20220406-软件架构&算法导论

1.过程描述

1.1 Software architecture

1)4+1view model

Logical view：

- focused mostly on achieving the **functional requirements** of a system.The context is the services that should be provided to end user
- defining all of the **classes**; their attributes; their behaviors
- showing the **relationship** between **software objects and components**
- **State diagram** and **classes disgram** are most commonly used

Process view

- focus on achieving nonfunctional requirements which specify the desired qualities for the system, including quality attributes like performance and availability
- show the **execution order** of different objects, and the calls to methods defined by the logical view in the correct order. Behaviors that are asynchronous or concurrent are also described
- **UML sequence diafram** and **UML activity diagram** are often used

Development view

- consider things like **programming languages, libraries and tool sets**
- also includes management details like scheduling, budgets and work assignments. Especially project management

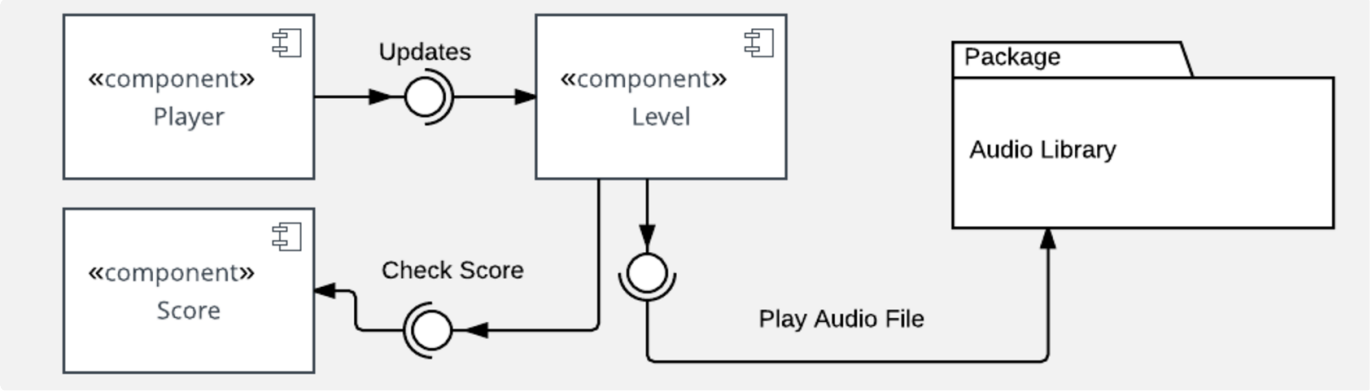
Physical view

- handles how elements in the logical process and development view to be mapped to different **nodes** of **hardware** for running the system.
- **UML deployment diagram** can express

Scenerios

- align with the use cases or user tasks of a system
- how the four other views work together
- often use script that describes **the sequence of interactions between objects and processes**

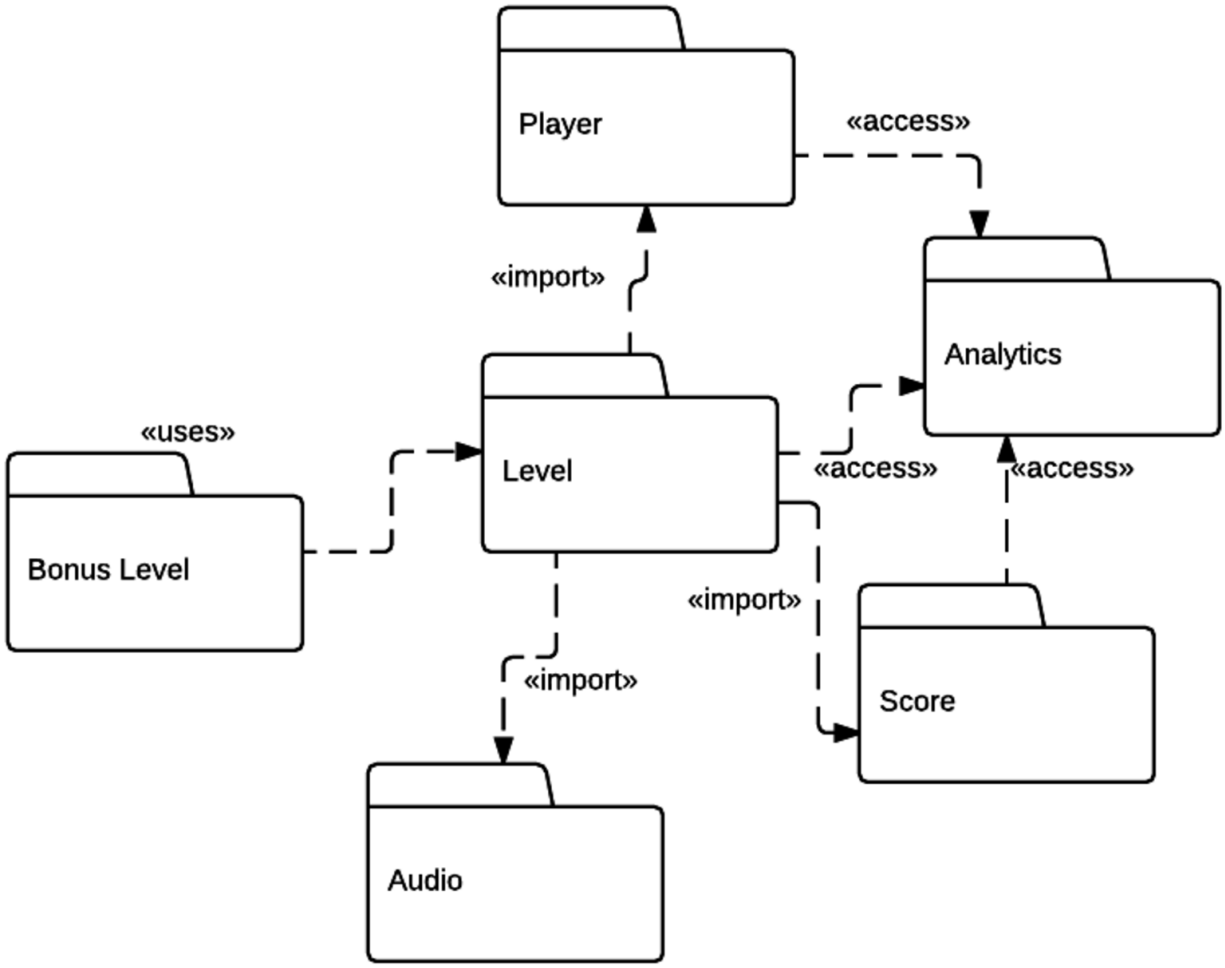
2)Component diagram



- the basis are component and their relationships
- a ball connector, is how you display a provided interface in component diagrams. the purpose of a provided interface is to show that **a component offers an interface for others to interact with it**
- a socket connector, display a required interface. to show that **a component expects a certain interface**

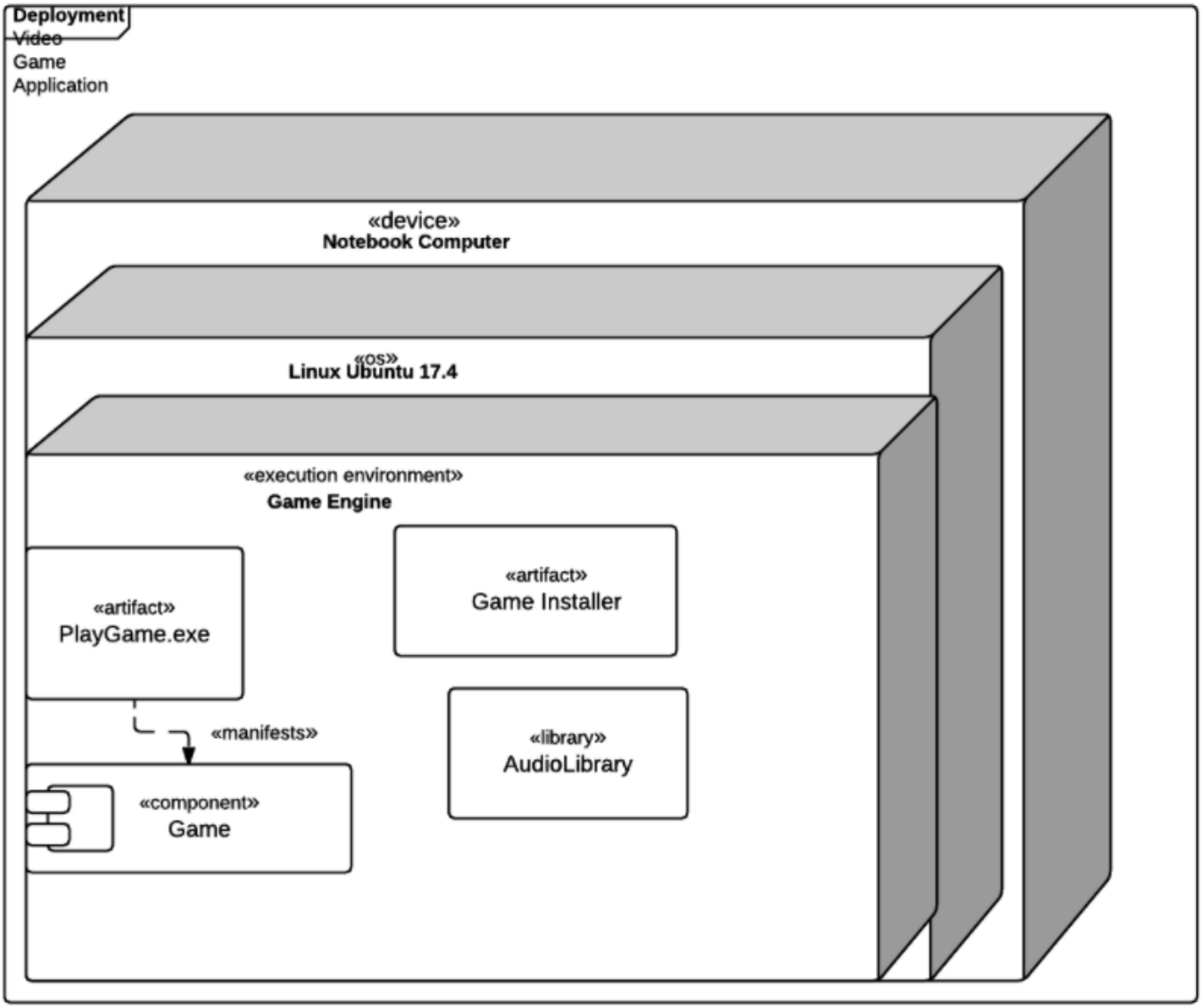
- first indentify the **main objects** used in the system, then indentify all of the **relevant libraries** needed for the system

3)UML Package diagram



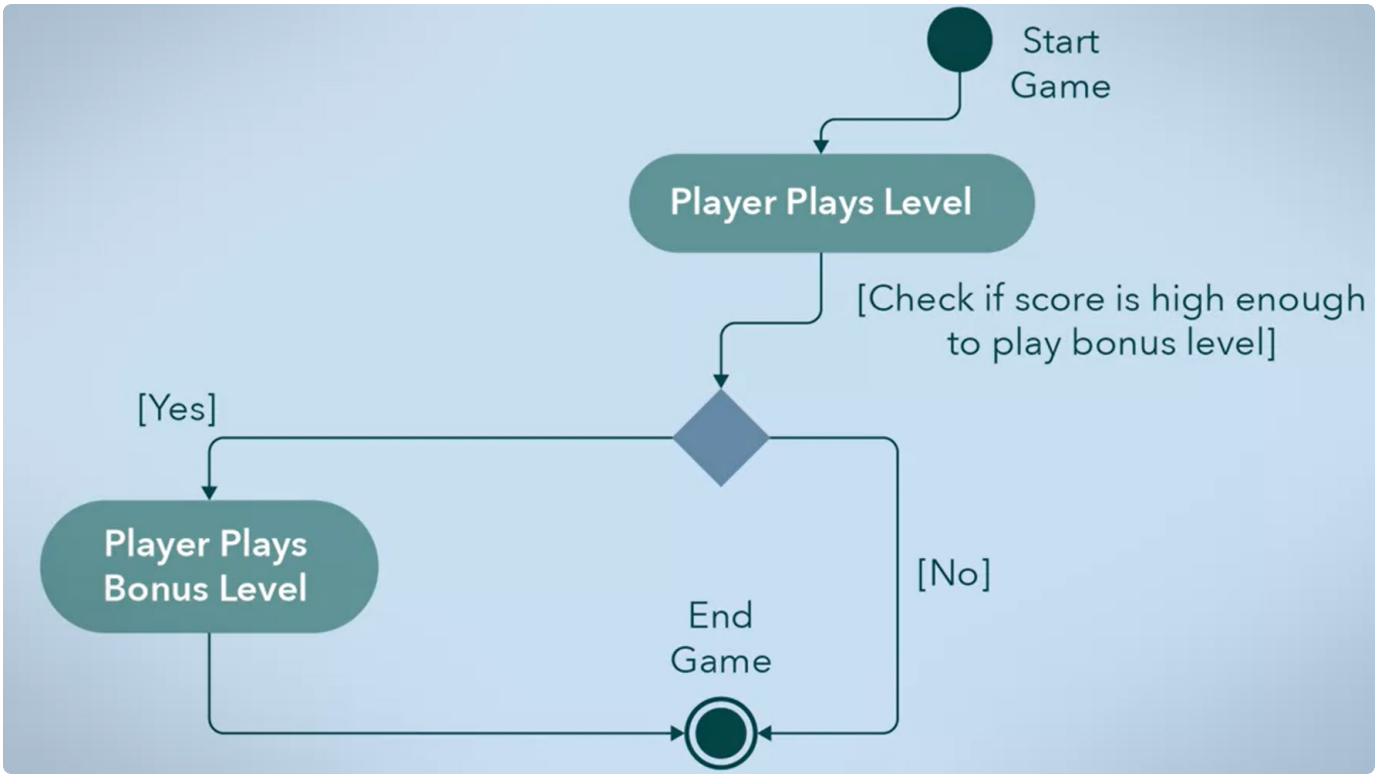
- A package **groups together elements** of your software that are related based on data, classes or user tasks. Also defines a **namespace** for the elements it contains
- Package diagram shows **packages and the dependencies** between them
- In the package, certain elements can be marked as public or private based on the + and – signs in front of their names
- The **import** tag says the import is public, the **access** tag says the import is private, no need to make names in the imported further known outside the namespace

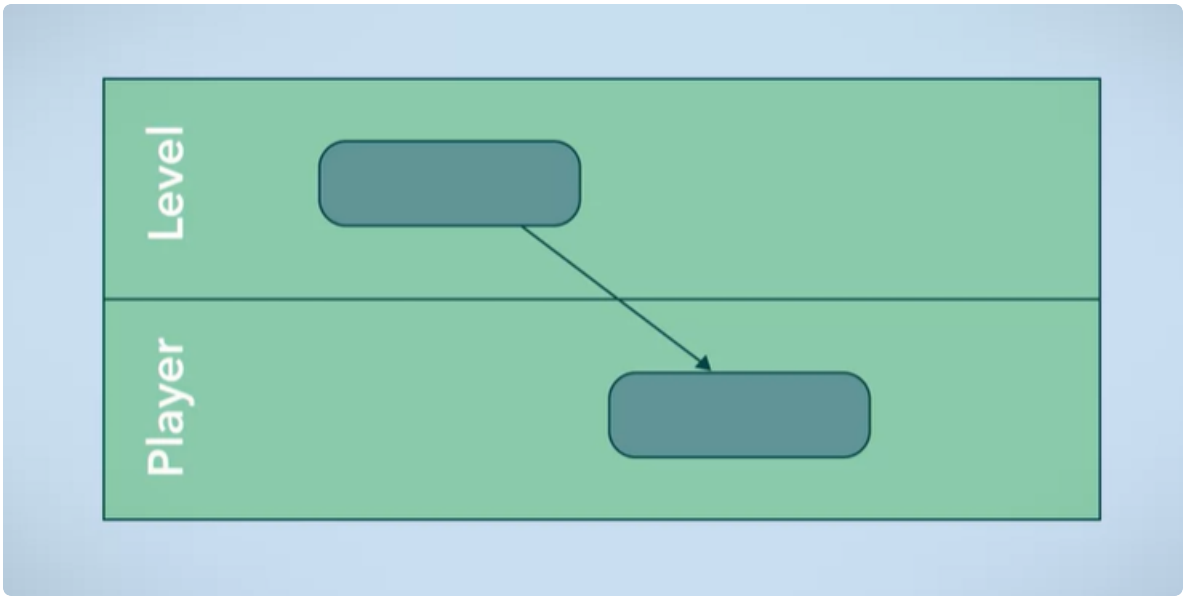
4)UML Deployment diagram



- The deployment digram gives a high level look at the **artifacts, libraries, main components, machines and devices** that your application needs to run
- **Artifact** is a physical result of the development process, final pieces to put together
- **specification level diagram** gives an overview of artifacts and deployment targets, without referencing specific details like machine names
- **instance level diagram** is a much more specific approach, which map a specific artifact to a specific deployment target, and can indentify specific machines and hardware devices
- **Nodes(the cube)** are deployment targets that contain artifacts available for execution. Hardware devices are also displayed in the same way as nodes
- Manifestation is where an artifact is a **physical realization of a software component**

5)UML Activity diagram



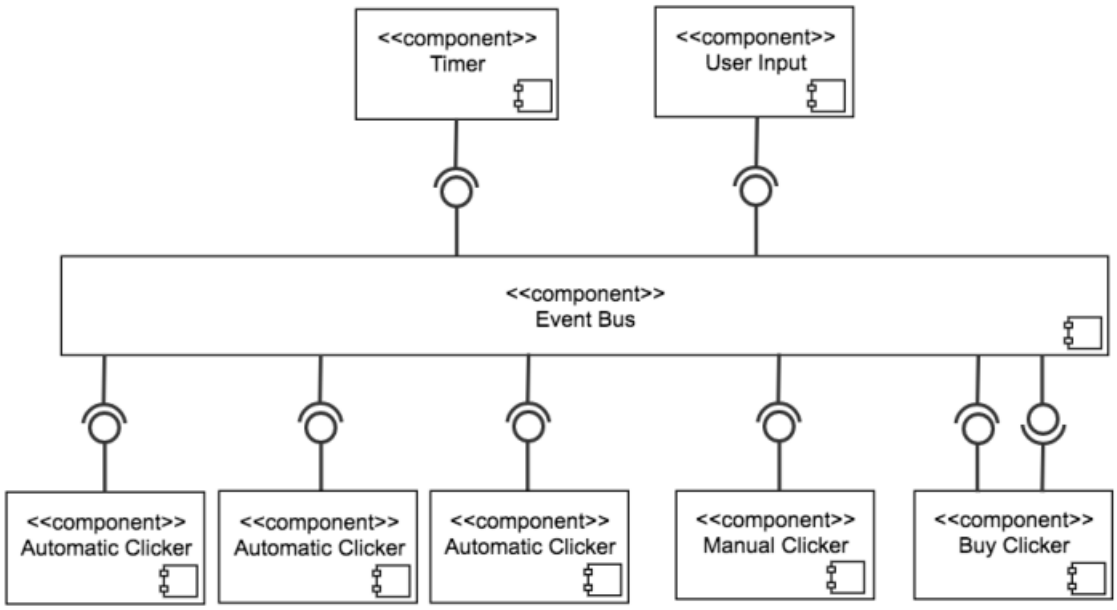


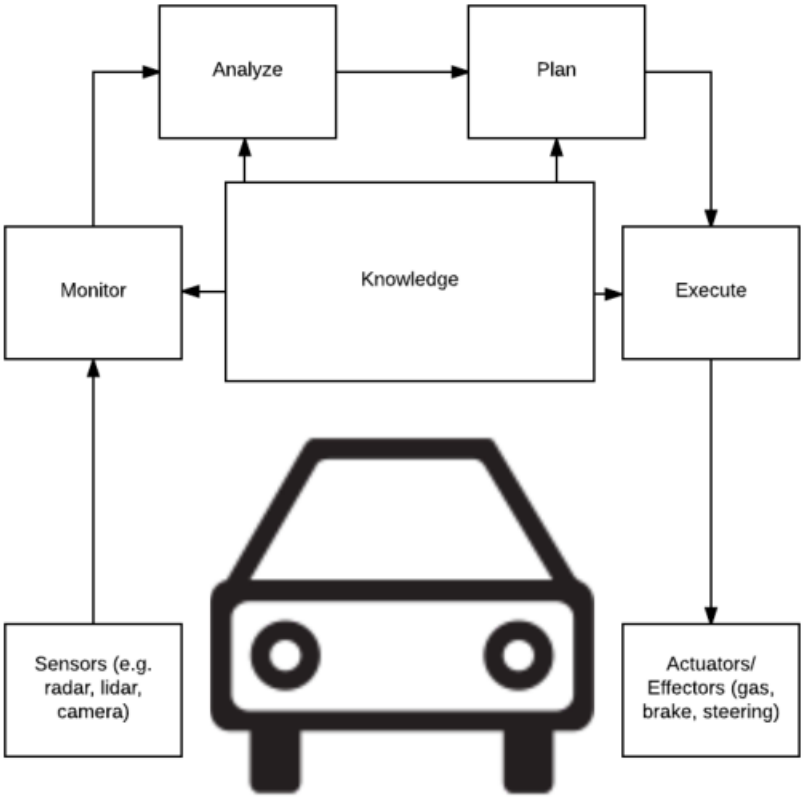
- In the activity diagram, you represent the **flow** from one activity to another in a software system. The purpose is to capture the dynamic behavior of the system, how control flows from one activity to another
- **activities** and **conditions** are two essiential parts
- there can be **parallel flows**
- **Partition** divide activities up into different catefories, such as where it occurs or the user role involved. **Swim lanes** are used to dislplay these partitions

6)Architectural styles

Name	Definition	Keypoints
language–based systems	types: • abstract data types and object–oriented architectural style • Main program and subroutine	inherience is allowed in object–oriented sytle
repository–based systems	data–centric, allow data to be stored and shared between multiple components. Can be achieved by integrating a method of shared storage such as a databse	
layered systems	layer is a collection of components that work together towards a common purpose. Components only interact with components in their own layer or adjacent layers	• top layer have no authority to allocate system resources or communicate with hardware directly. This "sandboxing" provides security and reliability to the kernel • loosely coupled, follows the principle of least konwledge
Client Server n–tier	tiers refer to components that are typically on different physical machines. relationship between 2 adjacent tiers is often a client/server relationship	• A tier can be both server and client • request–response relationships can be synchronous or aysnchronous • requires extra resources to manage the client/server relationships

Interpreter–based systems	allow end users to write scripts, macros or rules that access, compose and run the basic features of those systems. Encourage customization	portable java is a example
Dataflow systems	pipes and filters architectural style. perform transformation on data	
Implicit invocation systems	event–based architectural style. Events are both indicators of change in the system and triggers to functions. Events can be signal, user inputs, messages and etc	• functions take the form of event generators(send events) and event consumers(receive and process events). a funcation can be both • communication between functions is mediated by an event bus, so it is called implicit invocation • each event consumer registers with the event bus to notified of certain events • when the bus(always listening) detect an event, it distributes the event to consumers • 2 consumers can running at the same time on shared data. Semaphore is used to coordinated this process. Semaphore is a variable or abstract data type that toggles between 2 values.
Process control systems	feedback loops with sensor, controller, actuator and the process itself	





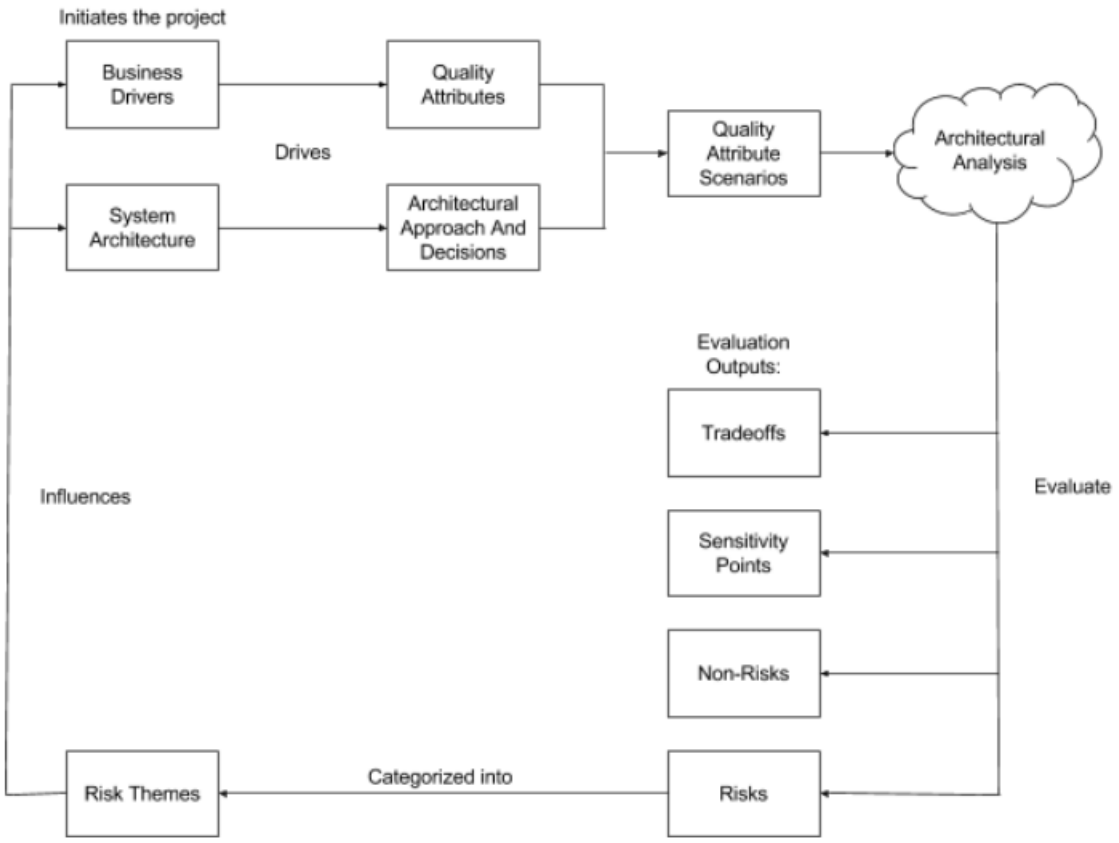
7) Quality attributes

Quality Attribute	Measurement of
Maintainability	The ease at which your system is capable of undergoing changes. Systems will undergo changes throughout its life cycle, so a system should be able to accommodate these changes with ease.
Reusability	The extent in which functions or parts of your system can be used in another. Reusability helps reduce the cost of re-implementing something that has already been done.
Flexibility	How well a system can adapt to requirements change. A highly flexible system can adapt to future requirements changes in a timely and cost-efficient manner.
Modifiability	The ability of a system to cope with changes, incorporate new, or remove existing functionality. This is the most expensive design quality to achieve, so cost of implementing changes must be balanced with this attribute.
Testability	How easy it is to demonstrate errors through executable tests. Systems should be tested as tests can be done quickly, easily, and do not require a user interface. This will help identify faults so that they might be fixed before system release.
Conceptual Integrity	The consistency across the entire system, such as following naming and coding conventions.

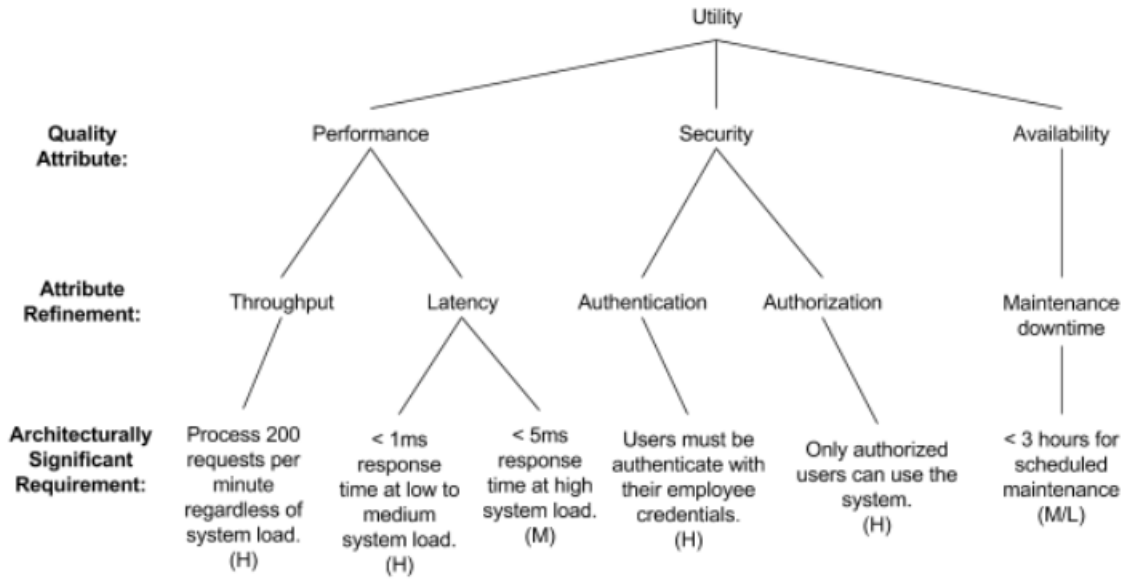
Quality Attribute	Measurement of
Availability	The amount of time the system is operational over a set period of time. A system's availability is measured by uptime, so it can be determined how well the system recovers from issues such as system errors, high loads, or updates. A system should be able to prevent these issues from causing downtime.
Interoperability	The ability of your system to understand communications and share data with external systems. This includes the system's ability to understand interfaces and use them to exchange information under specific conditions with those external systems. This includes communication protocols, data formats, and with whom the system can exchange information. Most modern systems are interoperable and do not exist in isolation.
Security	The system's ability to protect sensitive data from unauthorized and unauthenticated use. This attribute is important as most systems generally contain sensitive information. This information should be protected from those not authorized to see it, but readily available to those who have authorization. Closely associated with this is data integrity. A system should be able to control who can see the data versus who can change it.
Performance	The system's throughput and latency in response to user commands and events. Performance determines how well your system is able to respond to a user command or system event. Throughput is the amount of output produced over a period of time. Latency measures the time it takes to produce an output after receiving an input. This attribute has a major influence on architecture due to its importance to users. It is better to have a lower price per unit of performance. With the advancements in technology, this ratio has been steadily decreasing.
Usability	The ease at which the system's functions can be learned and used by the end users. Usability determines how well your system is able to address the requirements of end users. In order to have high usability, your system needs to be easy and intuitive to learn, minimize user errors, provide feedback to the user to indicate that the system has registered their actions, and make it easy for the user to complete their task.

Scenario Component	Scenario Component Value
Stimulus Source	- End user - Internal subsystems - External subsystems
Stimulus	- Incorrect user input - Internal exceptions - Unrecognized system request - High request volume - Heavy system load
Environment	- Normal operating environment - Starting up - Shutting down - Heavy load operating environment - Recovering from error - Processing request
Artifact	- System servers - System process
Response	- Log exceptions - Notify user - Send error response to external system - Redistribute data processing - Redistribute system requests - Notify user and external systems that the system is shutting down/starting up
Response Measure	- Time to restart (shutdown and startup sequence) - Time to undergo recovery - Time to complete requests when the request volume is high - Time to complete a process under heavy system load - Time to become available after encountering an incorrect input or request

8) ATAM process



The entire ATAM process itself can be broken down into nine steps.



1.2 算法导论

insertion sort C++ 复制代码

```
input:A[n]

for j=2 to A.length:
    key=A[j]
    i=j-1
    while i>0 and A[i]>A[j]
        A[i+1]=A[i]
        i=i-1
    A[i]=key
```

merge sort C++ 复制代码

```
merge(A,p,q,r):
n1=q-p+1;
n2=r-q;
for i=1 to n1:
    L[i]=A[p+i-1]
for j=1 to n2:
    R[j]=A[q+j]
L[n1+1]=∞
R[n2+1]=∞
i=1
j=1
for k=p to r:
    if L[i]<=R[j]:
        A[k]=L[i]
        i=i+1
    else:
        A[k]=R[j]
        j=j+1

merge-sort(A,p,r):
if p<r:
    q=[(r+q)/2](取下界)
    merge-sort(A,p,q)
    merge-sort(A,q+1,r)
    merge(A,p,q,r)
```

2.结果输出

今天主要看了算法导论一书，原计划完成第一部分，发现阅读难度比预期高上不少，只大概阅读了第一部分的一半内容。晚上看完了软件架构课程的视频内容，只能说是囫囵吞枣般的结束了，没有特别多的收获，后续可能看情况做一下里面的作业，毕竟只有这样才能真正学以致用。