Concept/Topic: *Architecturally Evident Coding Styles (AECS)*

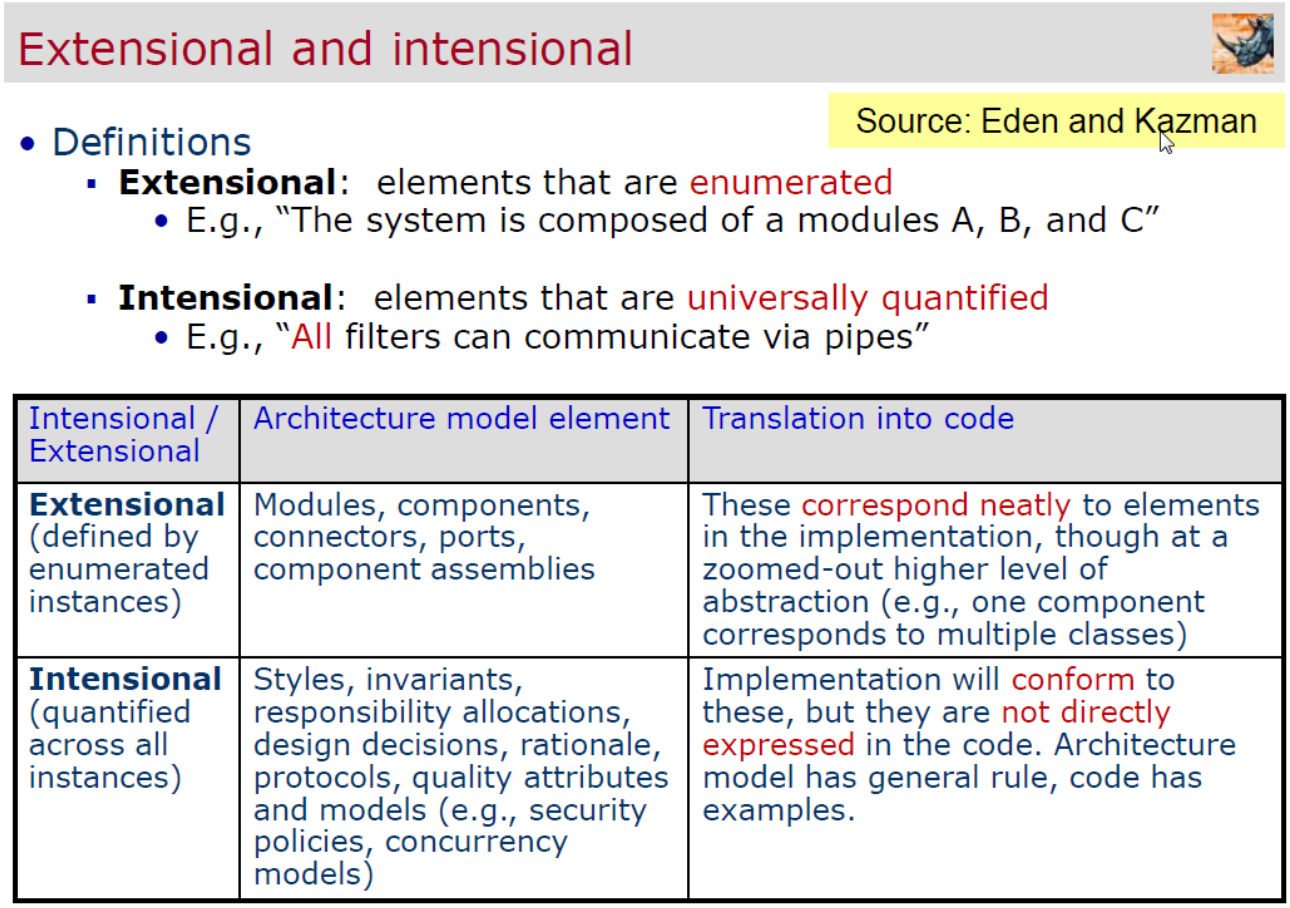
Context

Sometimes, to-be architecture and as-is code get out of synch (hopefully only temporarily). Even worse, in the real world not all developers always respect and implement what has been decided architecturally (for several reasons that do not matter here). Such mismatches between desired and actual architecture are easier to detect if architectural concepts such as components, connectors, and quality properties are visible (or evident) in the code. However, it is hard to express design intent in code.

Concepts

AECS Origins and First Advice

G. Fairbanks advocated the term *Architecturally Evident Coding Styles (AECS)* in his conference tutorials, drawing on Chapter 10 on his book "Just Enough Software Architecture" (Fairbanks (2010)). See for instance his tutorial held at SATURN 2013[1] and presentation recorded at Agile Roots 2010[2]. He distinguishes between extensional and intensional architectural model elements:



[1] https://resources.sei.cmu.edu/asset_files/presentation/2013_017_001_48651.pdf
[2] http://www.georgefairbanks.com/blog/architecture-in-code-agileroots-2010/

His key ideas are organized in 6 slices in the conference tutorial and 10 intents in the book. They can be paraphrased and annotated like this:

- Components and patterns should be visible in code (e.g., abstract classes be defined for components/patterns).
- Startup code should be centralized (and explicitly named) so that it can be located easily and tested separately.
- Quality-related properties should be marked as such, for instance with annotations or predefined, parsable comments.

An AECS can be promoted by applying pattern languages such as DDD – if this is done visibly and explicitly. Hence, all DDD patterns per se qualify as elements of an AECS. Some of them aim at making design visible in the code explicitly, for example Intention-Revealing Interfaces and other "supple design" patterns.

What is in a Name?

Finding good names is hard as M. Fowler put it here[3]. Your lecturer's advice is to use a grammar construct that unveils type of artifact: verbs for use cases and methods, nouns for components and data members, etc.

Another recommendation is to be *intention revealing* and use concrete concepts from the domain in most (if not all) names, including those of parameters and variables. Such domain name element can be accompanied by a pattern name (or pattern component name) to indicate architectural role and responsibilities. The names should not get too long, two to three words per name usually are sufficient. Abbreviations should generally be avoided, but are tolerable here, for instance for the pattern name suffix (example: `InsurancePolicyDTO`).

Other things you can do:

- Make external interfaces very visible as such, including pre- and postconditions.
- Add links to quality attribute scenarios or quality stories, sometimes even sample data to comments.
- Indicate memory requirements (state management is key concerns).
- Practice the Ubiquitous Language everywhere (as the name suggests).
- Put methods like `checkPrecondition()` or `authorizeRequest()` in the same place – and use such expressive names rather than generic ones such as `check()` or semi-cryptic ones such as `evalRBACPol()`.

Additional Resources

S. Brown picked up the notion of AECS in his own work, see for instance this online article[4].

Note that not all such advice always applies and you do not have to agree with everything said and written. Like in most fields, there are no "silver bullets" in software engineering and architecture, suited solutions and conventions differ by project context and community. For instance, S. Brown's advice not to use layers as primary package names can be challenged, there are good reasons to use them exactly for that purpose (see sample application used in exercises 5 and 6).

References

Fairbanks, G. 2010. *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall & Brainerd.

[3] https://martinfowler.com/bliki/TwoHardThings.html
[4] http://www.codingthearchitecture.com/2014/06/01/an_architecturally_evident_coding_style.html