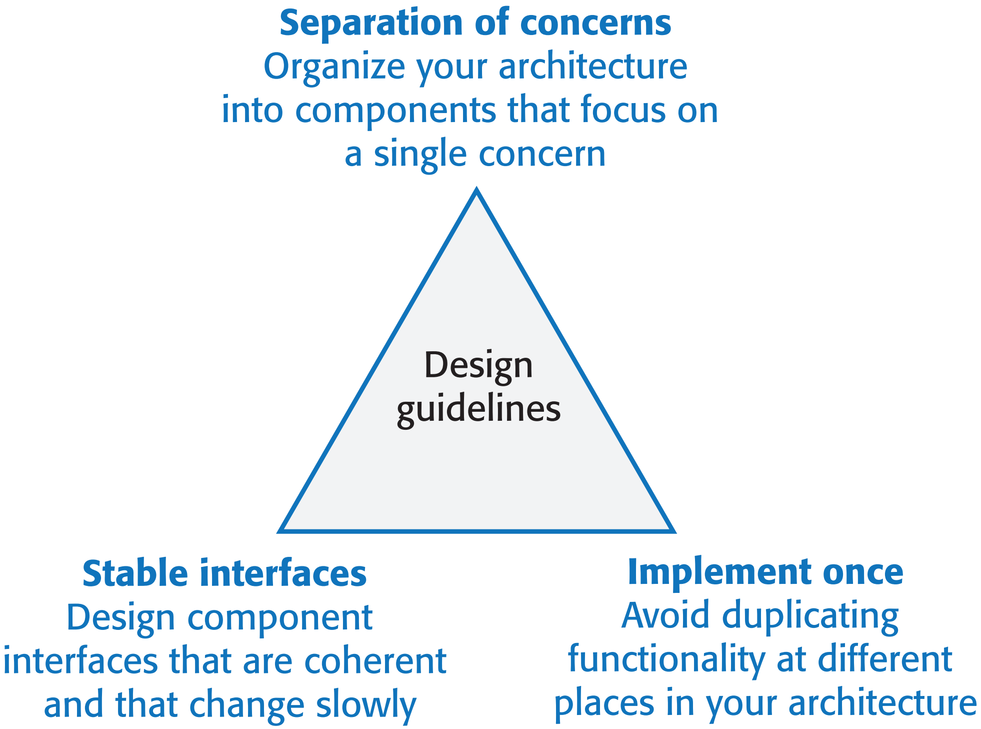


GLOBAL EDITION

Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

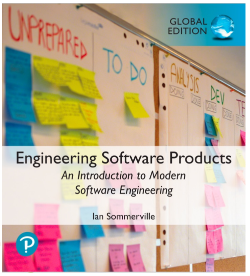


Separation of concerns
Organize your architecture into components that focus on a single concern

Stable interfaces
Design component interfaces that are coherent and that change slowly

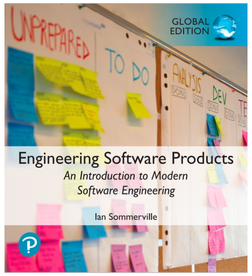
Implement once
Avoid duplicating functionality at different places in your architecture

Design guidelines and layered architectures



- Each layer is an area of concern and is considered separate from other layers.
  - The top layer is concerned with user interaction, the next layer down with user interface management, the third layer with information retrieval and so on.
- Within each layer, the components are independent and do not overlap in functionality.
  - The lower layers include components that provide general functionality so there is no need to replicate this in the components in a higher level.
- The architectural model is a high-level model that does not include implementation information.
  - Ideally, components at level X (say) should only interact with the APIs of the components in level X-1. That is, interactions should be between layers and not across layers.

Cross-cutting concerns



GLOBAL EDITION

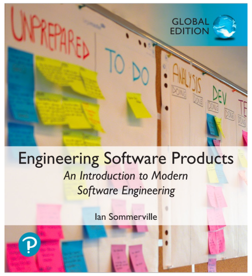
Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

- Cross-cutting concerns are concerns that are systemic, tha is, they affect the whole system.
- In a layered architecture, cross-cutting concerns affect all layers in the system as well as the way in which people use the system.
- Cross-cutting concerns are completely different from the functional concerns represented by layers in a software architecture.
- Every layer has to take them into account and there are inevitably interactions between the layers because of these concerns.
- The existence of cross-cutting concerns is the reason why modifying a system after it has been designed to improve its

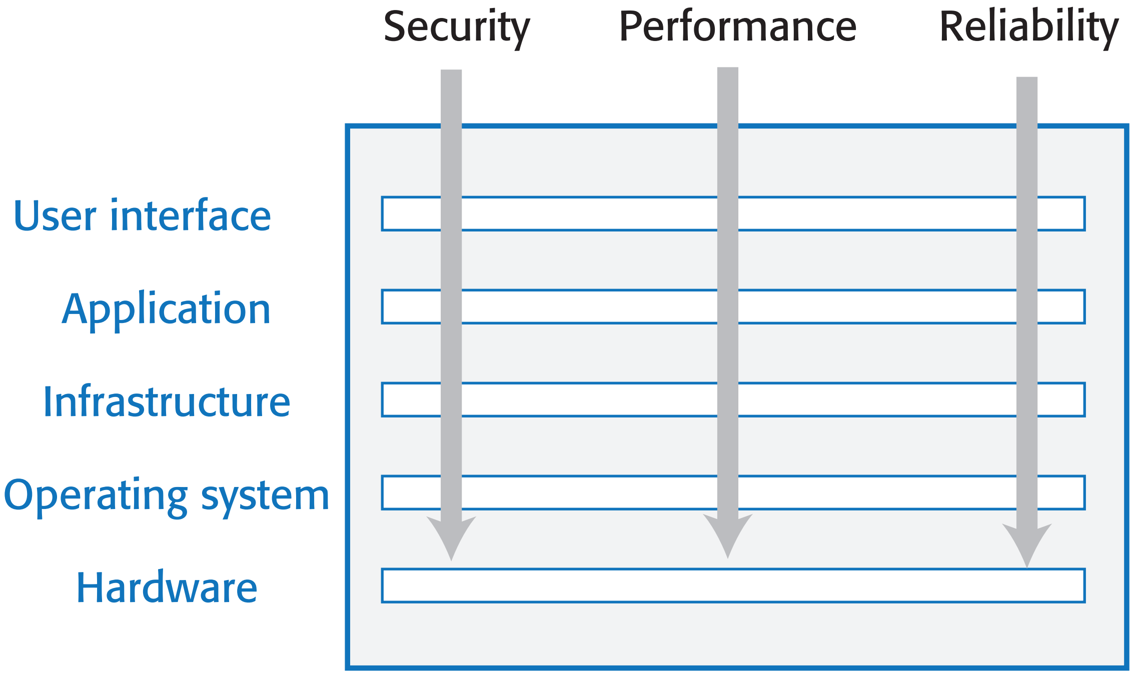
Cross-cutting concerns



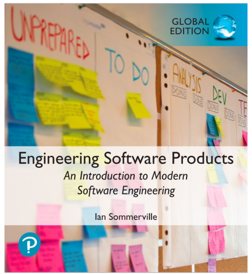
Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

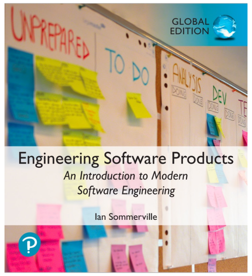


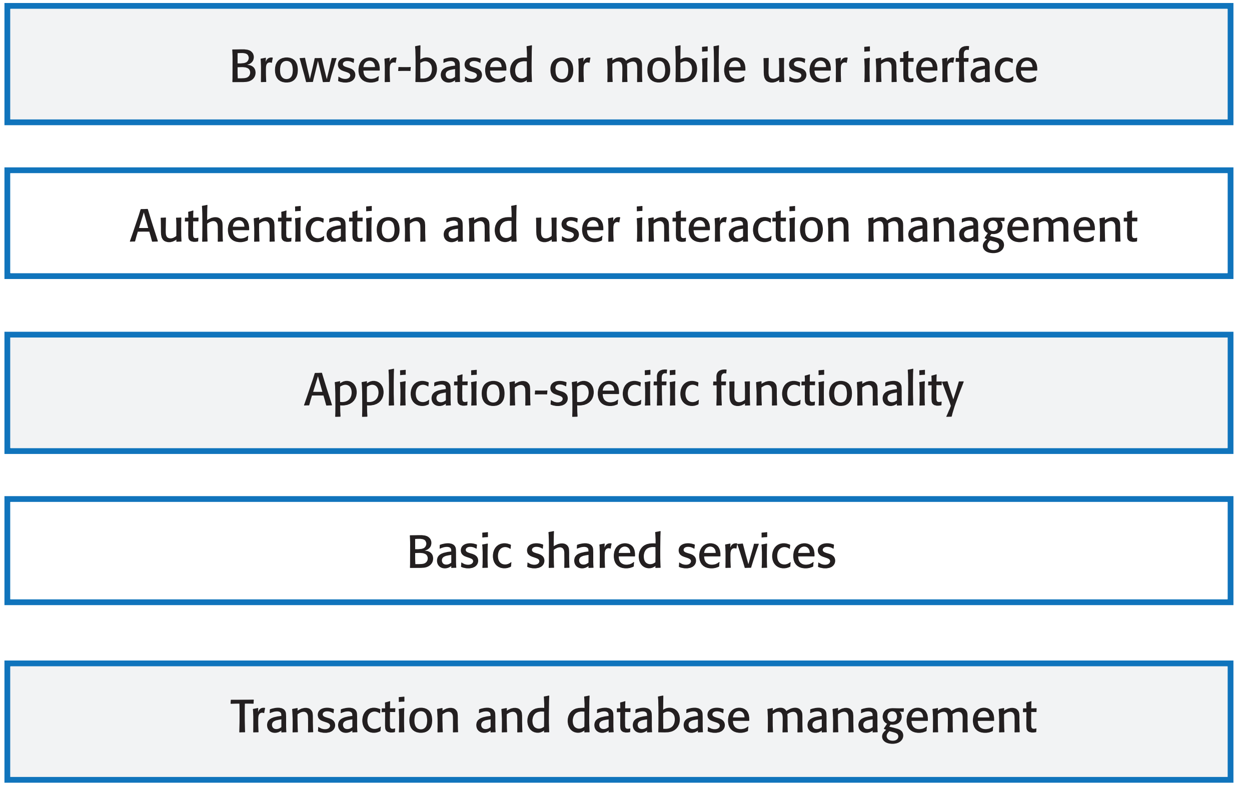
Security as a cross-cutting concern


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- *Security architecture*
  Different technologies are used in different layers, such as an SQL database or a Firefox browser. Attackers can try to use of vulnerabilities in these technologies to gain access.
- Consequently, you need protection from attacks at each layer as well as protection, at lower layers in the system, from successful attacks that have occurred at higher-level layers.
- If there is only a single security component in a system, this represents a critical system vulnerability. If all security checking goes through that component and it stops working properly or is compromised in an attack, then you have no reliable security in your system.
- By distributing security across the layers, your system is more resilient to attacks and software failure (remember the Rogue One example earlier).

A generic layered architecture for a web-based application


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville



Browser-based or mobile user interface

Authentication and user interaction management

Application-specific functionality

Basic shared services

Transaction and database management

Layer functionality in a web-based application

- *Browser-based or mobile user interface*
  A web browser system interface in which HTML forms are often used to collect user input. Javascript components for local actions, such as input validation, should also be included at this level. Alternatively, a mobile interface may be implemented as an app.

- *Authentication and UI management*
  A user interface management layer that may include components for user authentication and web page generation.

- *Application-specific functionality*
  An ‘application’ layer that provides functionality of the application. Sometimes, this may be expanded into more than one layer.

- *Basic shared services*
  A shared services layer, which includes components that provide services used by the application layer components.

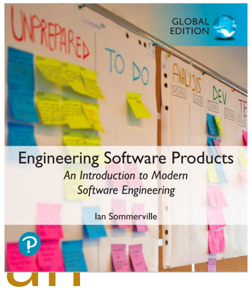
- *Database and transaction management*
  A database layer that provides services such as transaction management and recovery. If your application does not use a database then this may not be required.

iLearn architectural design principles

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

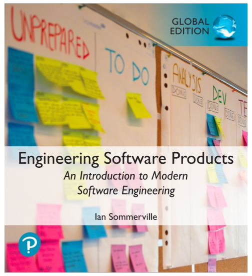
- *Replaceability*
  It should be possible for users to replace applications in the system with alternatives and to add new applications. Consequently, the list of applications included should not be hard-wired into the system.
- *Extensibility*
  It should be possible for users or system administrators to create their own versions of the system, which may extend or limit the 'standard' system.
- *Age-appropriate*
  Alternative user interfaces should be supported so that age-appropriate interfaces for students at different levels can be created.
- *Programmability*
  It should be easy for users to create their own applications by linking existing applications in the system.
- *Minimum work*
  Users who do not wish to change the system should not have to do extra work so that other users can make changes.

iLearn design principles


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

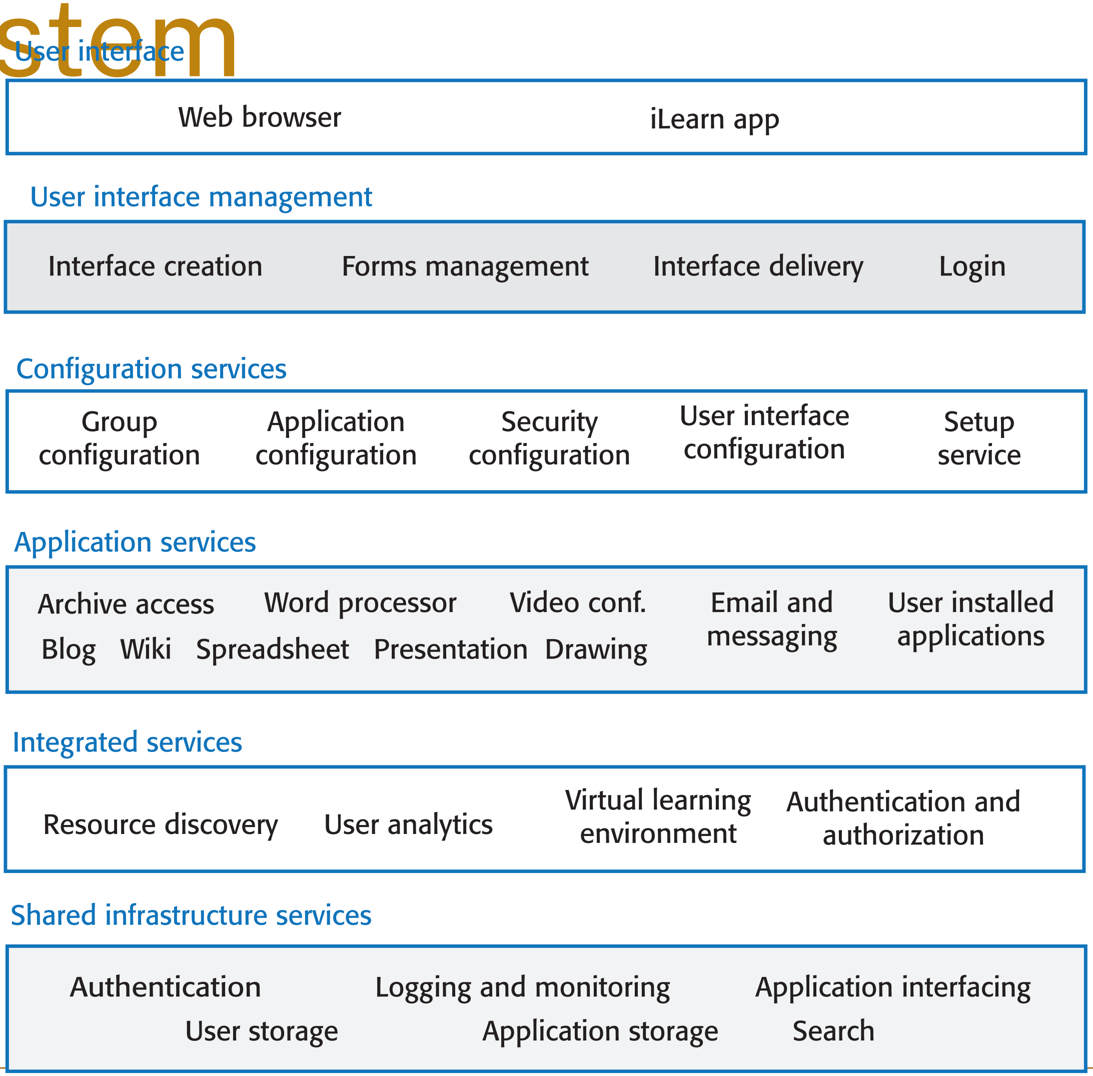
- Our goal in designing the iLearn system was to create an adaptable, universal system that could be easily updated as new learning tools became available.
  - This means that it must be possible to change and replace components and services in the system (principles (1) and (2)).
  - Because the potential system users spanned an age range from 3 to 18, we needed to provide age-appropriate user interfaces and to make it easy to choose an interface (principle (3)).
  - Principle (4) also contributes to system adaptability and principle (5) was included to ensure that this adaptability did not adversely affect users who did not require it.

Designing iLearn as a service-oriented system



- These principles led us to an architectural design decision that the iLearn system should be service-oriented.
- Every component in the system is a service. Any service is potentially replaceable and new services can be created by combining existing services. Different services delivering comparable functionality can be provided for students of different ages.
- Service integration
  - *Full integration* Services are aware of and can communicate with other services through their APIs.
  - *Partial integration* Services may share service components and databases but are not aware of and cannot communicate directly with other application services.
  - *Independent* These services do not use any shared system services or databases and they are unaware of any other services in the system. They can be replaced by any other comparable service.

A layered architectural model of the iLearn system



User interface

Web browser | iLearn app

User interface management

Interface creation | Forms management | Interface delivery | Login

Configuration services

Group configuration | Application configuration | Security configuration | User interface configuration | Setup service

Application services

Archive access | Word processor | Video conf. | Email and messaging | User installed applications

Blog | Wiki | Spreadsheet | Presentation | Drawing

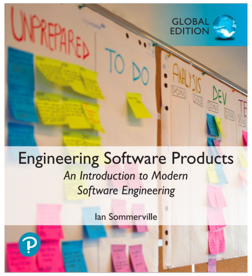
Integrated services

Resource discovery | User analytics | Virtual learning environment | Authentication and authorization

Shared infrastructure services

Authentication | Logging and monitoring | Application interfacing

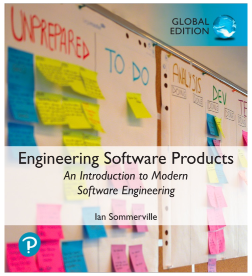
User storage | Application storage | Search

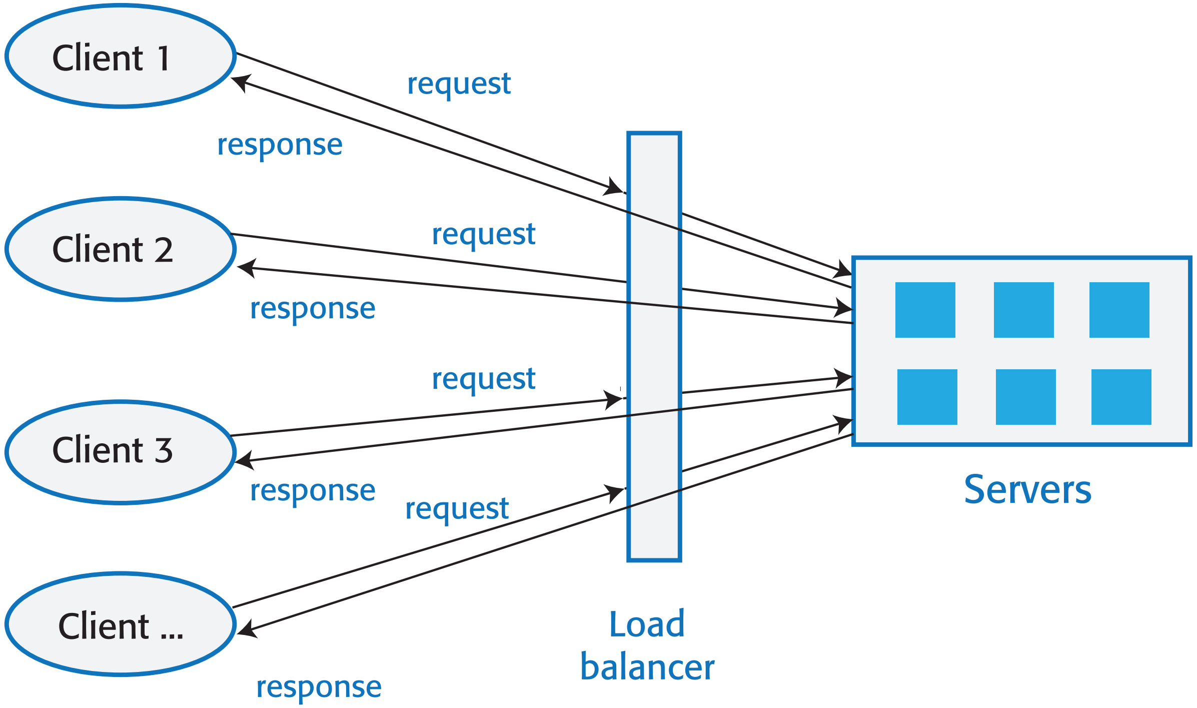

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Distribution architecture

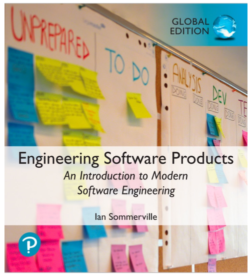
- The distribution architecture of a software system defines the servers in the system and the allocation of components to these servers.
- Client-server architectures are a type of distribution architecture that is suited to applications where clients access a shared database and business logic operations on that data.
- In this architecture, the user interface is implemented on the user's own computer or mobile device.
  - Functionality is distributed between the client and one or more server computers.

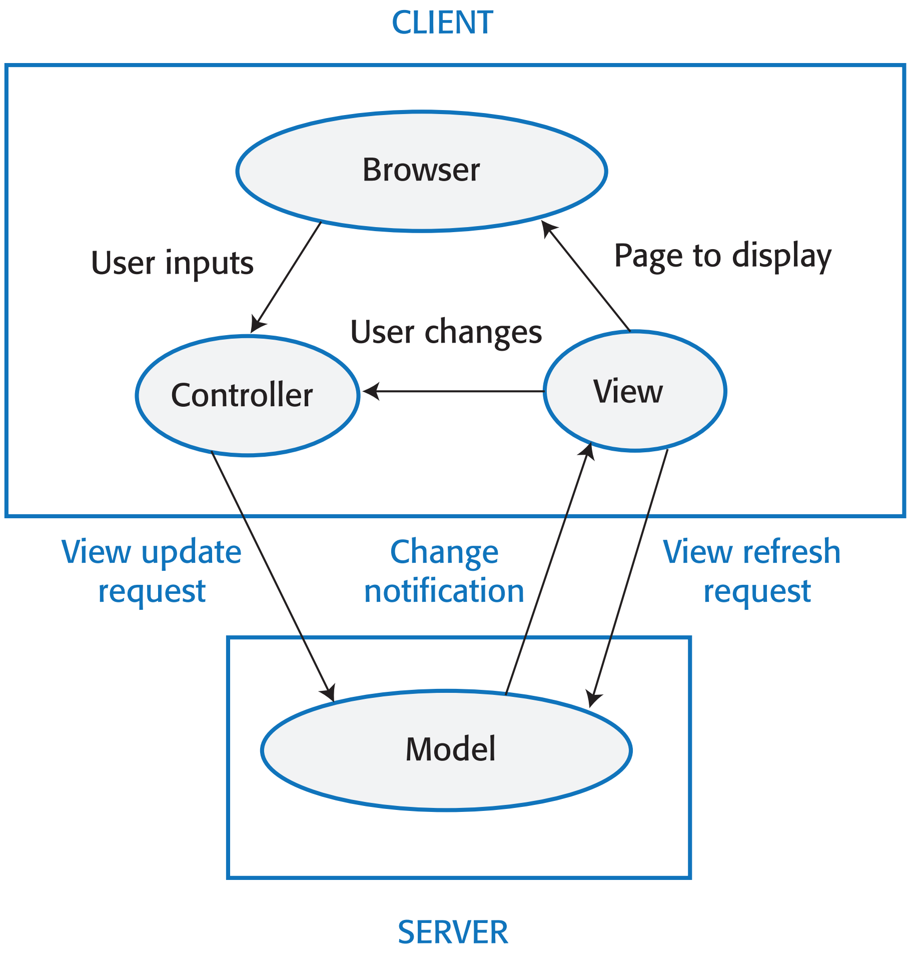
Client-server architecture



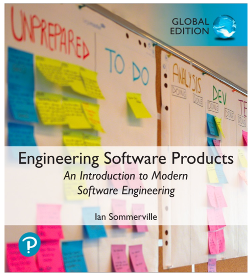


The model-view-controller pattern



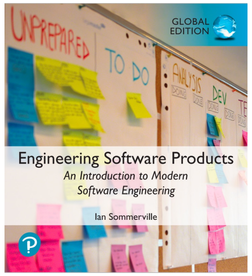


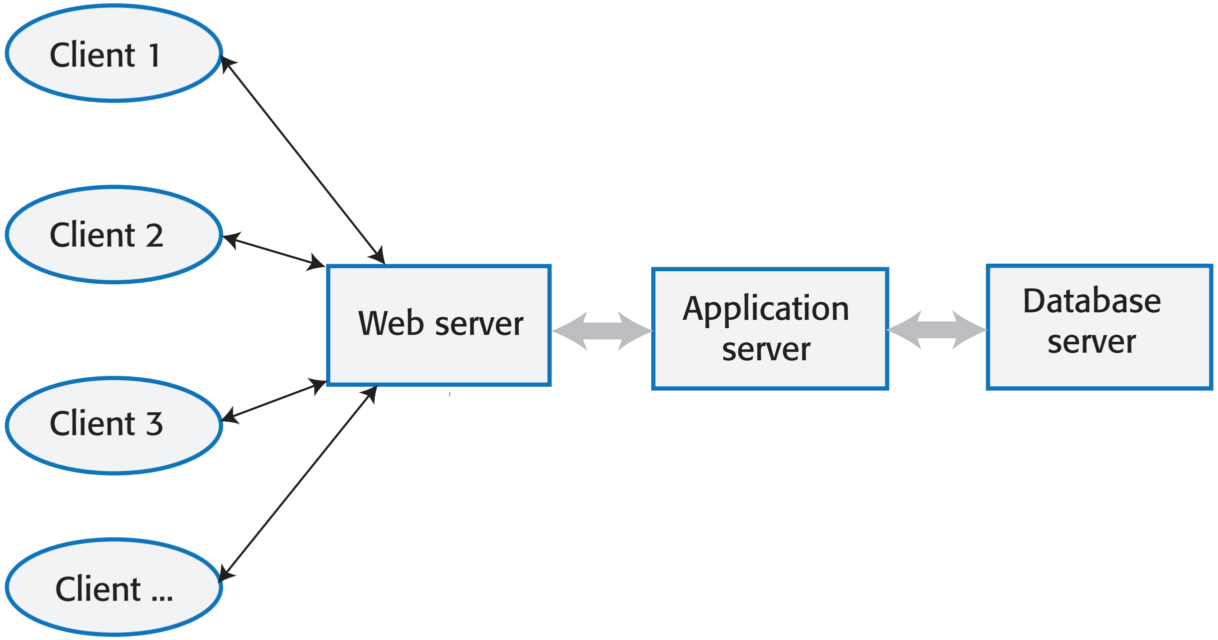
Client-server communication


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Client-server communication normally uses the HTTP protocol.
  - The client sends a message to the server that includes an instruction such as GET or POST along with the identifier of a resource (usually a URL) on which that instruction should operate. The message may also include additional information, such as information collected from a form.
- HTTP is a text-only protocol so structured data has to be represented as text. There are two ways of representing this data that are widely used, namely XML and JSON.
  - XML is a markup language with tags used to identify each data item.
  - JSON is a simpler representation based on the representation of objects in the Javascript language.

Multi-tier client-server architecture


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

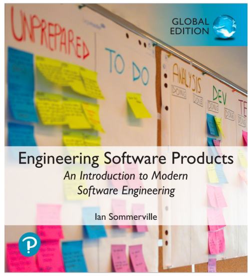


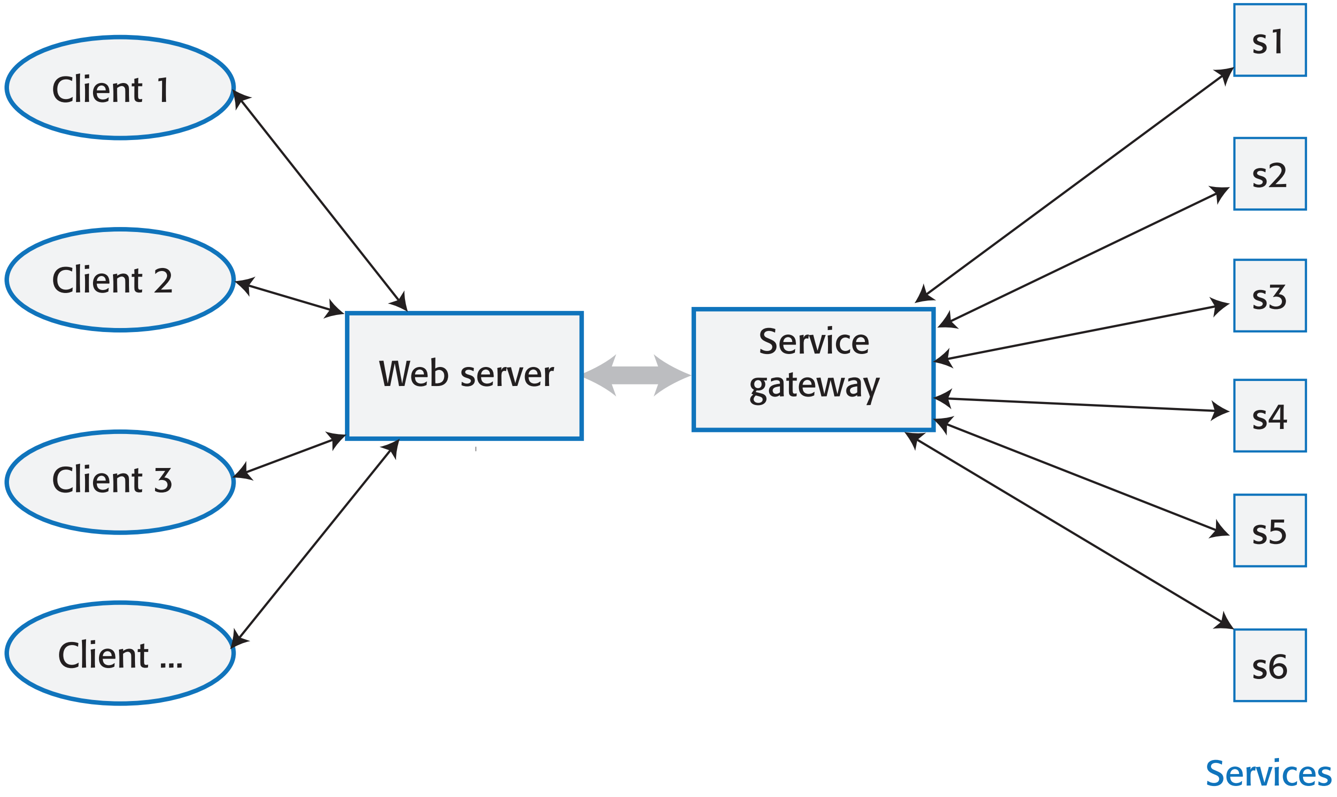
Service-oriented architecture

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

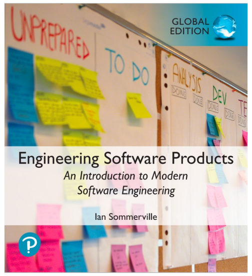
- Services in a service-oriented architecture are stateless components, which means that they can be replicated and can migrate from one computer to another.
- Many servers may be involved in providing services
- A service-oriented architecture is usually easier to scale as demand increases and is resilient to failure.

Service-oriented architecture


Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville



Issues in architectural choice

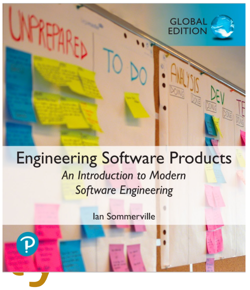

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Data type and data updates
  - If you are mostly using structured data that may be updated by different system features, it is usually best to have a single shared database that provides locking and transaction management. If data is distributed across services, you need a way to keep it consistent and this adds overhead to your system.
- Change frequency
  - If you anticipate that system components will be regularly changed or replaced, then isolating these components as separate services simplifies those changes.
- The system execution platform
  - If you plan to run your system on the cloud with users accessing it over the Internet, it is usually best to implement it as a service-oriented architecture because scaling the system is simpler.
  - If your product is a business system that runs on local servers, a multi-tier architecture may be more appropriate.

Technology choices

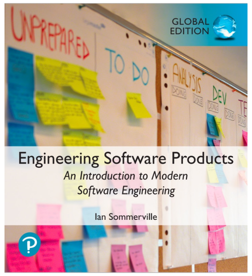
- *Database*
  Should you use a relational SQL database or an unstructured NOSQL database?
- *Platform*
  Should you deliver your product on a mobile app and/or a web platform?
- *Server*
  Should you use dedicated in-house servers or design your system to run on a public cloud? If a public cloud, should you use Amazon, Google, Microsoft, or some other option?
- *Open source*
  Are there suitable open-source components that you could incorporate into your products?
- *Development tools*
  Do your development tools embed architectural assumptions about the software being developed that limit your architectural choices?

Database


UNPREPARED
TO DO
ANALYSIS
DEV
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
P

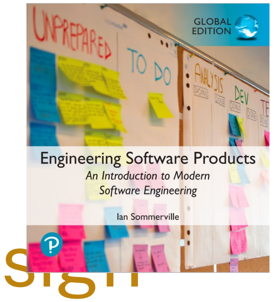
- There are two kinds of database that are now commonly used:
  - Relational databases, where the data is organised into structured tables
  - NoSQL databases, in which the data has a more flexible, user-defined organization.
- Relational databases, such as MySQL, are particularly suitable for situations where you need transaction management and the data structures are predictable and fairly simple.
- NoSQL databases, such as MongoDB, are more flexible and potentially more efficient than relational databases for data analysis.
  - NoSQL databases allow data to be organized hierarchically rather than as

Delivery platform


Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Delivery can be as a web-based or a mobile product or both
- Mobile issues:
  - *Intermittent connectivity* You must be able to provide a limited service without network connectivity.
  - *Processor power* Mobile devices have less powerful processors, so you need to minimize computationally-intensive operations.
  - *Power management* Mobile battery life is limited so you should try to minimize the power used by your application.
  - *On-screen keyboard* On-screen keyboards are slow and error-prone. You should minimize input using the screen keyboard to reduce user frustration.
- To deal with these differences, you usually need separate browser-based and mobile versions of your product front-end.
  - You may need a completely different decomposition architecture in these different versions to ensure that performance and other characteristics are maintained.

Server



GLOBAL EDITION

Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

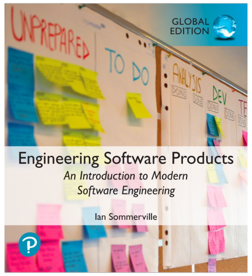
- A key decision that you have to make is whether to design your system to run on customer servers or to run on the cloud.
- For consumer products that are not simply mobile apps I think it almost always makes sense to develop for the cloud.
- For business products, it is a more difficult decision.
  - Some businesses are concerned about cloud security and prefer to run their systems on in-house servers. They may have a predictable pattern of system usage so there is less need to design your system to cope with large changes in demand.
- An important choice you have to make if you are running

Open source

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

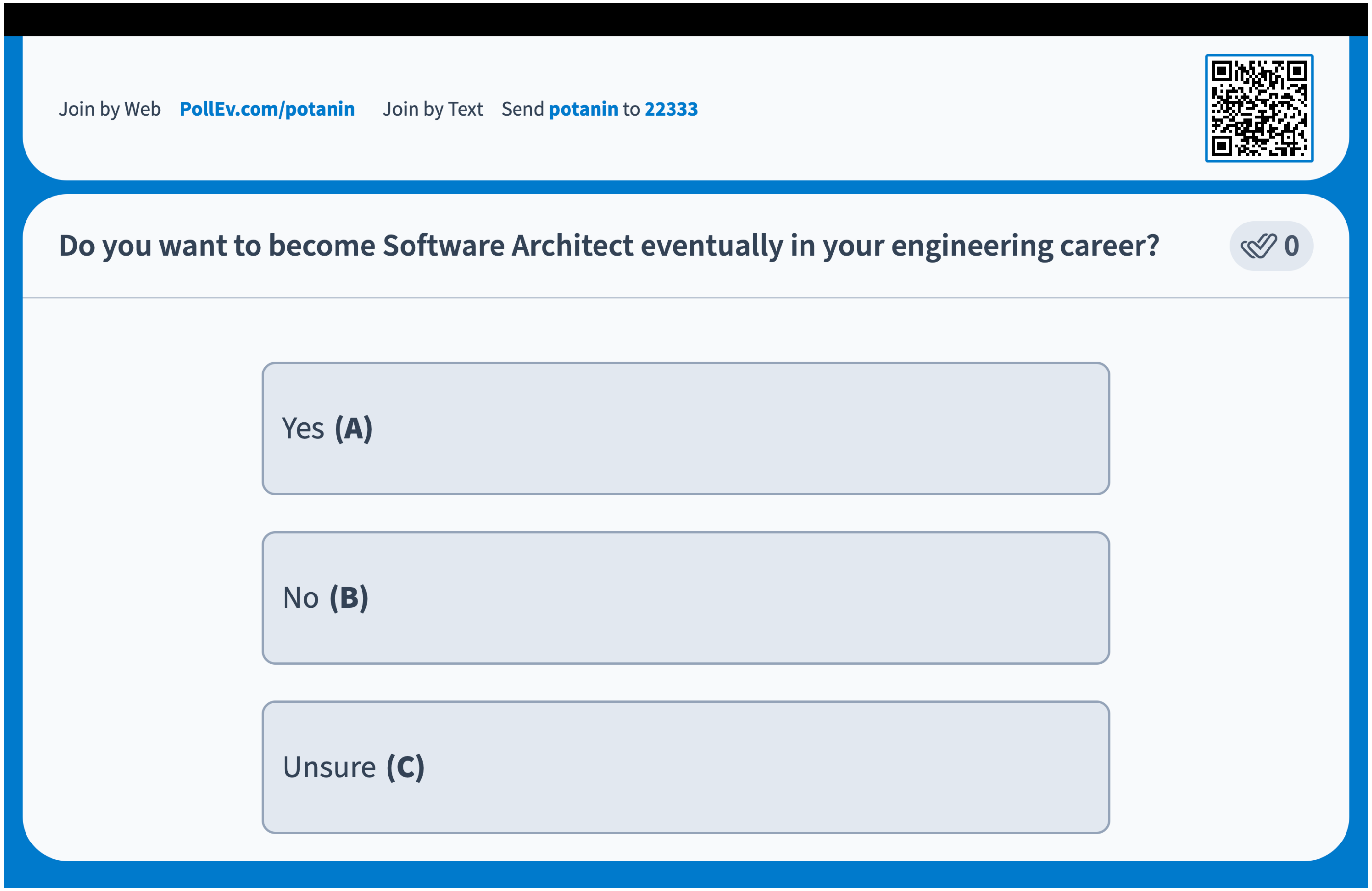
- Open source software is software that is available freely, which yo can change and modify as you wish.
  - The advantage is that you can reuse rather than implement new software, which reduces development costs and time to market.
  - The disadvantages of using open-source software is that you are constrained by that software and have no control over its evolution.
- The decision on the use of open-source software also depends on the availability, maturity and continuing support of open source components.
- Open source license issues may impose constraints on how you use the software.
- Your choice of open source software should depend on the type of product that you are developing, your target market and the expertise of your development team.

Development tools



- Development technologies, such as a mobile development toolkit or a web application framework, influence the architecture of your software.
  - These technologies have built-in assumptions about system architectures and you have to conform to these assumptions to use the development system.
- The development technology that you use may also have an indirect influence on the system architecture.
  - Developers usually favour architectural choices that use familiar technologies that they understand. For example, if your team have a lot of experience of relational databases, they may argue for this instead of a NoSQL database.

Poll Everywhere Time!



Join by Web **PollEv.com/potanin** Join by Text Send **potanin** to **22333**

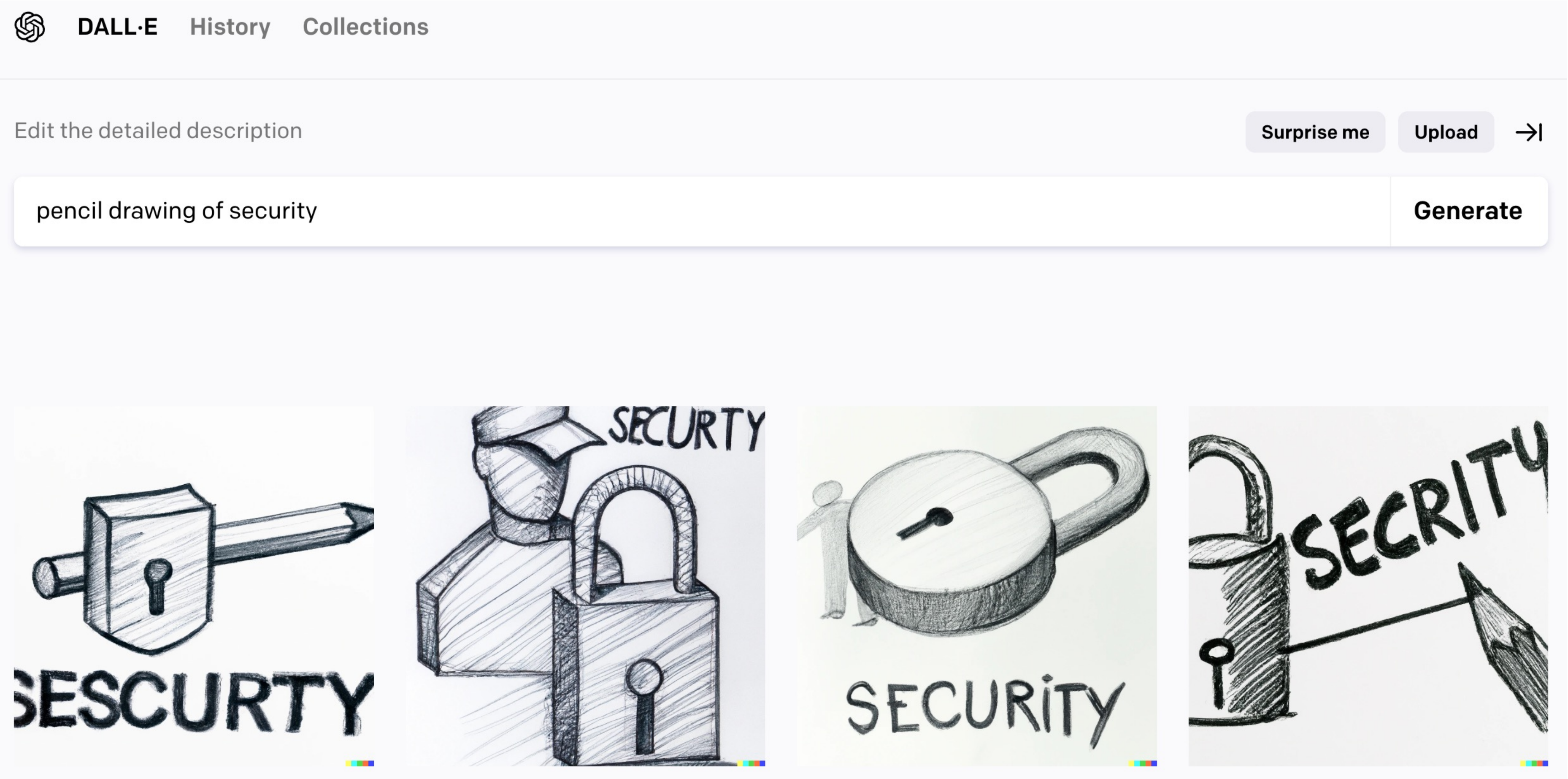
Do you want to become Software Architect eventually in your engineering career?

✓ 0

Yes **(A)**

No **(B)**

Unsure **(C)**

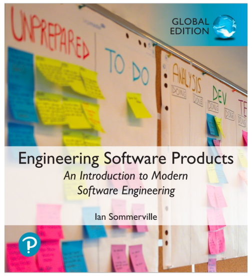


Edit the detailed description

pencil drawing of security

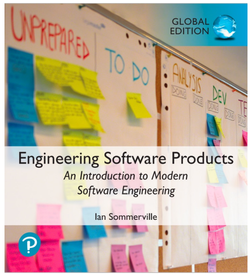
Security

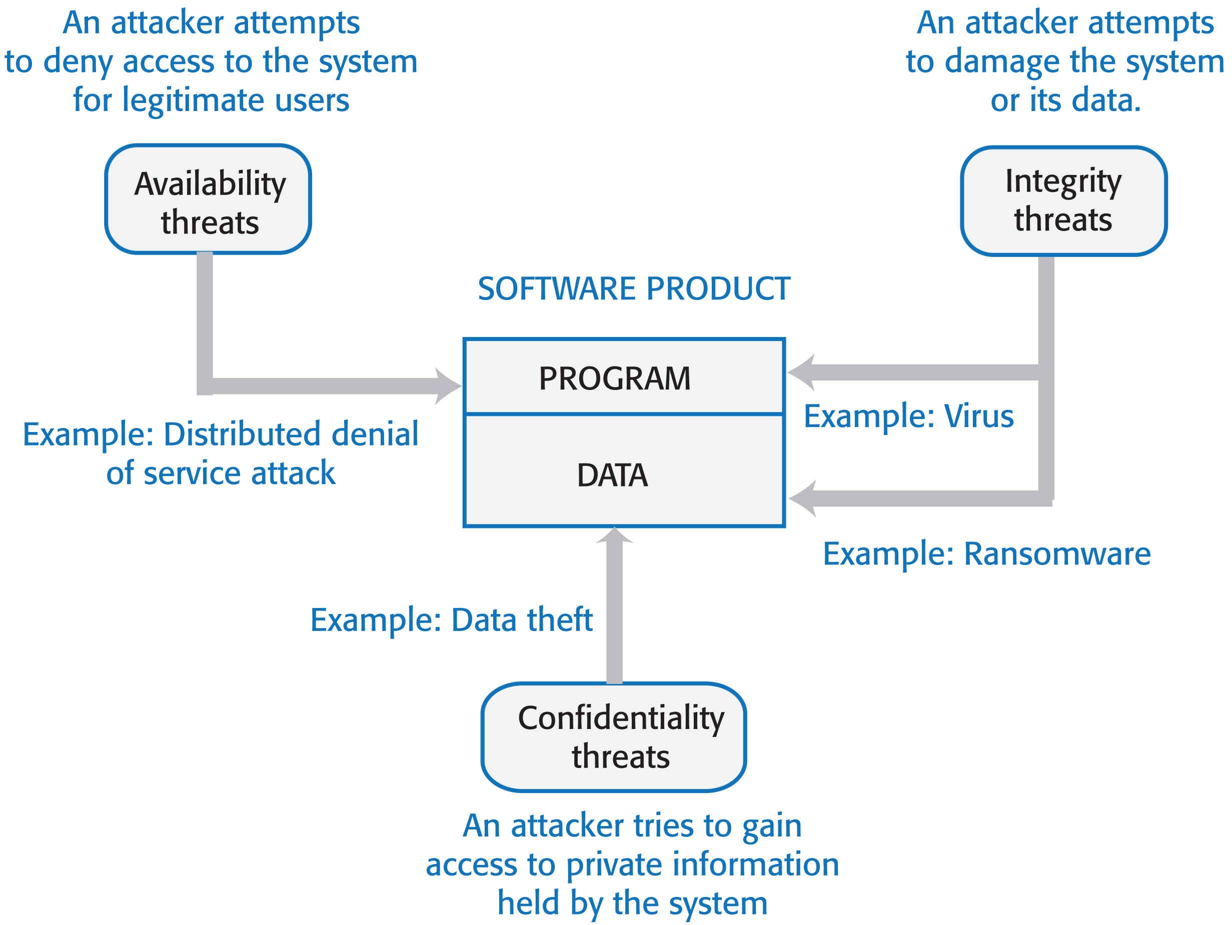
Software security


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Software security should always be a high priority for product developers and their users.
- If you don’t prioritize security, you and your customers will inevitably suffer losses from malicious attacks.
- In the worst case, these attacks could can put product providers out of business.
  - If their product is unavailable or if customer data is compromised, customers are liable to cancel their subscriptions.
- Even if they can recover from the attacks, this will take time and effort that would have been better spent working on their software.

Types of security threat


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville



An attacker attempts to deny access to the system for legitimate users

Availability threats

Example: Distributed denial of service attack

An attacker attempts to damage the system or its data.

Integrity threats

Example: Virus

Example: Ransomware

SOFTWARE PRODUCT

PROGRAM

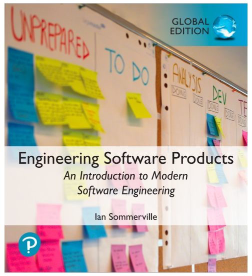
DATA

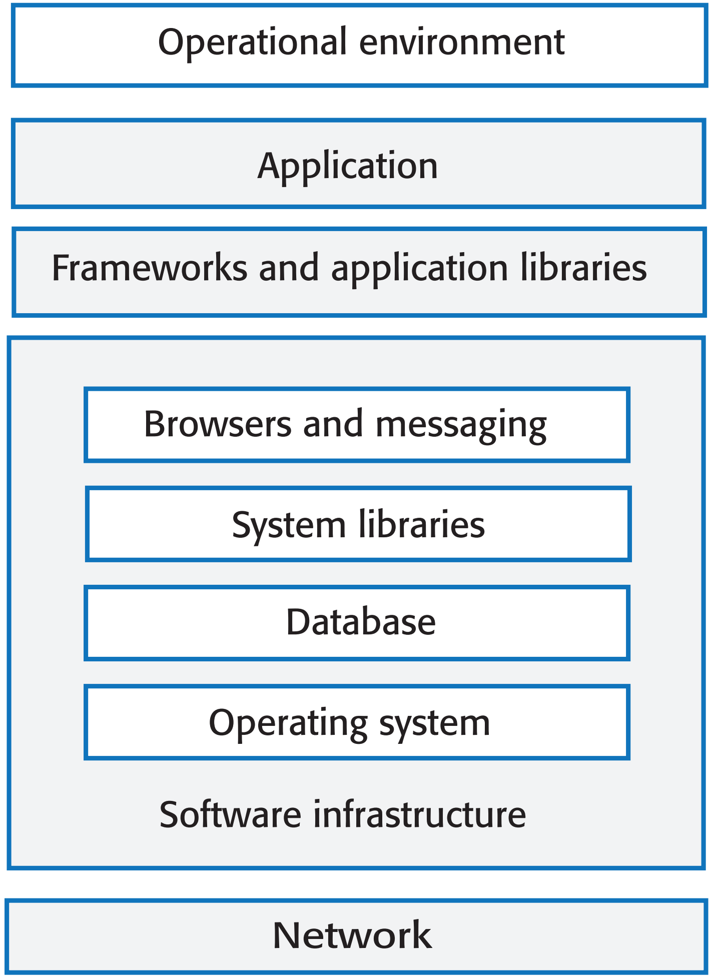
Example: Data theft

Confidentiality threats

An attacker tries to gain access to private information held by the system

System infrastructure stack





Security management

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

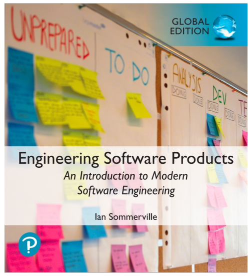
- ***Authentication and authorization***
  You should have authentication and authorization standards and procedures that ensure that all users have strong authentication and that they have properly access permissions properly. This minimizes the risk of unauthorized users accessing system resources.

- ***System infrastructure management***
  Infrastructure software should be properly configured and security updates that patch vulnerabilities should be applied as soon as they become available.

- ***Attack monitoring***
  The system should be regularly checked for possible unauthorized access. If attacks are detected, it may be possible to put resistance strategies in place that minimize the effects of the attack.

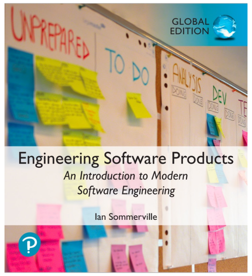
- ***Backup***
  Backup policies should be implemented to ensure that you keep undamaged copies of program and data files. These can then be restored after an attack.

Operational security


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

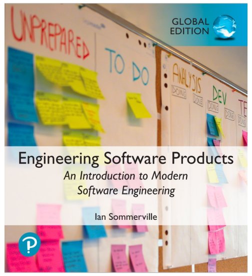
- Operational security focuses on helping users to maintain security. User attacks try to trick users into disclosing their credentials or accessing a website that includes malware such as a key-logging system.
- Operational security procedures and practices
  - *Auto-logout*, which addresses the common problem of users forgetting to logout from a computer used in a shared space.
  - *User command logging*, which makes it possible to discover actions taken by users that have deliberately or accidentally damaged some system resources.
  - *Multi-factor authentication*, which reduces the chances of an intruder gaining access to the system using stolen credentials.

Injection attacks



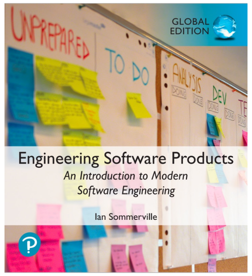
- Injection attacks are a type of attack where a malicious user uses a valid input field to input malicious code or database commands.
- These malicious instructions are then executed, causing some damage to the system. Code can be injected that leaks system data to the attackers.
- Common types of injection attack include buffer overflow attacks and SQL poisoning attacks.

SQL poisoning attacks


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

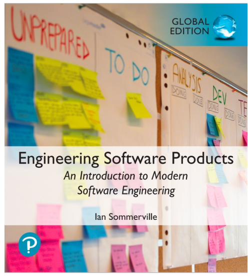
- SQL poisoning attacks are attacks on software products that use an SQL database.
- They take advantage of a situation where a user input is used as part of an SQL command.
- A malicious user uses a form input field to input a fragment of SQL that allows access to the database.
- The form field is added to the SQL query, which is executed and returns the information to the attacker.

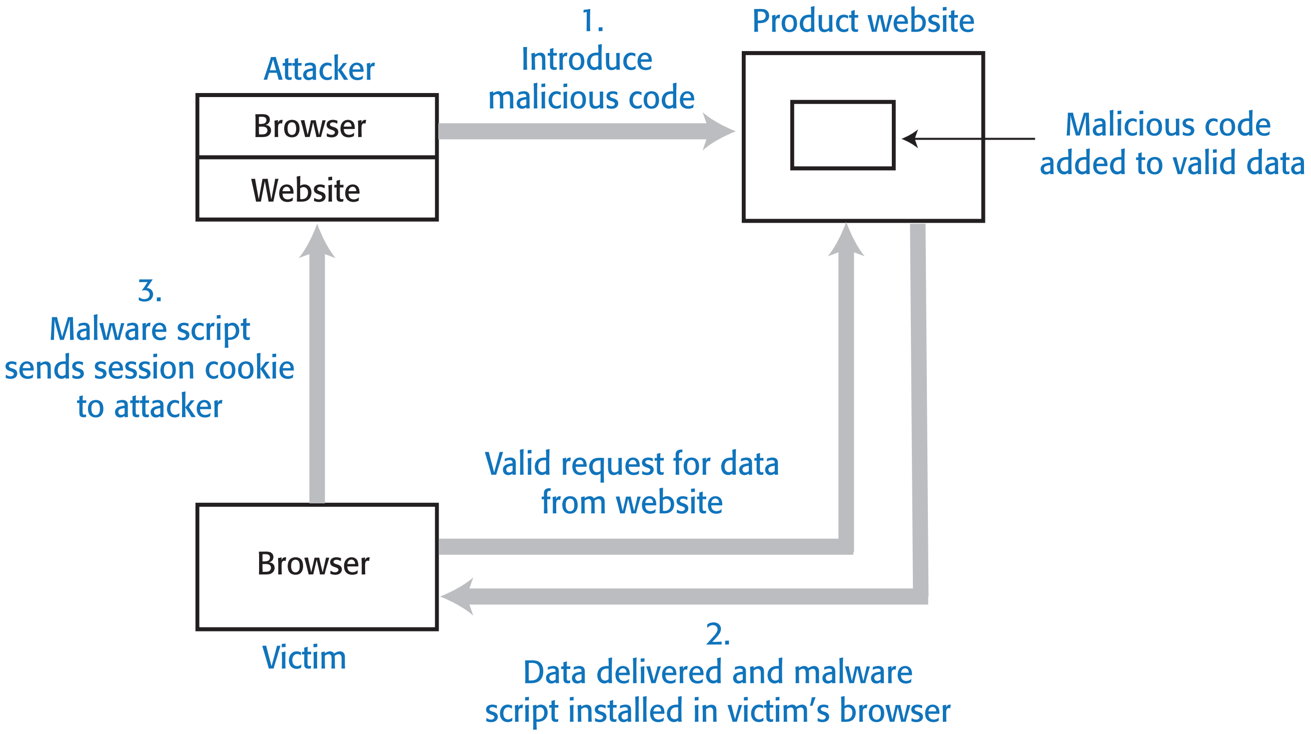
Cross-site scripting attacks


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Cross-site scripting attacks are another form of injection attack.
- An attacker adds malicious Javascript code to the web page that is returned from a server to a client and this script is executed when the page is displayed in the user's browser.
- The malicious script may steal customer information or direct them to another website.
  - This may try to capture personal data or display advertisements.
  - Cookies may be stolen, which makes a session hijacking attack possible.
- As with other types of injection attack, cross-site scripting attacks may be avoided by input validation.

Cross-site scripting attack



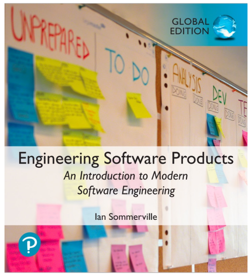

Attacker
Browser
Website
1. Introduce malicious code
Product website
Malicious code added to valid data
3. Malware script sends session cookie to attacker
Valid request for data from website
Browser
Victim
2. Data delivered and malware script installed in victim's browser

Session hijacking attacks

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

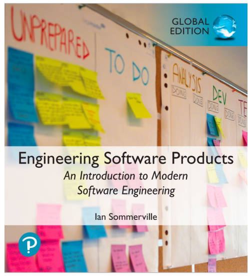
- When a user authenticates themselves with a web application, a session is cre
  - A session is a time period during which the user's authentication is valid. They don't have to re-authenticate for each interaction with the system.
  - The authentication process involves placing a session cookie on the user's device
- Session hijacking is a type of attack where an attacker gets hold of a session cookie and uses this to impersonate a legitimate user.
- There are several ways that an attacker can find out the session cookie value including cross-site scripting attacks and traffic monitoring.
  - In a cross-site scripting attack, the installed malware sends session cookies to the attackers.
  - Traffic monitoring involves attackers capturing the traffic between the client and server. The session cookie can then be identified by analysing the data exchanged.

Actions to reduce the likelihood of hacking


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

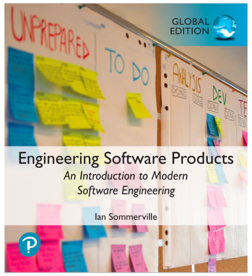
- ***Traffic encryption***
  Always encrypt the network traffic between clients and your server. This means setting up sessions using https rather than http. If traffic is encrypted it is harder to monitor to find session cookies.
- ***Multi-factor authentication***
  Always use multi-factor authentication and require confirmation of new actions that may be damaging. For example, before a new payee request is accepted, you could ask the user to confirm their identity by inputting a code sent to their phone. You could also ask for password characters to be input before every potentially damaging action, such as transferring funds.
- ***Short timeouts***
  Use relatively short timeouts on sessions. If there has been no activity in a session for a few minutes, the session should be ended and future requests directed to an authentication page. This reduces the likelihood that an attacker can access an account if a legitimate user forgets to log off when they have finished their transactions.

Denial of service attacks



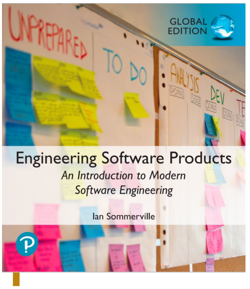
- Denial of service attacks are attacks on a software system that are intended to make that system unavailable for normal use.
- Distributed denial of service attacks (DDOS) are the most common type of denial of service attacks.
  - These involve distributed computers, that have usually been hijacked as part of a botnet, sending hundreds of thousands of requests for service to a web application. There are so many service requests that legitimate users are denied access.
- Other types of denial of service attacks target application users.
  - User lockout attacks take advantage of a common authentication policy that locks out a user after a number of failed authentication attempts. Their aim is to lock users out rather than gain access and so deny the service to these users.
  - Users often use their email address as their login name so if an attacker has access to a database of email addresses, he or she can try to login using these addresses.
- If you don’t lock accounts after failed validation, then attackers can use brute-force attacks on your system. If you do, you may deny access to legitimate users.

Brute force attacks



- Brute force attacks are attacks on a web application where the attacker has some information, such as a valid login name, but does not have the password for the site.
- The attacker creates different passwords and tries to login with each of these. If the login fails, they then try again with a different password.
  - Attackers may use a string generator that generates every possible combination of letters and numbers and use these as passwords.
  - To speed up the process of password discovery, attackers take advantage of the fact that many users choose easy-to-remember passwords. They start by trying passwords from the published lists of the most common passwords.
- Brute force attacks rely on users setting weak passwords, so you can circumvent them by insisting that users set long passwords that are not in a dictionary or are common words.

Authentication


GLOBAL EDITION

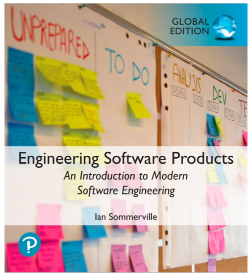
Engineering Software Products

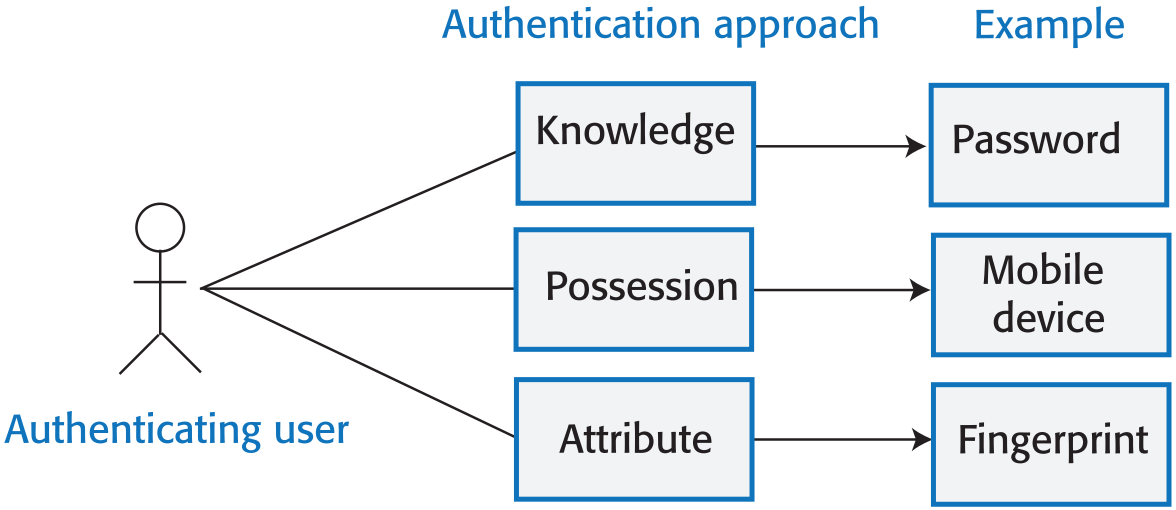
An Introduction to Modern Software Engineering

Ian Sommerville

- Authentication is the process of ensuring that a user of your system is who they claim to be.
- You need authentication in all software products that maintain user information, so that only the providers of that information can access and change it.
- You also use authentication to learn about your users so that you can personalize their experience of using your product.

Authentication approaches


Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



Authentication approach

Example

Knowledge

Password

Possession

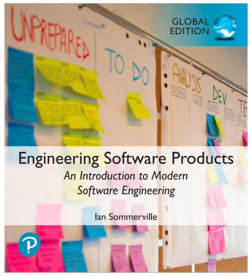
Mobile device

Attribute

Fingerprint

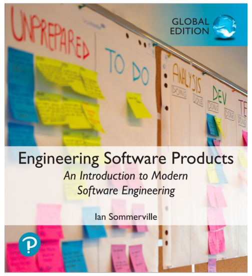
Authenticating user

Authentication methods



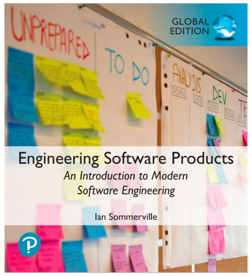
- Knowledge-based authentication
  - The user provides secret, personal information when they register with the system. Each time they log on, the system asks them for this information.
- Possession-based authentication
  - This relies on the user having a physical device (such as a mobile phone) that can generate or display information that is known to the authenticating system. The user inputs this information to confirm that they possess the authenticating device.
- Attribute-based authentication is based on a unique biometric attribute of the user, such as a fingerprint, which is registered with the system.
- Multi-factor authentication combines these approaches and requires users to use more than one authentication method.

Weaknesses of password-password-based authentication



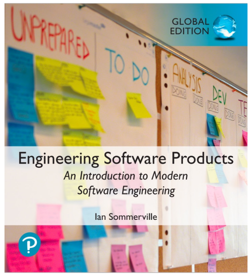
- ***Insecure passwords***
  Users choose passwords that are easy to remember. However, it is also easy for attackers to guess or generate these passwords, using either a dictionary or a brute force attack.
- ***Phishing attacks***
  Users click on an email link that points to a fake site that tries to collect their login and password details.
- ***Password reuse***
  Users use the same password for several sites. If there is a security breach at one of these sites, attackers then have passwords that they can try on other sites.
- ***Forgotten passwords***
  Users regularly forget their passwords so that you need to set up a password recovery mechanism to allow these to be reset. This can be a vulnerability if users' credentials have been stolen and attackers use it to reset their passwords.

Federated identity

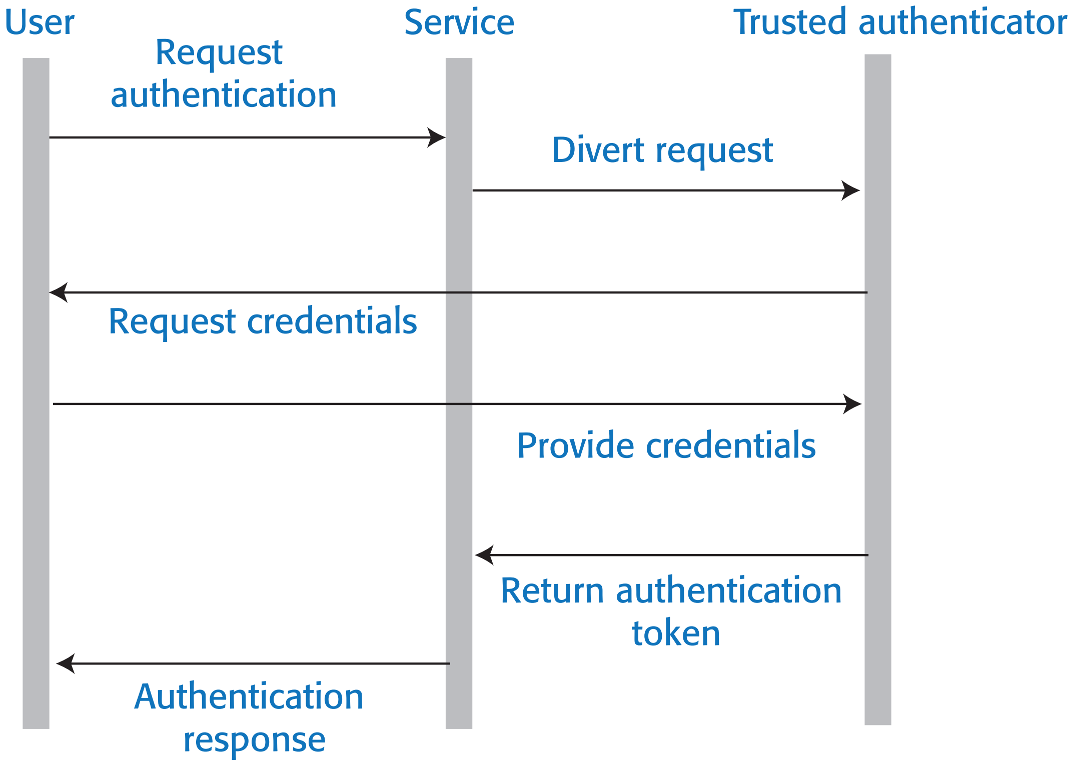

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- Federated identity is an approach to authentication where you use an external authentication service.
- ‘Login with Google’ and ‘Login with Facebook’ are widely used examples of authentication using federated identity.
- The advantage of federated identity for a user is that they have a single set of credentials that are stored by a trusted identity service.
- Instead of logging into a service directly, a user provides their credentials to a known service who confirms their identity to the authenticating service.
- They don’t have to keep track of different user ids and passwords. Because their credentials are stored in fewer places, the chances of a security breach where these are revealed is reduced.

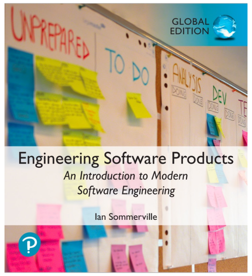
Federated identity



GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

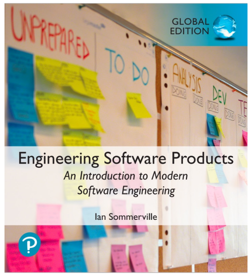


Authorization



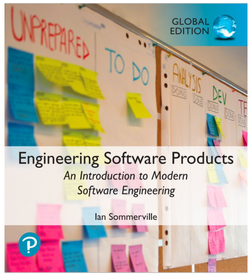
- Authentication involves a user proving their identity to a software system.
- Authorization is a complementary process in which that identity is used to control access to software system resources.
  - For example, if you use a shared folder on Dropbox, the folder's owner may authorize you to read the contents of that folder, but not to add new files or overwrite files in the folder.
- When a business wants to define the type of access that users get to resources, this is based on an access control policy.
- This policy is a set of rules that define what information (data and programs) is controlled, who has access to that information and the type of access that is allowed

Access control policies



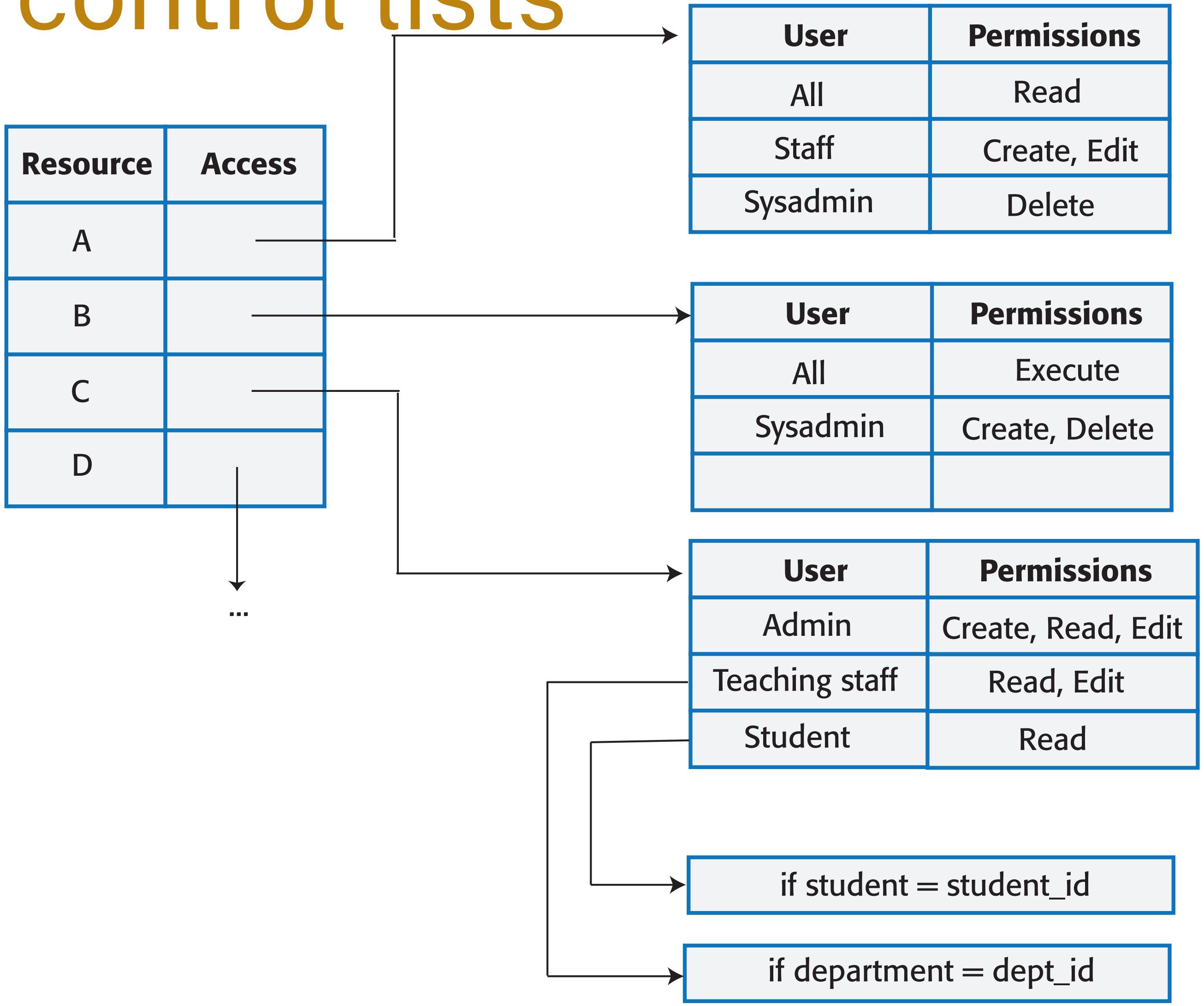
- Explicit access control policies are important for both legal and technical reasons.
  - Data protection rules limit the access the personal data and this must be reflected in the defined access control policy. If this policy is incomplete or does not conform to the data protection rules, then there may be subsequent legal action in the event of a data breach.
  - Technically, an access control policy can be a starting point for setting up the access control scheme for a system.
  - For example, if the access control policy defines the access rights of students, then when new students are registered, they all get these rights by default.

Access control lists



- Access control lists (ACLs) are used in most file and database systems to implement access control policies.
- Access control lists are tables that link users with resources and specify what those users are permitted to do.
  - For example, for this book I would like to be able to set up an access control list to a book file that allows reviewers to read that file and annotate it with comments. However, they are not allowed to edit the text or to delete the file.
- If access control lists are based on individual permissions, then these can become very large. However, you can dramatically cut their size by allocating users to groups and then assigning permissions to the group

Access control lists



Resource	Access
A	
B	
C	
D	

…

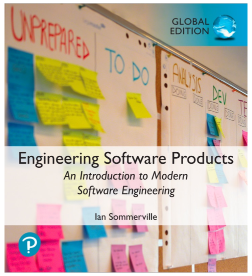
User	Permissions
All	Read
Staff	Create, Edit
Sysadmin	Delete

User	Permissions
All	Execute
Sysadmin	Create, Delete

User	Permissions
Admin	Create, Read, Edit
Teaching staff	Read, Edit
Student	Read

if student = student_id

if department = dept_id


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Encryption

GLOBAL EDITION

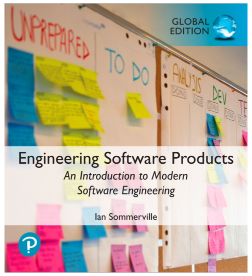
Engineering Software Products

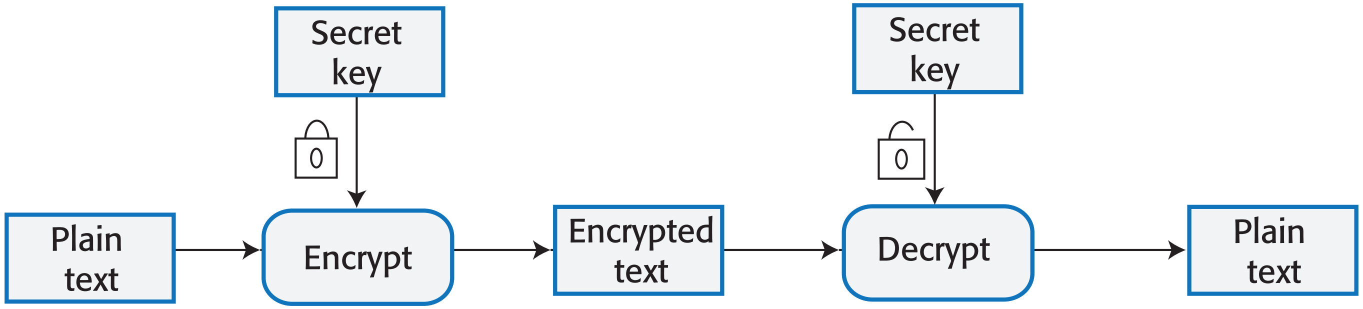
An Introduction to Modern Software Engineering

Ian Sommerville

- Encryption is the process of making a document unreadable by applying an algorithmic transformation to it.
- A secret key is used by the encryption algorithm as the basis of this transformation. You can decode the encrypted text by applying the reverse transformation.
- Modern encryption techniques are such that you can encrypt data so that it is practically uncrackable using currently available technology.
- However, history has demonstrated that apparently strong encryption may be crackable when new technology becomes available.
- If commercial quantum systems become available, we will have to use a completely different approach to encryption on

Encryption and decryption



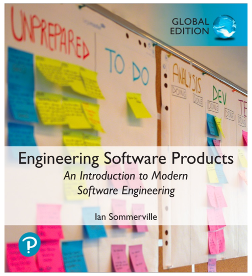


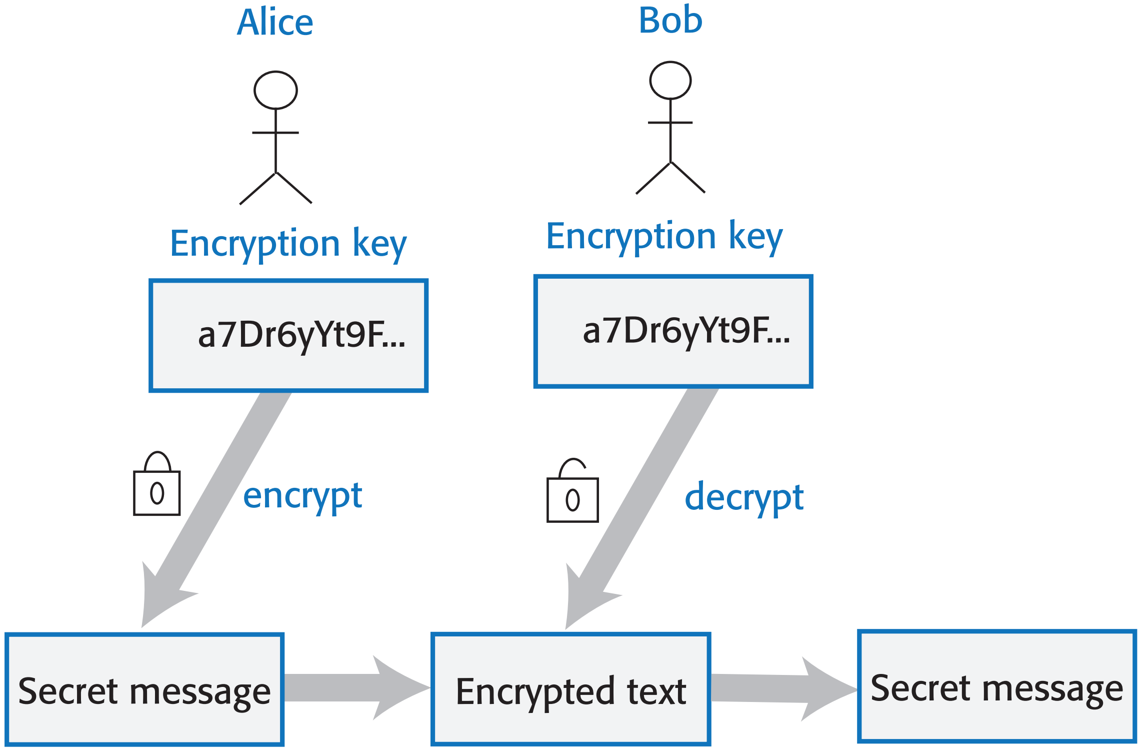
Symmetric encryption

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

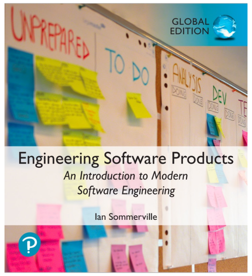
- In a symmetric encryption scheme, the same encryption key is used for encoding and decoding the information that is to be kept secret.
- If Alice and Bob wish to exchange a secret message, both must have a copy of the encryption key. Alice encrypts the message with this key. When Bob receives the message, he decodes it using the same key to read its contents.
- The fundamental problem with a symmetric encryption scheme is securely sharing the encryption key.
- If Alice simply sends the key to Bob, an attacker may intercept the message and gain access to the key. The attacker can then decode all future secret communications.

Symmetric encryption


Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION



Asymmetric encryption



GLOBAL EDITION

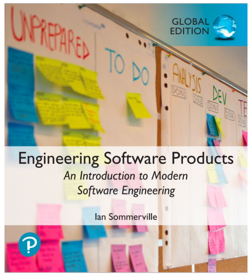
Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

- Asymmetric encryption, does not require secret keys to be shared.
- An asymmetric encryption scheme uses different keys for encrypting and decrypting messages.
- Each user has a public and a private key. Messages may be encrypted using either key but can only be decrypted using the other key.
- Public keys may be published and shared by the key owner. Anyone can access and use a published public key.
- However, messages can only be decrypted by the user's private key so is only readable by the intended recipient

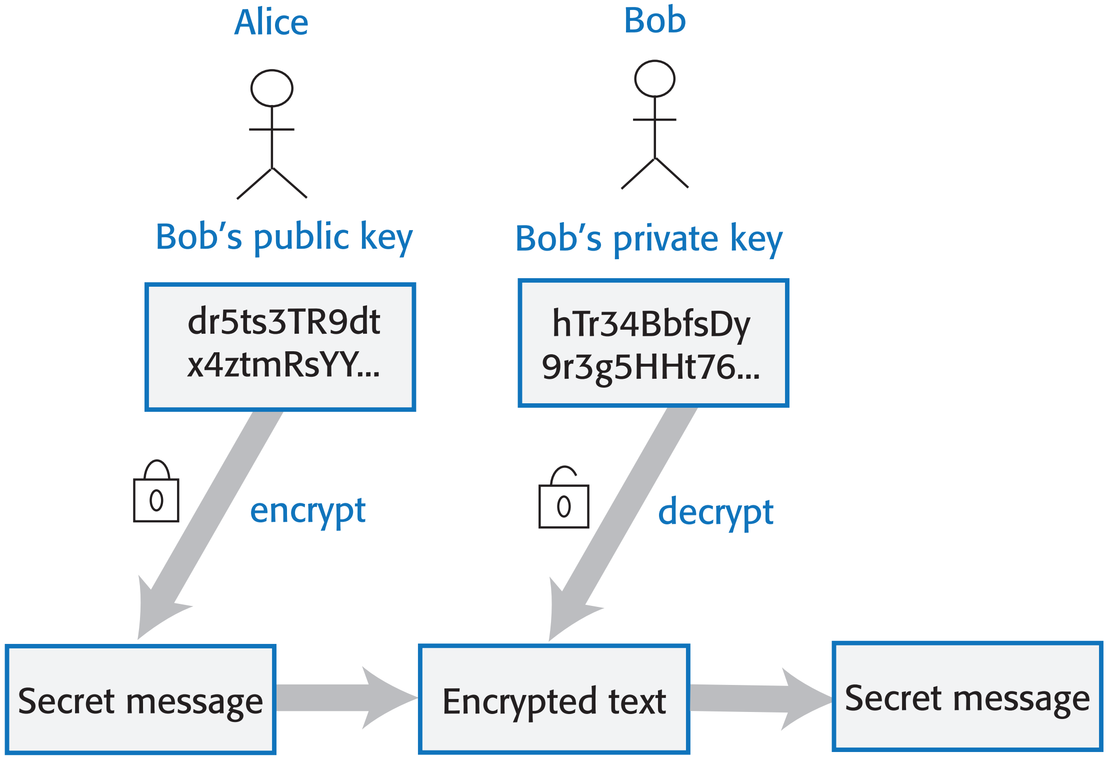
Asymmetric encryption



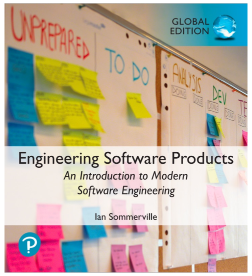
Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

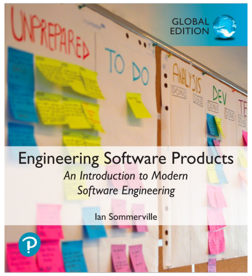


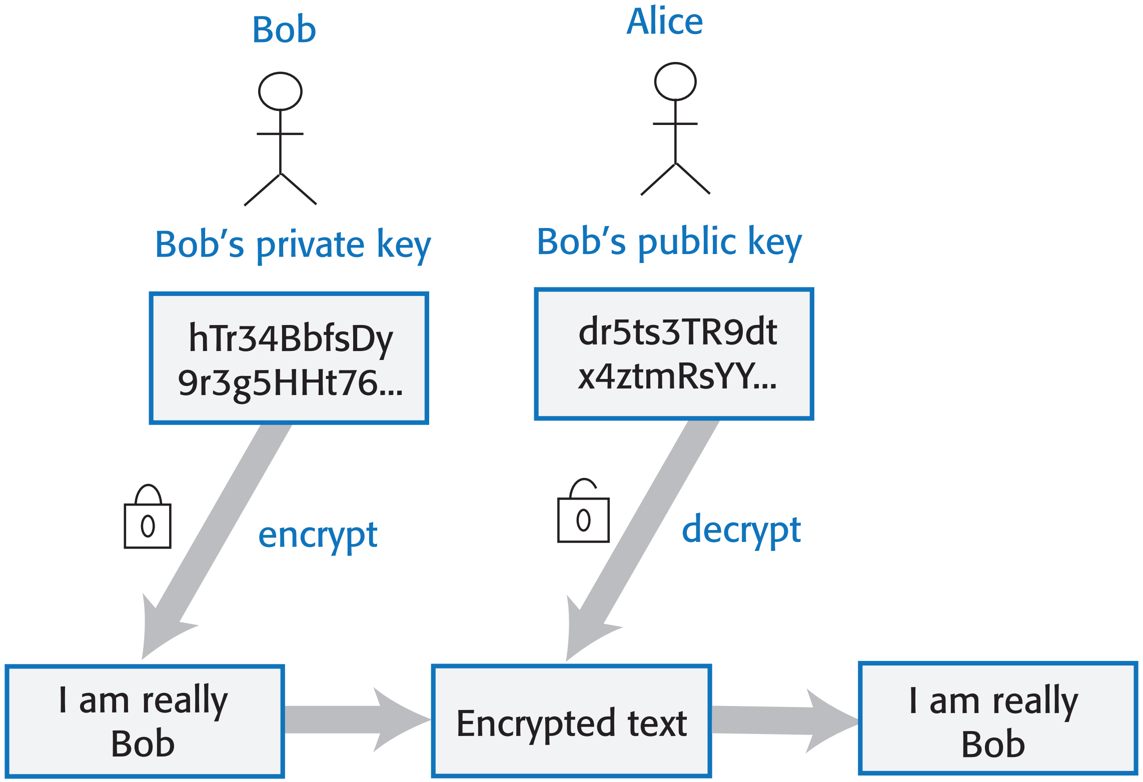
Encryption and authentication


Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

- Asymmetric encryption can also be used to authenticate the sender of a message by encrypting it with a private key and decrypting it with the corresponding public key.
- Say Alice wants to send a message to Bob and she has a copy of his public key.
- However, she is not sure whether or not the public key that she has for Bob is correct and she is concerned that the message may be sent to the wrong person.
- Private/public key encryption can be used to verify Bob's identity.
  - Bob uses his private key to encrypt a message and sends this to Alice. If it can be decrypted using Bob's public key, then Alice has the correct key.

Encryption for authentication

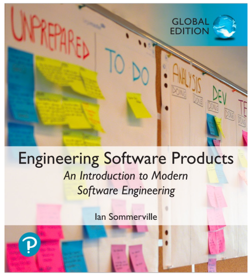




TLS and digital certificates

- The https protocol is a standard protocol for securely exchanging texts on the web. It is the standard http protocol plus an encryption layer called TLS (Transport Layer Security). This encryption layer is used for 2 things:
  - to verify the identity of the web server;
  - to encrypt communications so that they cannot be read by an attacker who intercepts the messages between the client and the server
- TLS encryption depends on a digital certificate that is sent from the web server to the client.
  - Digital certificates are issued by a certificate authority (CA), which is a trusted identity verification service.
  - The CA encrypts the information in the certificate using their private key to create a unique signature. This signature is included in the certificate along with the public key of the CA. To check that the certificate is valid, you can decrypt the signature using the CA's public key.

Digital certificates



- ***Subject information***
  Information about the company or individual whose web site is being visited. Applicants apply for a digital certificate from a certificate authority who checks that the applicant is a valid organization.

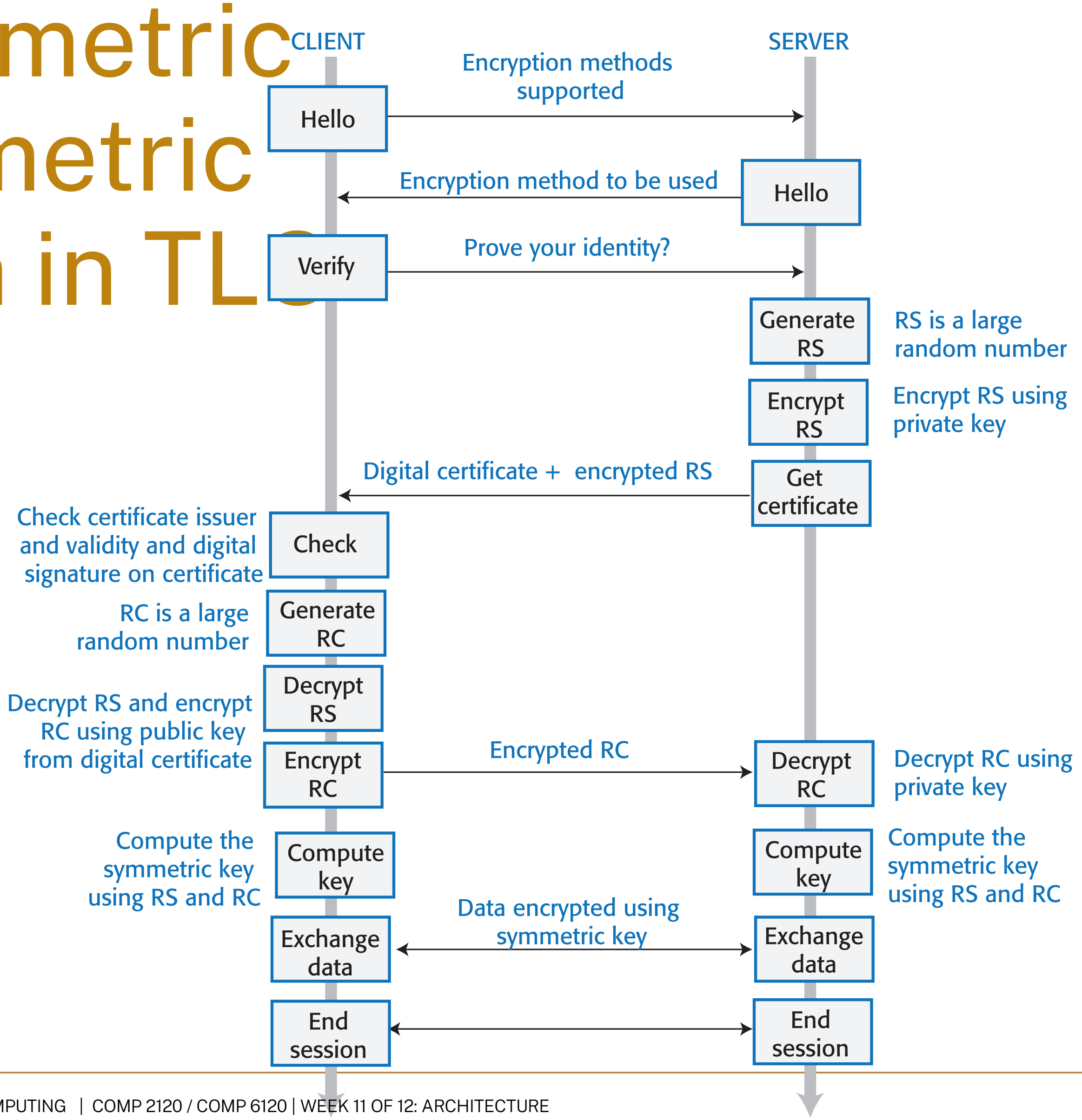
- ***Certificate authority information***
  Information about the certificate authority (CA) who has issued the certificate.

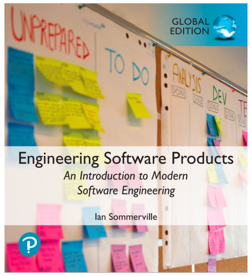
- ***Certificate information***
  Information about the certificate itself, including a unique serial number and a validity period, defined by start and end dates.

- **Digital signature**
  The combination of all of the above data uniquely identifies the digital certificate. The signature data is encrypted with the CA's private key to confirm that the data is correct. The algorithm used to generate the digital signature is also specified.

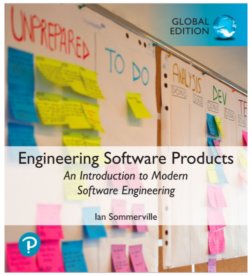
- ***Public key information***
  The public key of the CA is included along with the key size and the encryption algorithm used. The public key may be used to decrypt the digital signature.

Using symmetric and asymmetric encryption in TLS


CLIENT
SERVER
Hello
Encryption methods supported
Hello
Encryption method to be used
Verify
Prove your identity?
Generate RS
RS is a large random number
Encrypt RS
Encrypt RS using private key
Get certificate
Digital certificate + encrypted RS
Check
Check certificate issuer and validity and digital signature on certificate
Generate RC
RC is a large random number
Decrypt RS
Decrypt RS and encrypt RC using public key from digital certificate
Encrypt RC
Encrypted RC
Decrypt RC
Decrypt RC using private key
Compute key
Compute the symmetric key using RS and RC
Compute key
Compute the symmetric key using RS and RC
Exchange data
Data encrypted using symmetric key
Exchange data
End session
End session


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

TLS explained



- The digital certificate that the server sends to the client includes the server’s public key. The server also generates a long random number, encrypts it using its private key and sends this to the client.

- The client can then decrypt this using the server’s public key and, in turn, generates its own long random number. It encrypts this number using the server’s public key and sends it to the server, which decrypts the message using its private key. Both client and server then have two long random numbers.

- The agreed encryption method includes a way of generating an encryption key from these numbers. The client and server independently compute the key that will be used to encrypt subsequent messages using a symmetric approach.

- All client-server traffic is encrypted and decrypted using that computed key. There is no need to exchange the key itself.

Data encryption

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- As a product provider you inevitably store information about your users and, f cloud-based products, user data.

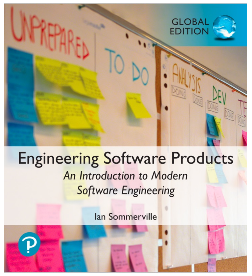
- Encryption can be used to reduce the damage that may occur from data theft. If information is encrypted, it is impossible, or very expensive, for thieves to access and use the unencrypted data.

  - Data in transit.
    When transferring the data over the Internet, you should always use the https rather than the http protocol to ensure encryption.

  - Data at rest.
    If data is not being used, then the files where the data is stored should be encrypted so that theft of these files will not lead to disclosure of confidential information.

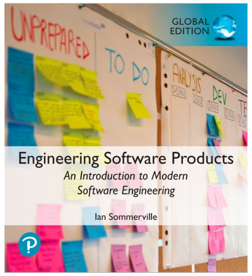
  - Data in use
    The data is being actively processed. Encrypting and decrypting the data slows down the system response time. Implementing a general search mechanism with encrypted data is impossible.

Encryption levels



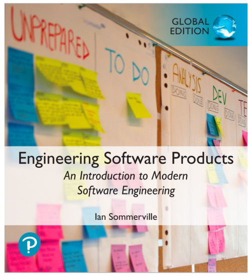
Application	The application decides what data should be encrypted and decrypts that data immediately before it is used.
Database	The DMBS may encrypt the entire database when it is closed, with the database decrypted when it is reopened. Alternatively individual tables or columns may be encrypted/decrypted.
Files	The operating system encrypts individual files when they are closed and decrypts them when they are reopened.
Media	The operating system encrypts disks when they are unmounted and decrypts these disks when they are remounted.

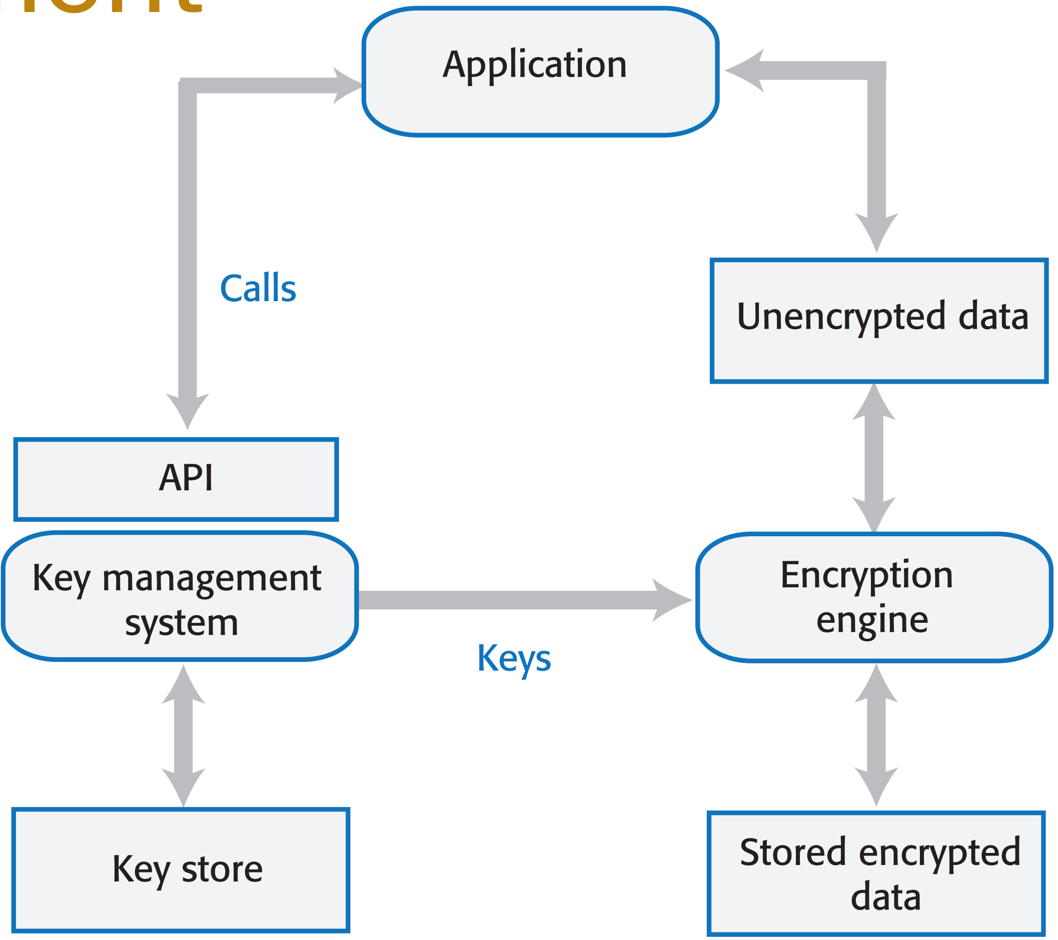
Key management


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

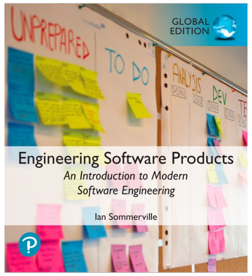
- Key management is the process of ensuring that encryption keys are securely generated, stored and accessed by authorized users.
- Businesses may have to manage tens of thousands of encryption keys so it is impractical to do key management manually and you need to use some kind of automated key management system (KMS).
- Key management is important because, if you get it wrong, unauthorized users may be able to access your keys and so decrypt supposedly private data. Even worse, if you lose encryption keys, then your encrypted data may be permanently inaccessible.
- A key management system (KMS) is a specialized database that is designed to securely store and manage encryption keys, digital certificates and other confidential

Using a KMS for encryption management





Long-term key storage



Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

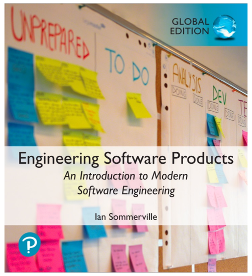
- Business may be required by accounting and other regulations to keep copies of all of their data for several years.
  - For example, in the UK, tax and company data has to be maintained for at least six years, with a longer retention period for some types of data. Data protection regulations may require that this data be stored securely, so the data should be encrypted.
- To reduce the risks of a security breach, encryption keys should be changed regularly. This means that archival data may be encrypted with a different key from the current data in your system.
- Therefore, key management systems must maintain multiple, timestamped versions of keys so that system backups and archives can be decrypted if required.

Privacy

Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
GLOBAL EDITION

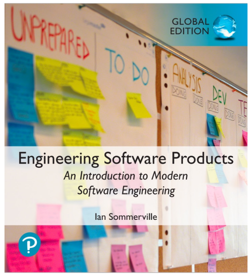
- Privacy is a social concept that relates to the collection, dissemination and appropriate use of personal information held by a third-party such as a company or a hospital.
- The importance of privacy has changed over time and individuals have their own views on what degree of privacy is important.
- Culture and age also affect peoples' views on what privacy means.
  - Younger people were early adopters of the first social networks and many of them seem to be less inhibited about sharing personal information on these platforms than older people.
  - In some countries, the level of income earned by an individual is seen as a private matter; in others, all tax returns are openly published.

Business reasons for privacy


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

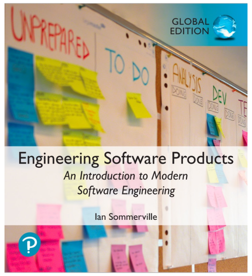
- If you are offering a product directly to consumers and you to conform to privacy regulations, then you may be subject to legal action by product buyers or by a data regulator. If your conformance is weaker than the protection offered by data protection regulations in some countries, you won't be able to sell your product in these countries.
- If your product is a business product, business customers require privacy safeguards so that they are not put at risk of privacy violations and legal action by users.
- If personal information is leaked or misused, even if this is not seen as a violation of privacy regulations, this can lead to serious reputational damage. Customers may stop using your product because of this.

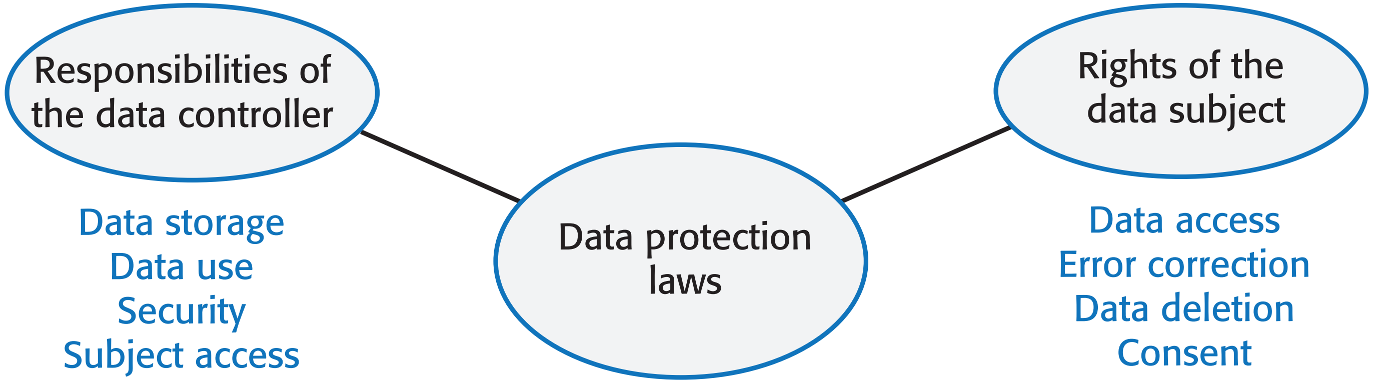
Data protection laws


GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- In many countries, the right to individual privacy is protected by data protection laws.
- These laws limit the collection, dissemination and use of personal data to the purposes for which it was collected.
  - For example, a travel insurance company may collect health information so that they can assess their level of risk. This is legal and permissible.
  - However, it would not be legal for those companies to use this information to target online advertising of health products, unless their users had given specific permission for this.

Data protection laws





Data protection principles (1)

GLOBAL EDITION

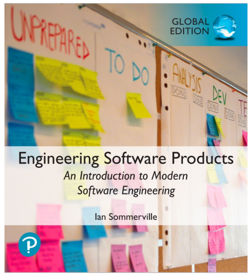
Engineering Software Products

An Introduction to Modern Software Engineering

Ian Sommerville

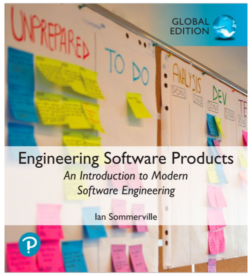
- ***Awareness and control***
  Users of your product must be made aware of what data is collected when they are using your product, and must have control over the personal information that you collect from them.
- ***Purpose***
  You must tell users why data is being collected and you must not use that data for other purposes.
- ***Consent***
  You must always have the consent of a user before you disclose their data to other people.
- ***Data lifetime***
  You must not keep data for longer than you need to. If a user deletes their account, you must delete the personal data associated with that account.

Data protection principles (2)



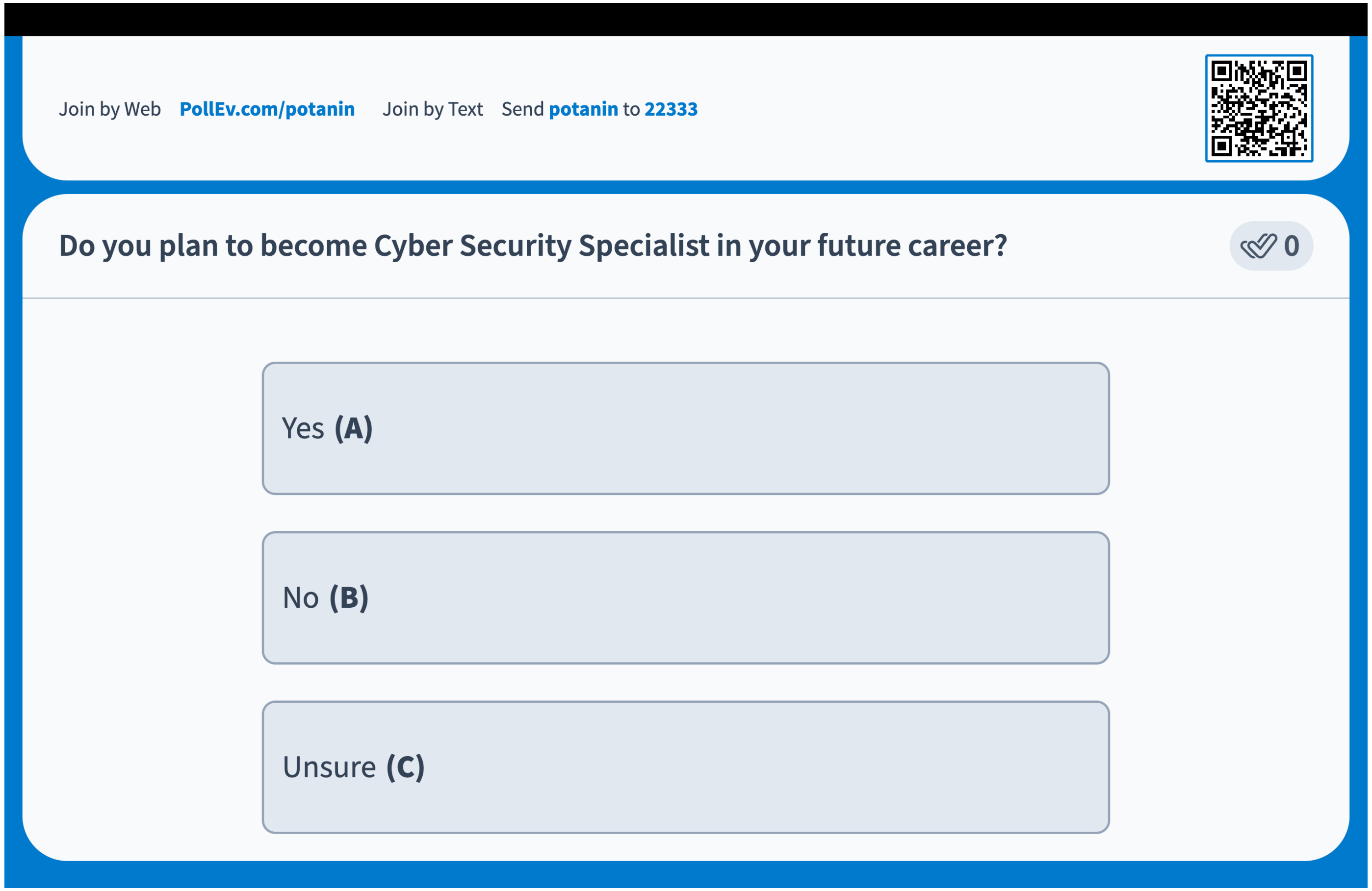
- **Secure storage**
  You must maintain data securely so that it cannot be tampered with or disclosed to unauthorized people.
- **Discovery and error correction**
  You must allow users to find out what personal data that you store. You must provide a way for users to correct errors in their personal data.
- ***Location***
  You must not store data in countries where weaker data protection laws apply unless there is an explicit agreement that the stronger data protection rules will be upheld.

Privacy policy


UNPREPARED
TO DO
ANALYSIS
DEV
GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

- You should to establish a privacy policy that defines how personal and sensitive information about users is collected, stored and managed.
- Software products use data in different ways, so your privacy policy has to define the personal data that you will collect and how you will use that data.
- Product users should be able to review your privacy policy and change their preferences regarding the information that you store.
- Your privacy policy is a legal document and it should be auditable to check that it is consistent with the data protection laws in countries where your software is sold.
- Privacy policies should not be expressed to users in a long 'terms and conditions' document that, in practice, nobody reads.
- The GDPR now require software companies to include a summary of their privacy policy, written in plain language rather than legal jargon, on their website.

Poll Everywhere Time!



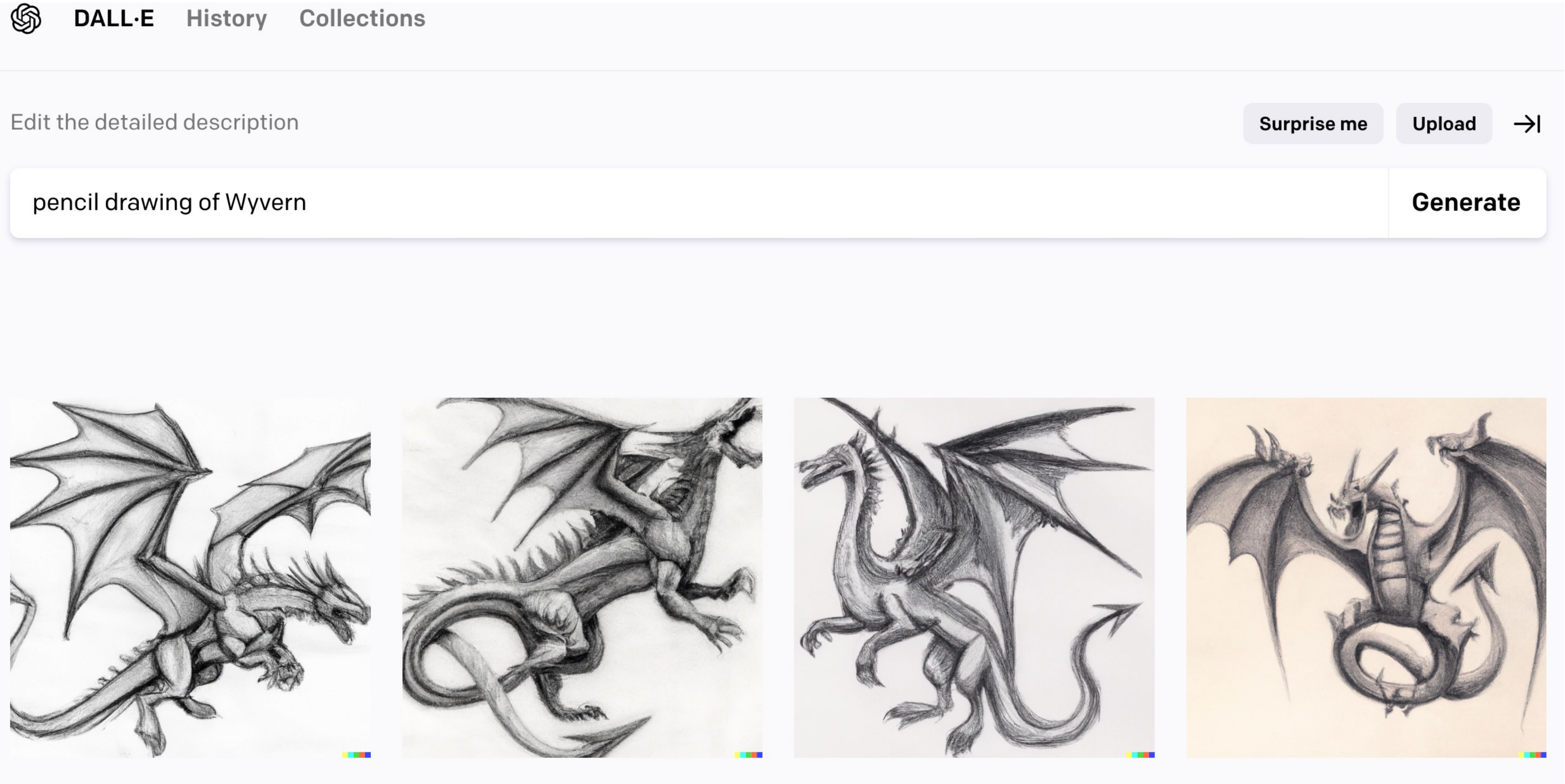
Join by Web **PollEv.com/potanin** Join by Text Send **potanin** to **22333**

Do you plan to become Cyber Security Specialist in your future career?

Yes **(A)**

No **(B)**

Unsure **(C)**

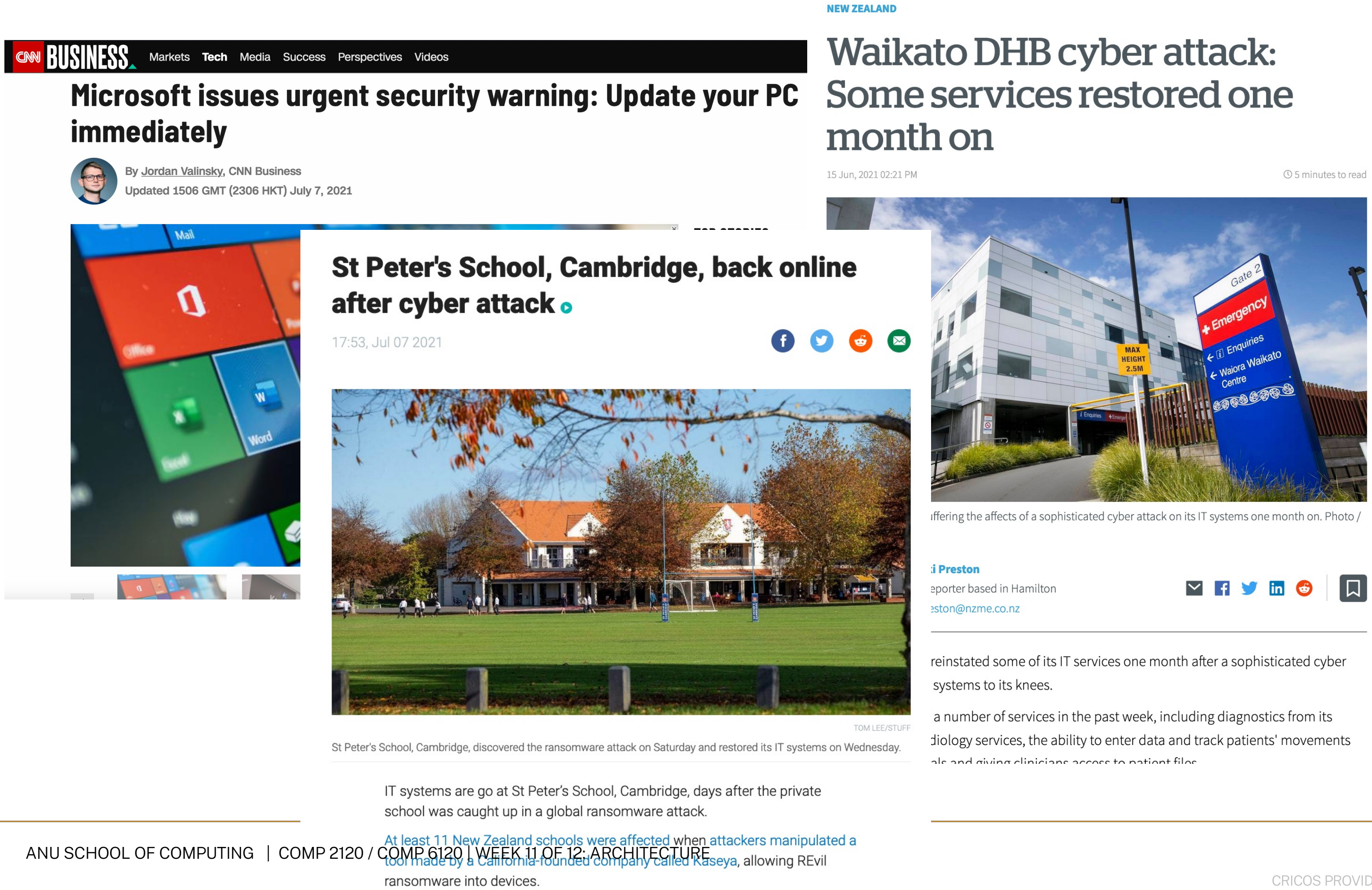


Edit the detailed description

pencil drawing of Wyvern

Wyvern

Software Security is a Big Problem



CNN BUSINESS Markets Tech Media Success Perspectives Videos

Microsoft issues urgent security warning: Update your PC immediately

By Jordan Valinsky, CNN Business
Updated 1506 GMT (2306 HKT) July 7, 2021

St Peter's School, Cambridge, back online after cyber attack

17:53, Jul 07 2021

TOM LEE/STUFF

St Peter's School, Cambridge, discovered the ransomware attack on Saturday and restored its IT systems on Wednesday.

IT systems are go at St Peter's School, Cambridge, days after the private school was caught up in a global ransomware attack.

At least 11 New Zealand schools were affected when attackers manipulated a tool made by a California-founded company called Kaseya, allowing REvil ransomware into devices.

NEW ZEALAND

Waikato DHB cyber attack: Some services restored one month on

15 Jun, 2021 02:21 PM

5 minutes to read

iffering the affects of a sophisticated cyber attack on its IT systems one month on. Photo /

i Preston
eporter based in Hamilton
eston@nzme.co.nz

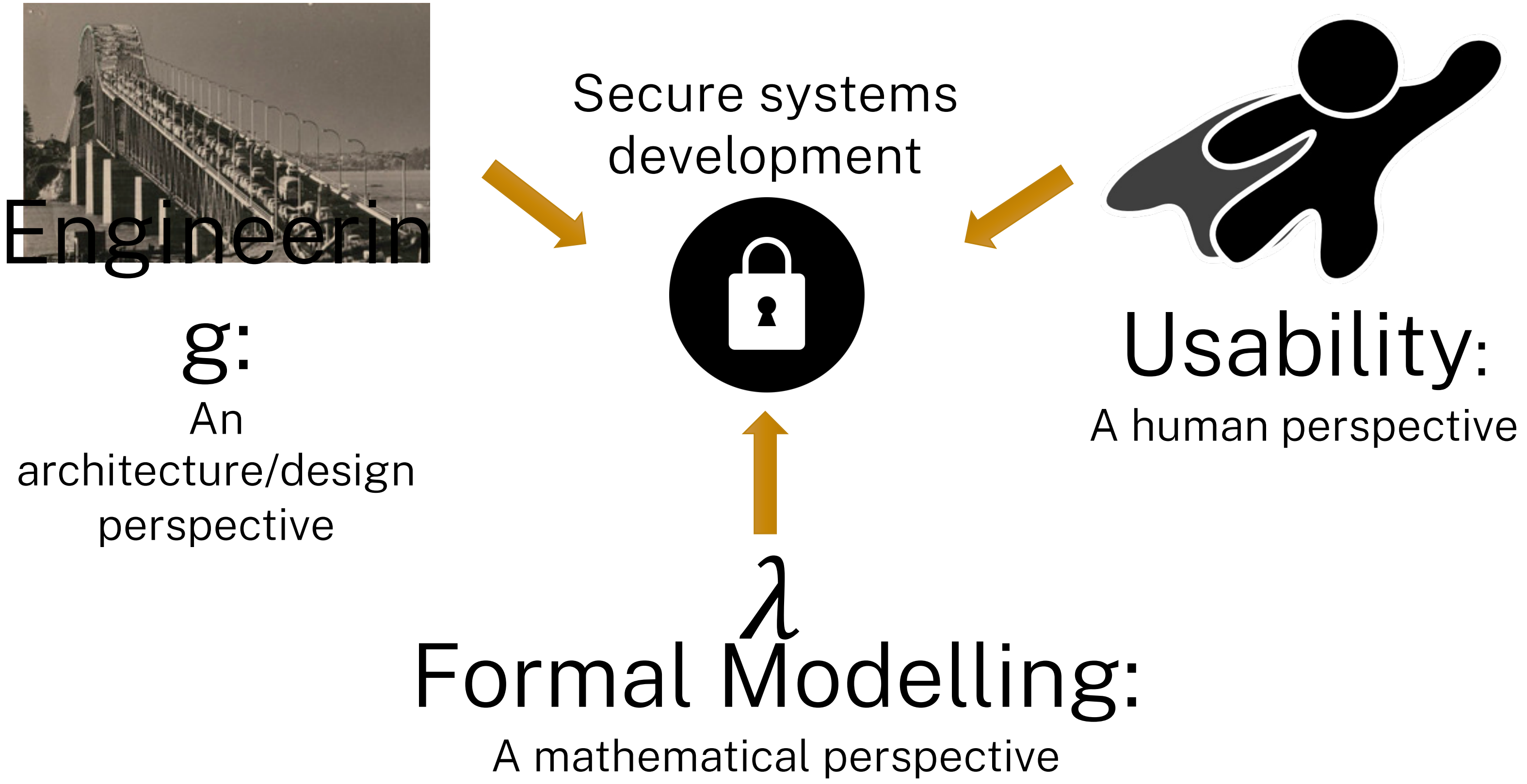
reinstated some of its IT services one month after a sophisticated cyber systems to its knees.

a number of services in the past week, including diagnostics from its diology services, the ability to enter data and track patients' movements als and giving clinicians access to patient files

Why Systems are Vulnerable?

- We “know” how to code securely
  - Follow the rules: CERT, Oracle, …
  - Technical advances: types, memory safety
- But we still fail too often!
- Root causes
  - Coding instead of engineering
  - Human limitations
  - Unusable tools

Our Approach:
Usable Architecture-Based Security



Secure systems development

Engineerin
g:

An architecture/design perspective

Usability:

A human perspective

λ

Formal Modelling:

A mathematical perspective

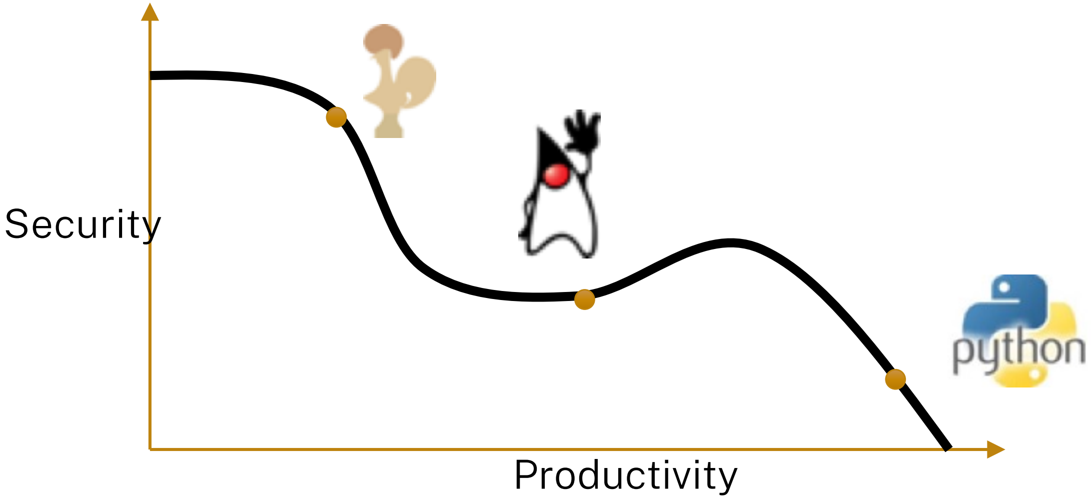
The Wyvern Programming Language

- Designed for security and productivity from the ground up
- General purpose, but emphasising web, mobile, and IoT apps

http://wyvernlang.github.io/

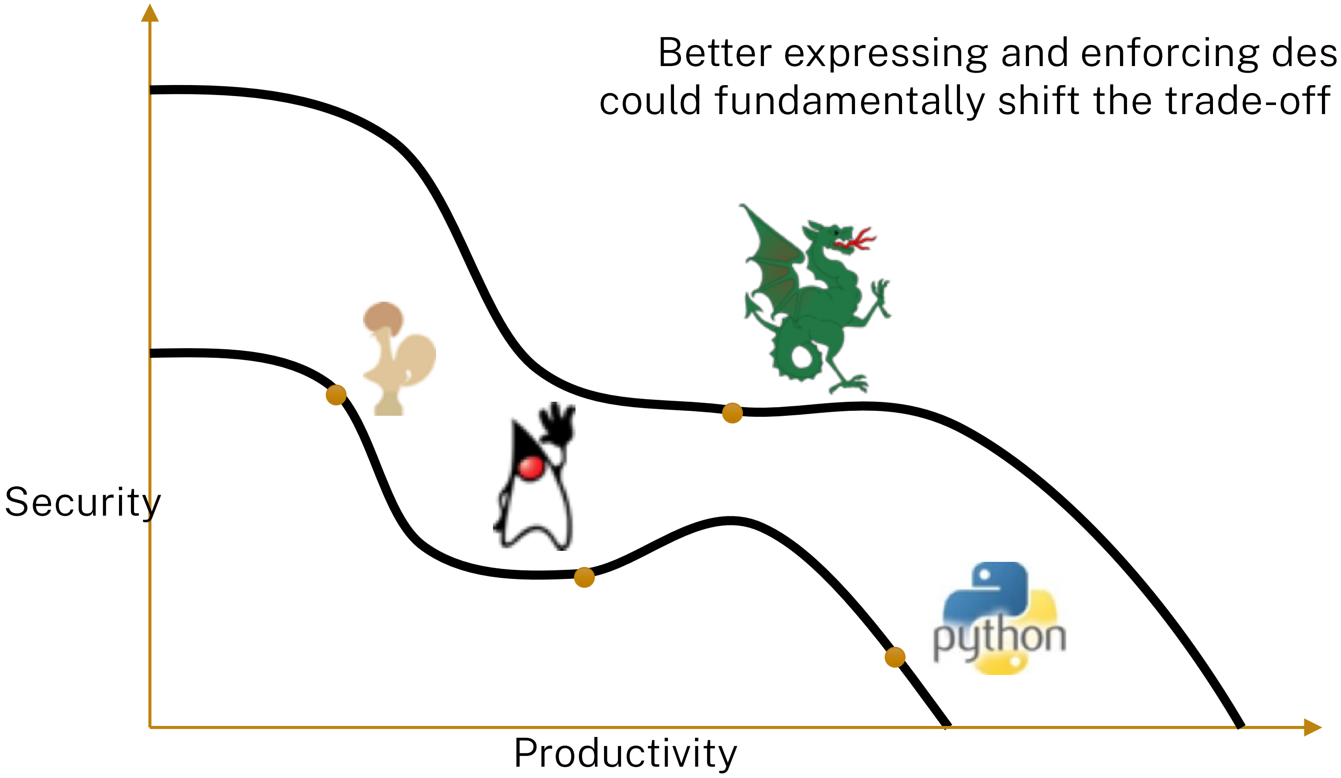
The Wyvern Programming Language

- But you might ask: “Isn’t there a trade-off between security and productivity?
- What is Wyvern’s secret sauce?



DYNAMIC TYPING
STATIC TYPING
DONE!

Shifting the Trade-off Curve



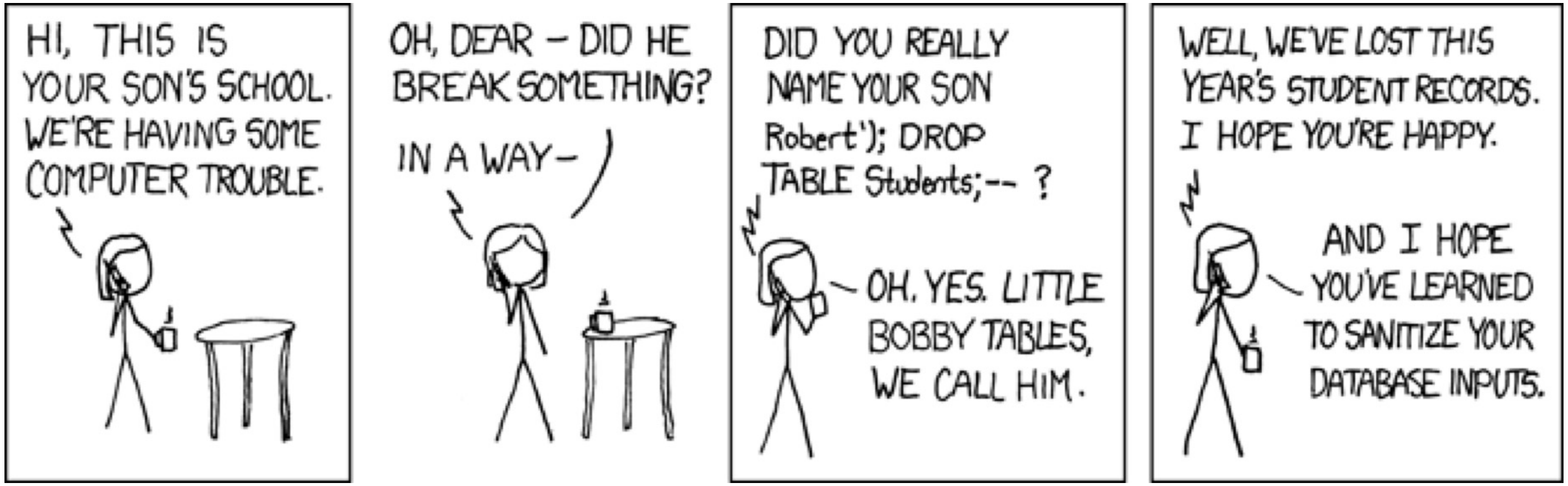
The Wyvern Programming Language

- Design goals
  - Sound, modern language design
    - Type- and memory- safe, mostly functional, advanced module system
  - Incorporate usability principles
  - Security mechanisms built in

Hello, world!

```
require stdout
stdout.print(“Hello, world!\n”)
```

SQL Command Injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

SQL Injection: a Solved Problem?

```
PreparedStatement s = connection.prepareStatement(
      “SELECT * FROM Students WHERE name = ?;”);
s.setString(1, userName);
s.executeQuery();
```

Prepare a statement with a hole

Fill the hole securely

Evaluation:

✗ Usability: unnatural, verbose

✗ Design: string manipulation captures domain poorly

✗ Language semantics: largely lost – just strings

- No type checking, IDE services, …

Wyvern: Usable Secure Programming

- SQL query in Wyvern

Introduces a domain-specific language (DSL) on the next indented lines

```
connection.executeQuery(~)
      SELECT * FROM Students WHERE name = {studentName}
```

Semantically rich DSL. Can provide type checking, syntax highlighting, autocomplete, …

Safely incorporates dynamic data – as data, not a command

- Claim: the secure version more natural and more usable
    - no empirical evaluation, but can be a project!

Technical Challenge: Syntax Conflicts

λ

- Language extensions as libraries has been tried before
  - Example: SugarJ/Sugar* [Erdweg et al, 2010; 2013]

```
import XML, HTML
val snippet = ~
      How do I <b>parse</b> this example?
```

Is it XML or HTML?

Syntax Conflicts: Wyvern Solution

`metadata` keyword indicates we are importing syntax, not just a library

```
import metadata XML, HTML

val snippet : XML = ~
      How do I <b>parse</b> this example?
```

No ambiguity: the compiler loads the unique parser associated with the expected type XML

Syntax of language completely unrestricted – indentation separates from host language

Technical Challenge: Semantics

λ

Q: Is it safe to run custom parser at compiler time?
A: Yes – immutability types used to ensure imported metadata is purely functional, has no network access, etc.

```
import metadata SQL
val connection = SQL.connect(...)
val studentName = input(...)
connection.executeQuery(~)
      SELECT * FROM Students WHERE name = {studentName}
```

Language definition includes custom type checker – can verify query against database schema

SQL extension has access to variables and their types in Wyvern host language

Example

```
serve : (URL, HTML) -> ()
```

base language
URL TSL
HTML TSL
CSS TSL
String TSL
SQL TSL

```
serve(`products.nameless.com`, ~)
  :html
    :head
      :title Product Listing
      :style ~
        body { font-family: %bodyFont% }
    :body
      :div[id=“search”]
        {SearchBox(“Products”)}
      :ul[id=“products”]
        {items_from_query(query(db,
           <SELECT * FROM products COUNT {n_products}>))}
```

Our Approach: Usable Architecture-Based Security

Engineering:

Express design in domain-specific way

DSL support in Wyvern

Usability:

Natural syntax, enabling IDE support

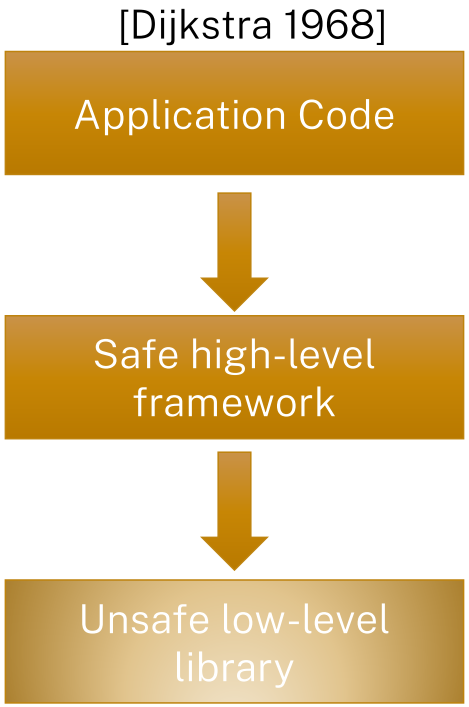
λ

Formal Modelling:

Type safety, variable hygiene, conflict-free extensions

An Old Idea: Layered Architectures

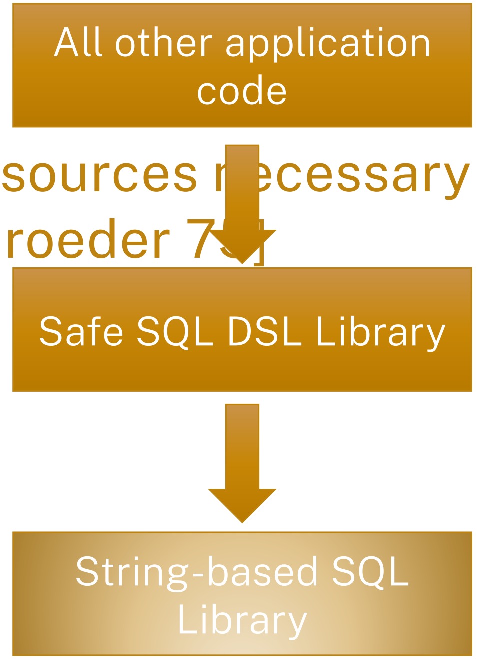
- Lowest layer: an unsafe, low-level library
- Middle layer: a higher-level framework
- Top layer: the application
- Code must obey strict layering
- Many variants:
  - Secure networking framework
  - Safe SQL-access library
  - Replicated storage library



- RQ: Can we use capabilities to enforce layered resource access?

Architecture:
Principle of Least Privilege

“Every module must be able to access only the resources necessary for its legitimate purpose.” [Saltzer & Schroeder 73]



All other application code

Safe SQL DSL Library

String-based SQL Library

Module Linking as Architecture

```
require db.stringSQL
```

To access external resources like a database, main requires a capability from the run-time system. A capability is an unforgeable token controlling access to a resource.

```
application.run()
```

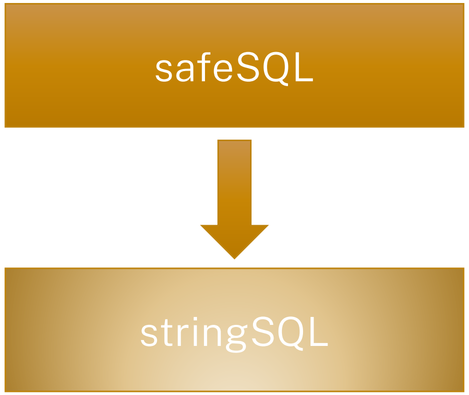
stringSQL

Module Linking as Architecture

We can import code modules, but they have no ambient authority to access resources (cf Newspeak). `sqlApplication` cannot access the database by itself.

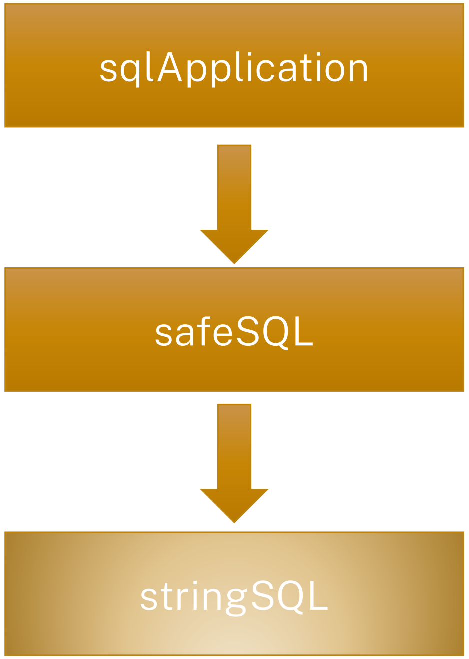
```
require db.stringSQL

import db.safeSQL
import app.sqlApplication

val sql = safeSQL(stringSQL)
val application = sqlApplication(sql)

application.run()
```

We must instantiate a `sqlApplication` object, passing it the resources it needs. We pass only a capability to the safe library.



Module Linking as Architecture

```
module def sqlApplication(safeSQL : db.SafeSQL)
def run() : Int
       // application code

require db.stringSQL

import db.safeSQL
import app.sqlApplication

val sql = safeSQL(stringSQL)
val application = sqlApplication(sql)

application.run()

module def safeSQL(strSQL : db.StringSQL)
// implement ADT in terms of strings
```



How Hard to Link it All Up?

- Most Wyvern modules don’t have state, can be freely imported
- Statically tracked: stateful modules/objects and resource types

Provides access to OS resource

```
resource type File
     def write(s : String)
```

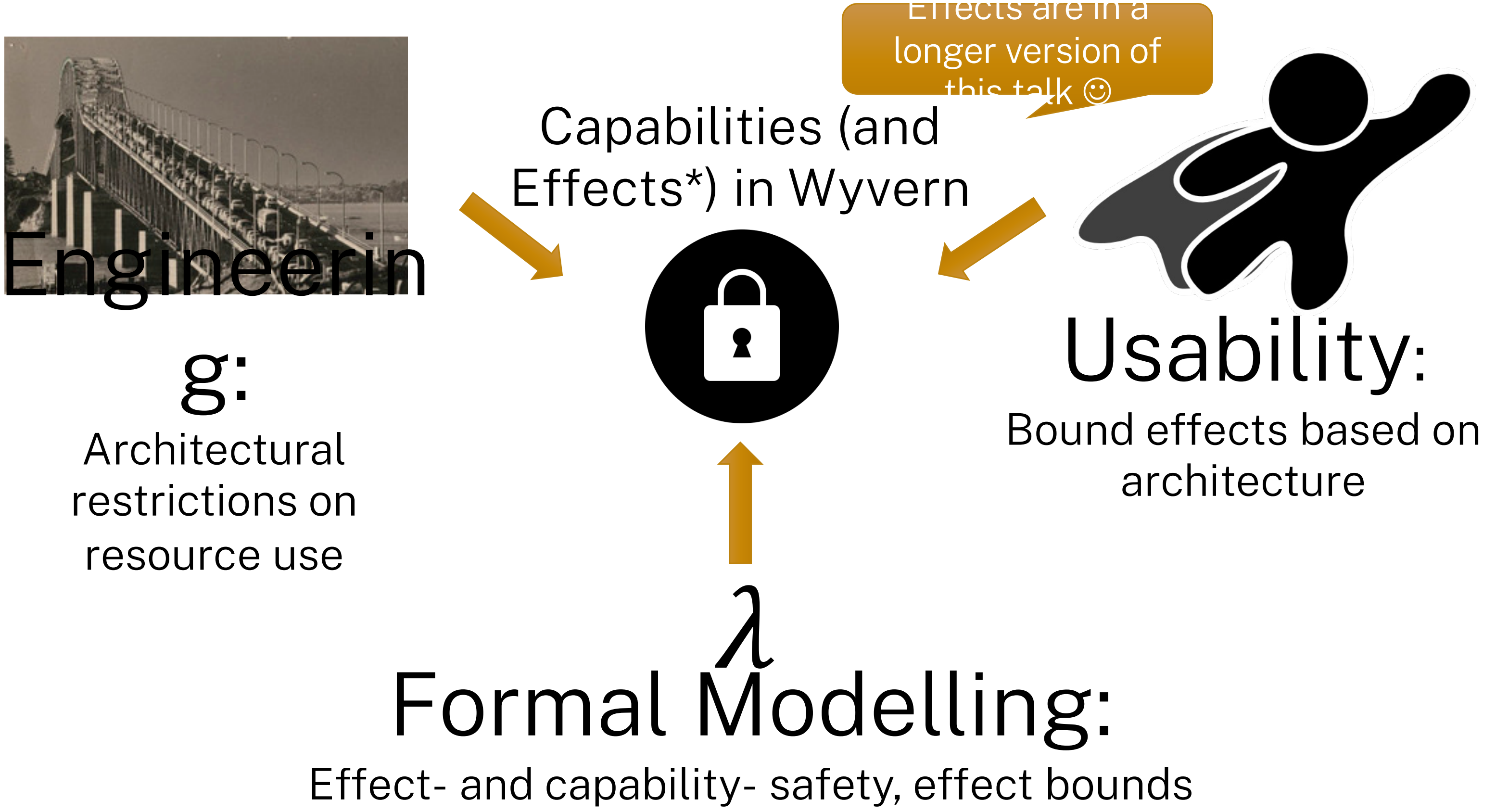
```
type SetM
        resource type Set
                def add(v : Int)
                def isMember(v : Int) : Bool
        def makeSet() : Set

module setM : SetM

module def client(aFile : File)
import setM ...
```

Type of modules is pure, no static state. Objects created by module may be stateful resources, though

Resources must be passed in, pure modules can just be imported

- resource types capture state or system access: other types do not
  - Useful design documentation; e.g. MapReduce tasks should be stateless
  - Supports powerful equational reasoning, safe concurrency, etc.

Our Approach:
Usable Architecture-Based Security



Effects are in a longer version of this talk ☺

Capabilities (and Effects*) in Wyvern

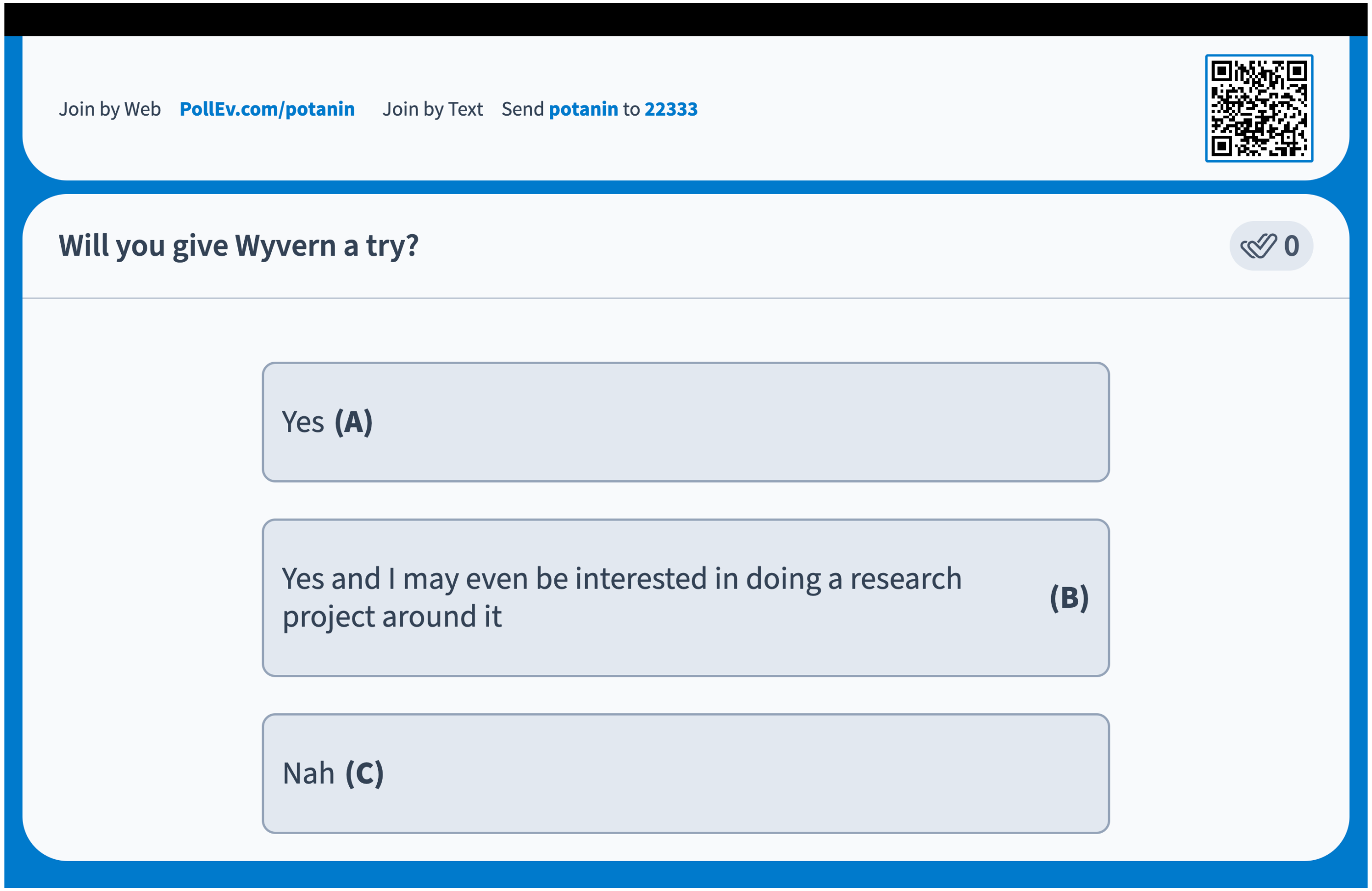
Engineering:
Architectural restrictions on resource use

Usability:
Bound effects based on architecture

λ
Formal Modelling:
Effect- and capability- safety, effect bounds

Questions?

Poll Everywhere Time!



Join by Web **PollEv.com/potanin** Join by Text Send **potanin** to **22333**

Will you give Wyvern a try? ✓ 0

- Yes **(A)**
- Yes and I may even be interested in doing a research project around it **(B)**
- Nah **(C)**