Heuristics for systems-level architecting

CS 428

Winter 2021

Bruce F. Webster

- "heuristic" = from the Greek heuriko, "to discover" (hence: "Eureka! I have found it!") – something to help guide you to a solution
  - In AI, a heuristic is typically a goal-seeking algorithm or principle
  - Here: principles, laws, rules, maxims or rules of thumb to remember in creating and building a system of any kind (not just information technology)
- D = descriptive, that is, telling you what can happen
- P = prescriptive, this is, telling you what you should be doing to help avoid the problems
- Source: **The Art of Systems Architecture** (3rd ed.) by Mark W. Maier and Eberhardt Rechtin (CRC Press, 2009)
- These slides only cover a subset from the document

What are these?

- Performance, cost, and schedule cannot be specified independently; at least one of the three must depend upon the other
- Efficiency is inversely proportional to universality [the reuse problem]
- The most reliable part on an airplane is the one that isn't there
- If the politics don't fly, the hardware never will
- In introducing technological and social change, how you do it is often more important than what you do
- There is no such thing as a purely technical problem

Multitask Heuristics

- The beginning is the most important part of the work. (Plato)
- In architecting a new program, all the serious mistakes are made the first day. (Spinrad)
- Success is defined by the beholder, not by the architect.
- Risk is also defined by the beholder, not by the architect.
- Four questions need to be answered as a self-consistent set if a system is to succeed economically, namely: Who benefits? Who pays? Who provides? and, as appropriate, Who loses?
- Don’t assume that they original statement of the problem is necessarily the best, or even the right, one.
- Do the hard parts first.

Scoping and Planning

Machiavelli, “The Prince”

- “It ought to be remembered that there is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things. Because the innovator has for enemies all those who have done well under the old conditions, and lukewarm defenders in those who may do well under the new. This coolness arises partly from fear of the opponents, who have the laws on their side, and partly from the incredulity of men, who do not readily believe in new things until they have had a long experience of them.”
- — **Niccolò Machiavelli,** The Prince

- If you can’t analyze it, don’t build it.
- Modeling is a craft and at times an art.
- A model is not reality.
- The map is not the territory.
- Any war game, systems analysis, or study whose results can’t be easily explained on the back of an envelope is not just worthless; it is probably dangerous.
- If you can’t explain it in five minutes, either you don’t understand it or it doesn’t work.
- A good solution somehow looks nice.

Modeling

- In any resource-limited situation, the true value of a given service or product is determined by what one is willing to give up to obtain it.
- When choices must be made with unavoidably inadequate information, choose the best available and then watch to see whether future solutions appear faster than future problems. ... If not, go back and choose again.
- The choice between architectures may well depend upon what set of drawbacks the client can handle best.
- Every once in a while you have to go back and see what the real world is telling you.

Prioritizing (Trades, options, choices)

- Group elements that are strongly related to each other; separate elements that are unrelated.
- Subsystem interfaces should be drawn so that each subsystem can be implemented independent of the specific implementation of the subsystems to which it interfaces.
- Choose a configuration with minimal communications between subsystems.
- Choose the elements so that they are as independent as possible; that is, elements with low external complexity (low coupling) and high internal complexity (high cohesion).
- The optimum number of architectural elements is the amount that leads to distinct action, not general planning.
- System structure should resemble functional structure.

Aggregating (“chunking”)

- Design the structure with good “bones”
- It is inadequate to architect up to the boundaries or interfaces of a system; one must architect across them
- Be prepared for reality to offer a few interfaces of its own
- Do not slice through regions where high rates of information exchange are required [back to high cohesion / low coupling]

Partitioning (Decompositioning)

- Relationships among the elements are what give systems their added value
- The greatest leverage – and the greatest dangers – of systems architecting are at the interfaces
- Be sure to ask the question, “What is the worst thing that other elements could do to you across the interface?”
- The product [system] and the process [human workflow] must match. Or, by extension, a system architecture cannot be considered complete lacking a suitable match with the process architecture

Integrating

- As time to delivery decreases, the threat to functionality increases.
- The number of [unknown] defects remaining in a system after a given level of test or review is proportional to the number found during that test or review [bugs closed/new bugs found metric]
- The test setup for a system is itself a system.
- The least expensive and most effective place to find and fix a problem is at its source.
- Quality can't be tested in; it has to be built in.
- High-quality, reliable systems are produced by high-quality architecting, engineering, design and manufacture, not by inspection, test, and rework.

Certifying (System Integrity, Quality, and Vision)

- System quality is defined in terms of customer satisfaction, not requirements satisfaction.
- If you think your design is perfect, it's only because you haven't shown it to someone else.
- "Proven" and "state of the art" are mutually exclusive qualities.
- The reverse of diagnostic techniques are good architectures.
- Before it's tried, it's opinion. After it's tried, it's obvious.
- The first quick look analyses are often wrong.
- Chances of recovery from a single flaw/failure are pretty good. Recovery from two or more independent failures is unlikely in real time and uncertain in any case.

Assessing Performance, Cost, Schedule, and Risk

Re-Architecting, Evolving, Modifying, and Adapting

- If you don't understand the existing system, you can't be sure you're rearchitecting a better one.
- When implementing a change, keep some elements constant to provide an anchor point for people to cling to.
- Before the change, it is your opinion. After the change, it is your problem.
- Unless constrained, rearchitecting has a natural tendency to proceed unchecked until it results in a substantial transformation of the system.

To do for the next two weeks

- **CRITICAL**: if you don't have a GitHub account already, got to GitHub.com and register for one. Then DM me or Zack Mortensen your GitHub user name
- Log into the class Github (https://github.com/cs428TAs/f2020/wiki)
  - Sometime this week: create a proposed project and/or endorse an existing one
  - By next Monday (01/18): vote on at least three (3) projects
  - Through next week, we will try to organize projects and teams
  - We will finalize projects and teams during class in two weeks (01/25)
- Join cs428-w21.slack.com if you haven't already
- Readings for next two weeks (by 01/25):
  - "The Five Orders of Ignorance", Philip Armour (PDF on the website)
  - "Heuristics for System Level Architecting", Meier & Rechtin (PDF here)
  - *Mythical Man-Month*, chapters 1, 2, 4, 5, 7, 11, 14