Mapping the Space of API Design Decisions
Authors: Jeffery Stylos and Brad Myers

Main points - Erika Harrison:
The main point of the paper is defining the API design space. It can be broken down into the two broad categories of structural design decisions (separating functionality into classes and interfaces), and class design decisions (or the separation into methods and fields). Within these two, there are also decisions relating to the use of design patterns, naming, data types and parameters used, or more broadly, architectural decisions and language level decisions.

How this paper applies to tabletop research and our research in particular - Mahmudul Hasan:

• How and when to use some specific features, such as the use of generic coding, while designing APIs for tabletop SDKs can be further investigated.
• Another avenue of future exploration is how new programming language paradigms may affect the tabletop API design space.
• We can also examine how tabletop API design decisions will be affected by a novel way of exposing modules and services.
• An interesting venture would be to study the centralized versus decentralized natures of tabletop API design to inspect which approach is more appropriate for its further evolution.
• Last but not the least, future research on tabletop API design decisions should focus on the design of generic APIs that would allow to easily port any application from our desktops directly to the tabletops without having to worry about its fixed orientation.

Discussion:
• Defining an API design space:
  o How do we separate Structural design
    Two design patterns explained in the paper
  o Language specific issues of the API
  o Frequency of design decisions
- How should we define class & methods
- Sub-optimal design
- Establishing different types of developers
  - Hobbyist
  - Scientific design
- As new programming languages come, how should the design space change with that
- New ways of exposing modules & services
  - Natural language based approaches
  - How that effects the type of decision made
- Decentralized contribution of framework development
- How the design decisions can interact with new API designs?
- What is the power of an API?
  - Does it allow to use multiple frameworks
- Why some APIs are built on .NET instead of C/C++
- Ability to interact with different API is missing
- The API are often build focusing the major client base
  - This limits certain features like game development
- What are the key features of an API
- API often focuses on comment features
- Consistency is important
- Power Vs. Usability
- Learnability, efficiency, usability, memorability

**Three top things:**
- What makes a good API?
  - learn ability, efficiency, error recoverability, satisfaction
• Who will be the user?
  o What should be the level of abstraction

• Inter-operability

**Other discussions:**
• What is different in table?
  o Orientation
  o Concurrency

• A common mistake during API design?
  o It seems that API developers often don’t know who will be the potential user of the API