Questions to Ponder in Advance of the Workshop

To help stimulate discussion during the workshop, we would like to ask *all* participants to consider the following questions prior to the workshop. The questions are in **bold**, surrounded by some explanation and examples.

Please be sure to have thought about these issues and be prepared to discuss them. Don't take the questions as being boundaries, however—if other issues strike you as important, please bring them as well!

Concerns and Methodologies

- Many different kinds of *concerns* are of importance in software systems. Some of them derive from the domain of the system, others from the software development process itself. To give but a few examples: key *data* and *functions* in the domain model; *features* in the system; non-functional *aspects*, such as distribution and error handling; *use cases*, describing particular scenarios of operation; *configurations*, *variations* and *versions*. Developers often encapsulate, or at least try to encapsulate, important concerns in whatever kinds of "modules" are available to them in the language(s) they are using. Examples include: functions, classes, interfaces, packages, UML or Catalysis diagrams, design patterns, frameworks, aspects, propagation patterns, subjects.

  What kinds of concerns do you deal with in your research or development activities? What is at least one example of each kind of concern?

- **For each kind of concern:**

  - **How do you go about identifying concerns of this kind in practice?** For example, in the Booch object-oriented design methodology, Booch recommends underlining all of the nouns and verbs in a requirements specification to begin the design process; the nouns become candidate classes (concerns) in the design, and the verbs become candidate methods of those classes. Features and use cases are also often identified from end-user requirements specifications; aspects are identified by noting capabilities which "cut across" multiple classes; one way to identify subjects is by determining when developers have different perspectives on a piece of software, and encapsulating each perspective in a subject; product lines are generally identified by discerning needed points of variation in a particular system, such as operating system or hardware dependencies.

  - **Can you represent and manipulate this kind of concern as a first-class entity?** For example, Java can represent class concerns readily, but not features, aspects or use cases.

  - **When do you identify and use this kind of concern?** For example, does it occur during requirements-gathering, design, initial coding, or software testing? During the course of evolution? To facilitate certain kinds of changes? Is it one that you use repeatedly throughout the software lifecycle, like classes? Is it one that you use for briefer periods of time, to accomplish certain tasks? An example of a transitory kind of concern occurs, for example, if a developer's task is "rename all classes that contain 'Impl' in their name so that they contain 'Spec' instead". In this case, the developer has two primary concerns: the set of all classes that contain 'Impl' in their name, and the set of code locations that make reference to these classes. (This is also a kind of concern that cannot be represented as a first-class entity in Java.)

Mechanisms and Tradeoffs

- Many kinds of mechanisms exist for achieving separation of concerns. For example:
  - Linguistic mechanisms: These include the definition of new languages or formalisms, and the use of features built into one or more formalisms, to represent concerns. Some examples of existing linguistic mechanisms that may be used to represent concerns are modules (e.g., classes, packages), annotations, comments, information transparency, reflection, aspects, macros (as in C++), etc.
  - Extra-linguistic mechanisms: These entail defining special mechanisms *outside* particular programming languages to permit the representation of concerns. These include propagation patterns, composition filters, subject-oriented programming, design patterns, version control and configuration management systems, separate documentation to describe concerns, etc.

  What kinds of mechanisms do you use to achieve separation of concerns in your work?

- **If you use more than one kind of mechanism, *when* do you use each of the mechanisms?** For example, do you use different mechanisms at different stages of the software lifecycle (e.g., some during initial development, and others during evolution)? Do you use different mechanisms to separate different kinds of concerns?
- Once concerns are separated, it is often useful to be able to *integrate* them. For example, aspect-oriented programming provides a *weaver* tool to add aspects to base classes; subject-oriented programming provides a *compositor* tool to integrate subjects together; Java classes are integrated together into components or running systems partly at compilation-time and partly at load-time, based on "imports" clauses and references from one class' members to those of another class; RCS provides the ability to merge separate branches.

  For each separation of concerns mechanism you have identified:

  - **How does it permit concern integration? What, specifically, can be integrated?**
  - **What are the "join points," and what integration properties do they imply?** When different concerns are integrated, there are points at which they come together, or interact, which are called *join points*. For example, composition filters operate by filtering method calls, so the join points are methods. Some weaving approaches combine entire methods as atomic units, whereas others weave code statements within methods. The nature of the join points affects a number of properties of an integration mechanism, such as flexibility, the ability to understand the integrated whole in terms of the components, reusability of components, and the nature and complexity of needed tool support.
  - **How much do concerns know about one another?** That is, how tightly are they bound to, and coupled with, one another? For example, Java classes know the name and package of every class to which they refer, and they know which members they can access in those classes, but they do not know, until runtime, which specific implementation class will actually be used (because of possible changes to the "class path"); aspects know about, and refer directly to, the base classes they augment, but they do not know about each other, and the base classes do not know about them; subjects know nothing about one another.
  - **When can concerns be integrated?** For example, is integration done statically (at compile, link, or analysis time), as in Catalysis, propagation patterns and subject-oriented programming? Dynamically (at runtime), as in composition filters or aspect

instances within aspect-oriented programming? If dynamically, can it be done once or repeatedly?

- **For each of the mechanisms you use, what do you see as its primary strengths with respect to the tasks to which you apply it? What are its primary weaknesses?** For example, are there particular kinds of concerns it lets you separate well, and others that it does not allow you to separate well? Are the concerns you separate first-class entities? Are they reusable? Are there circumstances under which you would not, or have not been able to, use this mechanism? Have you ever gotten frustrated with the mechanisms you had available to you, and wished that they had some additional feature or capability that they are lacking (or, conversely, that they did not have some feature that you find hinders or constrains you)? Please think about the weaknesses, as well as the strengths, of the mechanisms you use or defined as part of your research or development activities; that helps to promote understanding of the boundaries and tradeoffs.