Chris Cargile	CSCI656	April 21, 2014	Ch.12 HW Questions

This homework assignment required 3.5 h to complete. Reading required 2 h; answering questions from the assigned section required 1.5 h.

12.7.1) Define the following NFP's

A. **Efficiency:** software systems' ability to meet performance requirements and at the same time using minimal amount of resources in computing environment, economizing system resource usage.
B. **Complexity:** the degree of difficulty regarding understanding and verifying the design or implementation of a software system or one of its components, according to structure, interdependencies and lines of code size.
C. **Adaptability:** software's ability to satisfy new requirements and adjust to new operating conditions during its lifetime.
D. **Scalability:** capability for software systems to be adapted to meet revised requirements of size and scope.

12.7.3) How is each of the above NFP's manifested in software architectures?

Each of the NFP's above manifests itself in software architectures in terms of connectors, components and architectures, as a whole. Efficiency manifests itself in software architectures by ways of components optimally when they are kept small, their interfaces simple and compact and their interfaces are multi-functional. Optimizing connector designs for efficiency involves bearing in mind system usage economy by using multicast connectors sparingly to decrease burden on system overhead due to wasted resource usage and using asynchronous interactions when possible, location(aka distribution) transparency wisely.

Complexity is manifested in software architectures optimally when software systems are design in consideration of elements such as size, simplicity regarding structural aspects and minimal interdependencies. Keeping concerns divided into separate components, functionality – not interactions – in components and keeping them cogent, as much as possible. Insulating components, representing connectors as first-class citizens and dividing concerns for interaction, again, with regard to connectors helps achieve this. Using a system not tightly bound in which systems are systematically decomposed will help in achieving such a goal also.

Adaptability is achieved best when the system can evolve to achieving new requirements during its lifetime. Adaptability requirements are achieved when components and connectors exhibit the same characteristics of being separate but not dependent on one another where possible. Data being separated from meta-data while keeping connectors flexible and composable will aid in this goal, also.

Scalability is achieved best when systems are heterogeneous and portable. When components are given singular, clear purposes with understandable interfaces, data sources are distributed and replicated as much as possible, the system may best achieve this end. Connectors should be explicit, clearly defined and functionality isolated and separated by concern to reduce bottlenecks and optimize for parallel system processing.

Chris Cargile	CSCI656	April 21, 2014	Ch.12 HW Questions

12.8.2) Select any two of the architectures from Ch.4. For each NFP discussed in this chapter, analyze what can and cannot be determined about the selected architectures.

Architecture Types (2):	**Pipe-and-Filter Style** (pg.111)	**Interpreter Style** (pg.117) Note: the interpreter style is likely to employ direct procedure calls; thus connectors probably are not relevant for this style
What can be determined considering the NFPs		
• **Efficiency**	Software components expose simple, compact interfaces.	Component size, simplicity, nor compactness is not necessarily foreseeable.
• **Complexity**	Components are kept separate to address distinct concerns; data is usually routed through connectors, not components themselves.	Components are kept separate to address distinct concerns; connectors probably are not used;
• **Scalability & Heterogeneity**	Components have a simple, clear purpose;	Components have a simple, clear purpose;
• **Adaptability**	Components are not likely to be interdependent or their purposes vague so they will be adaptable; connectors too have an explicit, simple role so they are. If the OS/implementation were to permit variable-width (aka 'diameter') pipes, connector would be more flexible.	The interpreter (like other rule-based styles) may be the least likely of styles to exhibit this NFP. It is highly rigid and inflexible per the guidelines (or 'tactics') on pg.475.
• **Dependability (:Survivability, :Safety)**	This is likely the hardest among NFPs to be quantified for the above style. Faults may be as likely as lower-level operations in the OS to fail or the style may be enhanced for robustness; interdependencies should be	Attacks, failures, and accidents should be as avoidable as components are kept 'fireproof,' themselves. Component interaction guarantees would be a feasible means for ensuring commands from that (component) source are

	minimal, reflection, exception handling mechanisms achievable, and key state invariants specifiable.	ultimately forwarded to the interpreted state (-component); this would require the command-issuer resubmit, for interpretation, any commands not recognized/processed by the interpreter.

Can you recognize any of the NFP design guidelines in either of the two architectures?

Guidelines from the efficiency, complexity, scalability & heterogeneity and adaptability NFP's seem fitting to the Pipe-and-Filter style of architecture.

Guidelines for complexity, scalability & heterogeneity NFP's seem relevant to the Interpreter architectural style, also.

Recurring principles discussed regarding keeping components and connectors as simple and their purposes as straightforward as possible seem applicable to each.

What is your assessment of the architectures' respective support for the NFP under consideration?

Considering my responses above, I'd say each conforms generally well in several categories but clearly they have distinctive styles of programs for which they each are better suited. Neither style seems necessarily likely to be inherently dependable in the midst of attacks or surprise compromises to the systems on which they are intended to run.

The interpreter style is considered less adaptable, while the P-and-F style is less scalable than other styles.