Small group research project (50%)

- Groups of 2-4 (formed on your own)
- One-page proposal for a small-scale software engineering research project due January 31
- All groups make presentations to the rest of the class on March 13
- Write-ups (like a workshop or conference paper) due on March 17

But…

- …we haven't read much software engineering research yet
- …we haven't done any software engineering research before
- …we haven't thought about how to evaluate software engineering research (well, actually you have – and I want your thoughts on this to inform your projects)
- …<insert more>

Today

- Overview of research problems that (Marius and) I find interesting and tractable
- Some general comments first
- Marius on software configurations
- Me on possible projects in other areas
- You on anything I missed that you're interested in – this list is not intended to be complete in any sense
- Collectively there will be only 3 to 6 projects total
- Discuss ideas with Marius and me; we can suggest ideas, refinements, literature, etc.

Software engineering research

- Boehm: more software systems fail because they don't meet user needs than because they aren't implemented properly.
- Notkin: more software engineering research is uninteresting because the problem addressed is uninteresting rather than because the solution doesn't address the problem
- Software engineering problems of interest should usually (in my opinion), and very briefly…
  - Populate the world with a potentially powerful new approach (e.g., Parnas information hiding, Weiser program slicing, software model checking, Liblit et al. statistical defect location, Ernst dynamic invariants)
  - Make progress on an approach that is increasingly likely to have value (many…)
  - Carefully consider conventional wisdom (e.g., Knight and Leveson, Kim et al., Votta inspections)
- Your projects are highly time-limited; keeping this kind of idea in mind is good, though

Software engineering research students…

- …always hear me say, “What is it about software engineering that sucks, but shouldn’t?”
  - Often (but not always) focuses on tedious, error-prone activities
  - Griswold: we don’t restructure programs as often as we should because it’s error-prone?
  - Murphy: software architecture is well and good, but what happens when the code base drifts away from the architecture over time?
  - Ernst: invariants are good, but we don’t see as many as we’d like – is there a way to get more?
  - Nita: the only thing more prevalent that software configurations is the *ad hoc* nature of thinking about and manipulating them – can we do better?

Marius: configurations

Mining software repositories

- "Research is now proceeding to uncover the ways in which mining [software] repositories can help to understand software development, to support predictions about software development, and to plan various aspects of software projects." [MSR 2007 web page]

- Repositories are broadly defined to include code, defect databases, version control information, programmer communications, etc.

- Enablers include the Internet, open source, more repositories, more complex repositories, fast/cheap processors, big/cheap memories, big/cheap disks, data mining/machine learning results, new analyses, …

Great area…

- So many empirical questions can be studied
  - In essence, this is the science of studying software *in vivo*
- Beware: there is so much data that finding correlations is easy, but they often do not suggest causation (cf., *A Mathematician Reads the Newspaper*, Paulos)
- Beware: if one learns something this way, understand whether the intent is "general knowledge of software" or, more preferably, something that could be "actionable"

Examples

- Any evidence of differing approaches to refactoring in agile vs. non-agile software systems?
- Are the claims of 5:1 variations in (many dimensions of) programmer productivity justified by existing repositories?
- Any evidence, from repositories, of the benefits of using Mylyn?
- Can one characterize successful open source systems from unsuccessful ones in a precise way based on their repositories?
- …

Examples: other areas

- A key issue in design is anticipating future changes – but we do this (at best) in an informal and *ad hoc* way: is there a more disciplined way to think about this?
- Under what contexts is the "small scope hypothesis" more/less likely to hold?
- Apply
  - Alloy to a system of interest (with a clear focus)
  - Daikon to a system of interest (with a clear focus)
  - …
- Define a "physical" (say, keystrokes needed) model of the cost of developing and maintaining software

More examples

- Combine static and dynamic extractors to construct a “better” call graph
  - Or any such combination of static and dynamic tools with similar intent
- Can a concordance of (natural language) comments represent any value?
- Debugging when the object code is structured very differently from the source code is hard – consider this in the context of optimizing compilers, aspect-oriented weavers, or such
- Develop/modify a software engineering tool that runs “side-by-side” with its subject program on a multi-core machine

More examples

- Consider, as an optimization problem, how to best spend a fixed amount of resources split across (say) unit testing and fuzz (random) testing
- Belady and Lehman have some “laws” of software evolution, originally based on data from systems in the 1970’s and 1980’s – reassess these laws in the modern software context
- Consider Dwyer’s model of behavior vs. requirements (which I showed in class) – show how to build confidence in a significant rectangle (a set of behaviors **and** a set of requirements)

Your ideas?

Clarify and scope

- Many of the suggestions are fuzzy and ill-scoped
- After picking a topic, that's your first job – what's the definition, what's the scope, and what's the plan?
- We're here to advise…