Plugin architecture

Definition

- The Plugin architecture pattern consists of two types of architecture components: a core system and plug-in modules. Application logic is divided between independent plug-in modules and the basic core system, providing extensibility, flexibility, and isolation of application features and custom processing logic
- **Vocabulary for components and connectors:**
  - **Core :** The core is the main application, while plugins contain additional features that are added to the main application. Each plugin normally contains one new functionality. The core keeps track of attached plugins through a plugin registry, which includes information like the plug-in name and the protocol for accessing it.
  - **Plugin :** Plugins can be added or removed from the core anytime. One plugin's addition or removal from the core does not affect the other plugins. Plugins are beneficial, providing extensibility, flexibility, and isolation of application features and custom processing logic.

Topological constraints

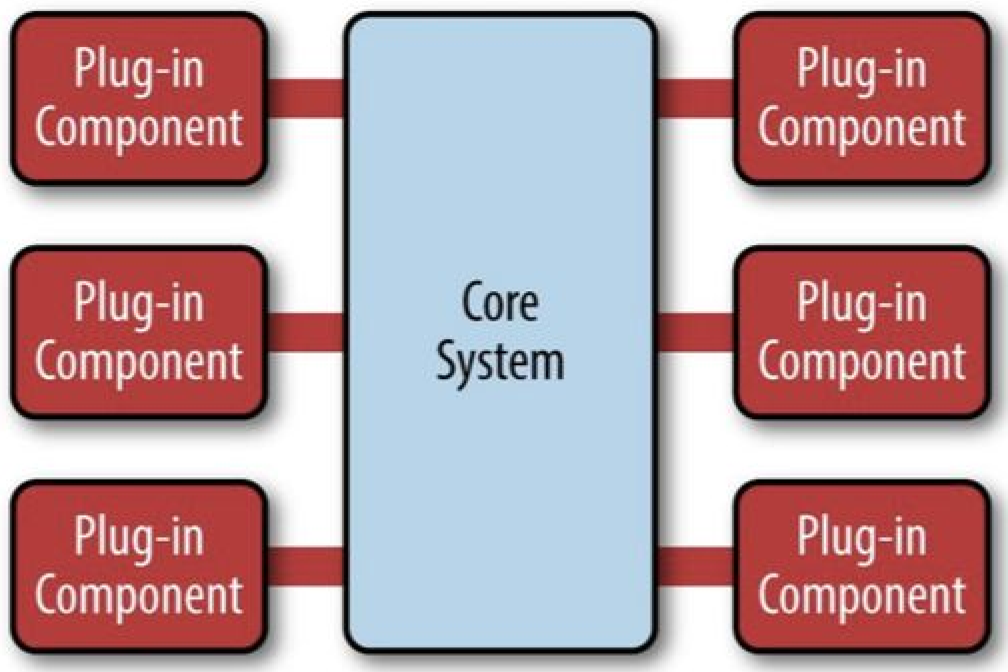

Image from https://www.oreilly.com/ideas/software-architecture-patterns/page/4/microkernel-architecture

● The core system is often defined as the general business logic or bare minimum for the application to function.
● The core needs some way to know what plugins are connected and how to use them (usually done through a plug-in registry).

Applicable problems

- Applications that want to enable easy extension of the main application by third-party developers, or that need to support the addition of new features later (for example, Wordpress has a large number of plugins that support additional features for building websites).

- Applications that deal with many different data types or formats, or need special configurations for different situations (IDE's, for example).

- Applications that want to remain as small as possible while allowing for plug-ins to be added only for the users that need them (for example, Adobe Flash Player for web browsers).

Resilience to change

- Highly customizable plugins - plugins can be easily added or removed also the code of the plugins can be modified without affecting other plugins or the overall functioning of the core.
- Less customizable core - any changes to the core might render all the plugins useless, hence it is important to have backwards compatibility in mind before making major changes to the core.

Negative behaviours

- It can be difficult to decide what belongs in the microkernel and what doesn't.

- The predefined API for connecting the plugins to the core application might not be a good fit for future plug-ins (developers may have to create adapters for third party plug-ins with different connection formats).

- Too many plugins can slow down the application.

- Even if a plugin works perfectly when tested alone, some bugs may appear only with certain combinations of plugins, making it hard to fully test an application.

- Testing can be slow: mock plugin runners are not always possible, so you might just have to run the plugin "for real".

- If you need a plug-in that does two things that work in close tandem, you might have to implement 2 plugins because each plugin should only do one thing. But then, you need to have the two plugins talk to each other, which defeats the purpose of the architecture (each plugin should function separately).

Supported NFPs

- Easily maintained/very modular:
    - If the app requires an update, we can update the specific plugin rather than the core system.The application can continue running during maintenance and only the plugin being worked on would be unavailable.
- Very testable :
    - Plug-ins can be tested in isolation and can be easily mocked by the core system to demonstrate or prototype a particular feature with little or no change to the core system.
- Good performance:
    - Can customize and streamline applications to only include those features you need.
    - Can trim down application server to only those features that are needed, removing expensive non-used features such as remote access, messaging, and caching that consume memory, CPU, and threads that slow down the app server.
- Security:
    - The main application can limit which resources a particular plugin has access to, which protects its data.
- Reusability:
    - In some applications, plugins for one application can be reused for another application.

Inhibited NFPs

- Poor scalability:
  - Can sometimes provide scalability at the plug-in feature level, but overall this pattern is not known for producing highly scalable applications.
  - Implemented as single units and hence not highly scalable, depending on how you implement the plug-in modules.
  - Having too many plugins can slow down the application, and it is difficult to scale the system up to allow for more functionality at this point.
- Resistant to change:
  - Hardware changes or software changes might create problems with some/all plugins, which would require a refactor of all plugin code.
- Difficult to design:
  - Requires thoughtful design and contract governance (changing the API to connect plug-ins to the core application is difficult, so you have to get it right the first time).
  - Contract versioning, internal plug-in registries, plug-in granularity, and the wide choices available for plug-in connectivity make design difficult.
- Difficult to test entire system :
  - While testing individual plugins is easy, it is often unknown how multiple plugins installed together will affect the system, as large numbers of possible combinations may exist, making it impractical to test every one
- Security
  - If a plugin contains a security vulnerability, it may negatively impact the main application.

Comparison with other architectures

- In Mobile code, plug in and black board architectures, the components are not coupled.
- The plugin architecture aims to do different tasks, the blackboard architecture aims to accomplish one single task.

Credit:

Students:

TetraSniffle (Daileen Dolan, Blake Gao, Danny Jin Feng Liu, Nabeel Chaudhry)

Team Flamingo (Stefan Gemnay, Sara Madigan, Christopher Mannes, Caio Souza)

Instructor and TAs:

Mei Nagappan, Achyudh Ram Keshav Ram, Aswin Vayiravan, Wenhan Zhu