CSE P503: Principles of Software Engineering

David Notkin
Autumn 2007

Looking back, looking forward

Prediction is very difficult, especially of the future.
--Niels Bohr

The best way to predict the future is to invent it.
--Alan Kay

Software is the soul to the lifeless body of the hardware. --Anonymous

Tonight

- Brief coverage of several topics
  - Software product lines
  - Globalization of software engineering
  - Economics-driven software engineering
  - Intentional Programming
  - Trust
- What I learned from p503 (after all, I told you it was all about me)
- Final questions, discussion points, etc.
- Course evaluations

Software product lines: wikipedia

- … refers to engineering methods, tools and techniques for creating a collection of similar software systems from a shared set of software assets using a common means of production. …
- The characteristic that distinguishes software product lines from previous efforts is predictive versus opportunistic software reuse. Rather than put general software components into a library in hopes that opportunities for reuse will arise, software product lines only call for software artifacts to be created when reuse is predicted in one or more products in a well defined product line.

It takes a village…

- *Software Reuse: Architecture, Process and Organization for Business Success*
  I. Jacobson, M. Griss, P. Jonsson
- The basic idea is that systematic reuse – almost always in the style of software product lines – demands that the entire organizational structure, software process, and software architecture be focused on reuse
- "Using software product line techniques, companies such as Nokia, HP, LSI Logic, Philips, and Cummins have improved time-to-market, engineering costs, portfolio size and defect rates by factors of 3 to 50." `softwareproductlines.com`
  - Ad hoc, serendipitous reuse may happen, but it won't provide significant value

Software product line process

- Product Management
  - This phase defines a scope, which tells what should and should not be part of the family
- Domain Engineering
  - This phase establishes a reusable platform by defining a set of common and variable requirements for all elements in the family
- Product Engineering
  - A given product is derived from the reusable platform, with the shared common requirements plus instantiated specific requirements

Globalization of software engineering

- Why globalize the engineering of software systems?
  - Economics – less expensive labor, better access to labor, faster product development
  - Better access to global markets
  - Team diversity can lead to better products
- What problems arise due to this globalization?

Workshop on Accountability and Traceability in Global Software Engineering

- "The reality of modern software development involves outsourcing, offshore development, and integration [of empirical data] from multiple sources. Unfortunately, corporate concerns often limit the availability and use of empirical data outside the project or company, which can make providing accountability and traceability for such projects difficult."

Software Tag: NAIST and Osaka University

- Empirical Software Engineering Data for Traceability and Transparency of Software Projects
- Basically the idea is to associate software metrics with software elements, allowing customers to see the metrics information and drawing their own judgments
- There are *huge* questions – technical, legal, etc. – but nonetheless it's a reasonably refreshing idea

A few observations

- Incredible investment and interest in China in software process, software process certification, software process improvement
- Globalization faces many of the obvious complications – time zone differences, language differences, increased demand on precise interface definition, etc.
- Some have slightly surprising consequences in some situations
  - Time zone differences not only time shifts but in many organizations increases the length of the work week due to the need for handoffs
  - Language differences may be a plus if two teams communicate in neither of the native languages – there are fewer subtle expectations about non-native languages

“Culture Can Confound Global Software Metrics” [Notkin]

- Even when the corporations and people are willing and able to share empirical data, there are numerous cultural complications that must not be ignored
- There is evidence of this point from fields such as medicine and labor statistics

Medicine and Culture: Lynn Payer

- Medical information coding and interpretation varies widely across four similar Western countries: USA, UK, France, Germany
- For example
  - A statistically significant difference in how doctors in these countries report the cause of death for some circumstances
  - A German patient with low blood pressure may be given prescription medicines, but in the U.S. the same patient would likely get a discount on life insurance

Why should we care?

- Payer argues that these differences are not a question of “good medicine” vs. “bad medicine” but are due largely to differing cultures
- But “[t]he widespread ignorance that medicine in highly developed countries can be so different has a number of serous implications. First, all sorts of unjustified conclusions are currently being drawn from international statistics…”

International labor statistics

- "When making international comparisons of average weekly hours of work, we suggest you carefully look at the concept used and measured by countries. Some countries provide data for hours actually worked, which exclude hours paid for but not worked, for annual leave, holidays, days off, personal leave, etc.; while others calculate hours paid for only … On the other hand, average weekly hours reflect effects of numerous factors such as paid or unpaid absenteeism, labour turnover, part-time work, strikes, and fluctuations in work schedules for economic reasons..." [International Labour Organization website]

But what about software?

- If there is a statistically significant difference in the way software engineers from different cultures, but on the same project, report the cause of a failure in some circumstances, this could affect reliability models
- If engineers in one culture judge a software behavior as an error, while others judge it as a feature, this could affect the data underlying key metrics
- If measuring something as “simple” as developer-hours is itself complex and culturally-based, then metrics derived based on it may be difficult to compare with the desired effect

Add on interpretation and intent …

- Metrics can be intentionally or unintentionally interpreted differently, for many reasons – and culture adds uncertainty
  - Protecting the boss … or trying to take the boss' job?
  - What is the reward structure? Does it vary across the participants in a global software engineering project?
  - …
- Intentions vary widely across cultures (see, for example, *More Like Us* by James Fallows), which may subtlety affect the use of data

Global teams are vital

- The benefits of global software engineering go *far* beyond direct economic benefits
- Bill Wulf argues that a diverse work force is *absolutely essential* to ensure that the best designs are found in all engineering disciplines
- Global teams that engineer software products will, in the long run, produce better designs than a monoculture
- Addressing the confounding of metrics due to varied cultures is one key step towards an effective, diverse, global work force for software engineering

Economics-driven software engineering (Sullivan)

- Modular software design architectures create value in at least two ways
  - they support the delivery of properties for which people are willing to pay
  - they create opportunities to make follow-on investments to adapt a design to deliver even more valuable properties
- Figuring out how to put a value on these opportunities is a real challenge, especially when the potential benefits are uncertain
  - Real options have the potential to address such questions
  - However, real options techniques (e.g., using Black-Scholes or binomial options pricing techniques) cannot be used directly because they make deep assumptions about the nature and measurability of the underlying uncertainties – and these assumptions are generally invalid in the setting of design of unprecedented systems
- So, how can one develop and validate options-like models for valuing investments in modular design architectures?

Core background

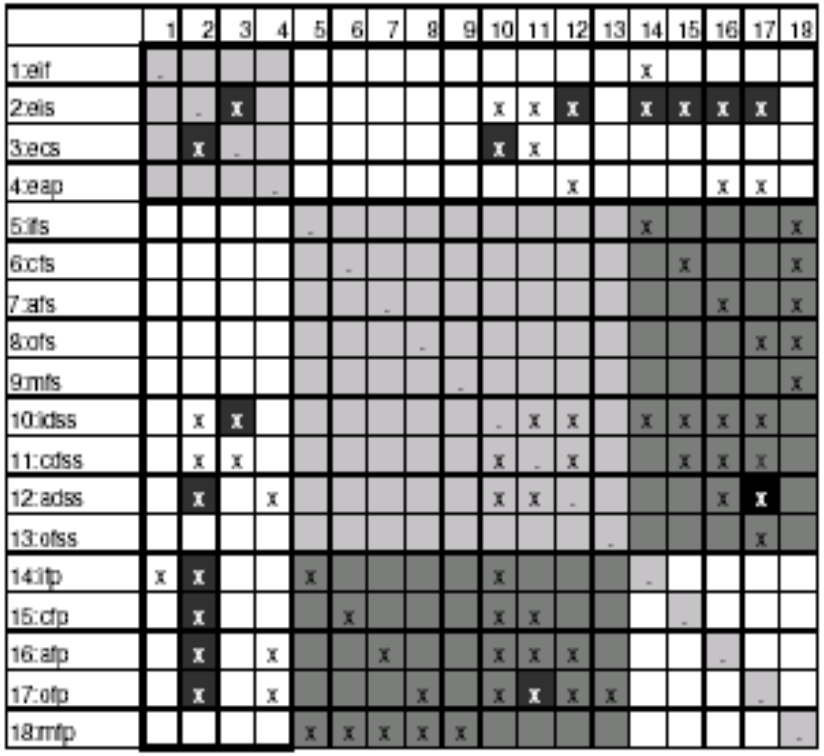
- DESIGN RULES: The Power of Modularity -- Carliss Y. Baldwin & Kim B. Clark
  - "…IBM gave us the prototype of modularized design. This book describes the history of that development, and then extracts and generalizes the principles. The IBM System/370 was the first modularized mainframe. Hardware was done first -- object-oriented software came later. Baldwin & Clark claim that their principles apply equally to social and legal institutions as well as technologies. Modularization could improve the design of almost everything." –W. Sheridan
- One key notion is the *design structure matrix* (DSM)

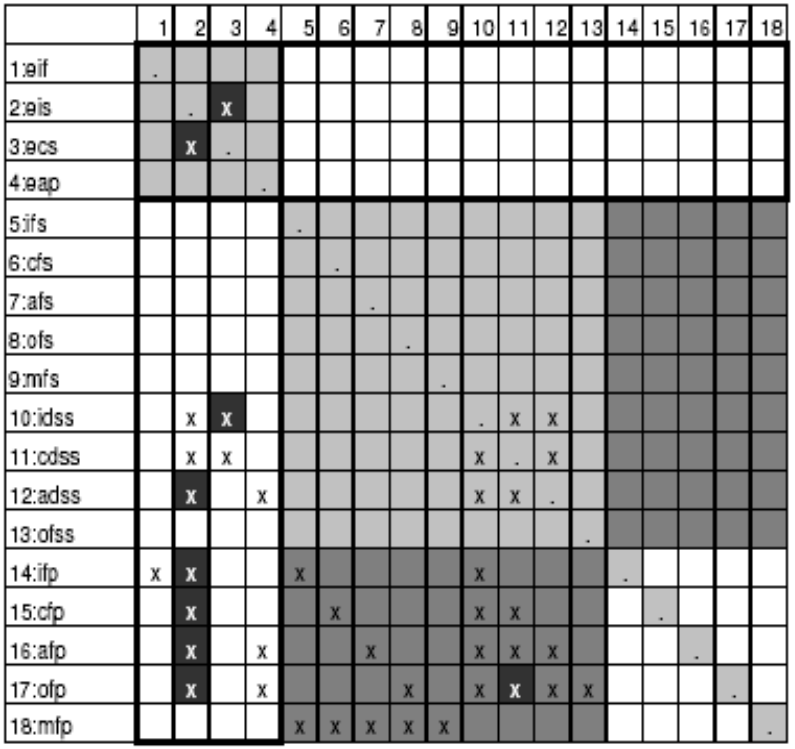
DSM (for software) – Sullivan, Cai, Hallen, Griswold

- A DSM represents dependencies among design parameters, the range of choices that can be made about an aspect of a design
- Typical software design parameters include data structures, algorithms, procedure and type signatures, graphical interface look-and-feel, interoperability, and performance characteristics
- The rows and columns of a DSM are labeled by the design parameters and dependences between two parameters are marked
- Marking row B, column A means that an effective choice for B depends on the choice for A
- Parameters that require mutual consistency—algorithm and data structures often go hand-in-hand, for example—result in symmetric markings
- Choosing the design parameters to model and the values they finally take on is the task of the designer

Yuanfang Cai: slide 1

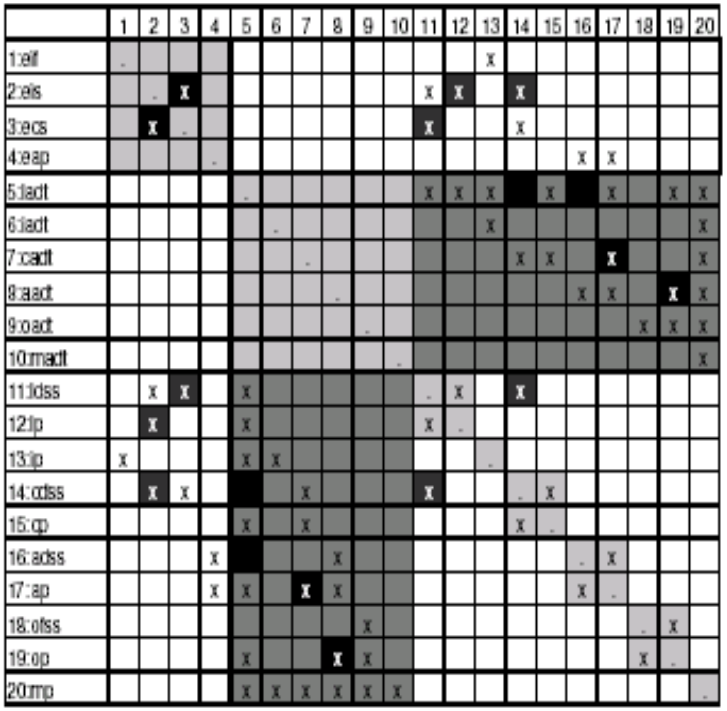
- **KWIC Sequential Design**

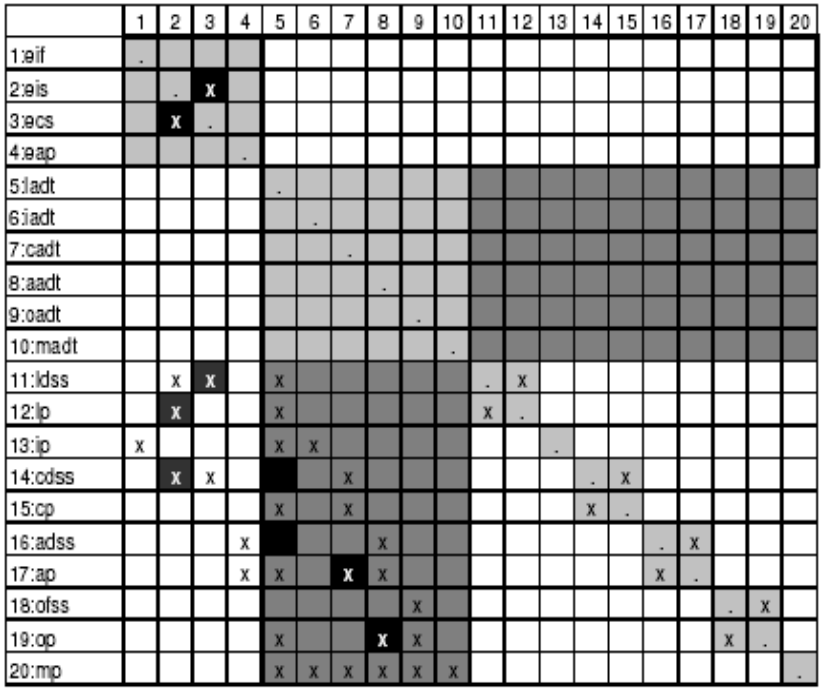




Slide 2

- **KWIC Information Hiding Design**





Design Rules

Just the slightest flavor of EDSER

- Sullivan, Boehm, Shaw, Cai, and others for more information

Intentional Programming (Simonyi)

"WYSIWYG editors simplified document creation by separating the document contents from the looks and by automating the reapplication of the looks to changing contents. In the same way Intentional Software simplifies software creation by separating the software contents in terms of their various domains from the implementation of the software and by enabling automatic regeneration of the software as the contents change. This way, domain experts can work in parallel with programmers in their respective areas of expertise; and the repeated intermingling can be automated."

–OOPSLA 2006 abstract

Technology Review (Scott Rosenberg, Jan/Feb 2007)

- If Simonyi has his way, programmers will stop trying to manage their clients' needs. Instead, for every problem they're asked to tackle--whether inventory tracking or missile guidance--they will create generic tools that the computer users themselves can modify to guide the software's future evolution.
- Suppose an international bank wanted to develop a new system for managing transactions in multiple currencies. First, the bank's own domain experts would define the system's functionality, using their customary terms and symbols and identifying the most important variables ("time" or "value" or "size of transaction") and the most common procedures ("convert holdings from one currency to another" or "purchase hedge against falling value"). Then the programmers would take that information and build a "domain specific" program generator that embodies that information. A separate software tool would allow the domain experts to experiment with different sets of data and ways to view that data as easily as business-people today rearrange their spreadsheets.

Continued….

- Simonyi argues that his approach solves several of software engineering's most persistent problems. Programmers today, he often says, are "unwitting cryptographers": they gather requirements and knowledge from their clients and then, literally, hide that valuable information in a mountain of implementation detail--that is, of code. The catch is, once the code is written, the programmers have to make any additions or changes by modifying *the code itself* . That work is painful, slow, and prone to error. We shouldn't be touching the code at all, Simonyi says. We should be able to design functions and data structures--which intentional programming represents as "intentional trees"--and let the generator modify the code accordingly.

Discussion?

Trust

- At a workshop this past summer, Gary and Judy Olson (renowned faculty at Michigan with expertise in HCI, psychology, and more) talked about the importance of building trust for teams to be successful
  - [It wasn't a software domain primarily, if I remember correctly, but that's not material to the point.]
- They consider ways to build trust at a distance, the benefits of radically collocated teams, the issues of culture in building trust, and more…

I like the “trust” word

- Indeed, it plays a crucial role in effective software engineering in a large number of dimensions
- Does …
  - … the development team trust one another?
  - … the development team trust the test team?
  - … the test team trust the development team?
  - … either team trust the management?
  - … the customer trust the company?
  - … the company trust the customer?
  - … <and more>

There are other questions of trust

- Do you trust the compiler enough to have it be the last place you look when tracking down a highly complicated bug?
  - In general, do you trust your tools?
- Do you trust the documentation to be accurate?
- Do you trust your project's schedule?
- Do you trust the researchers who say to use a new approach?

And understanding helps build trust

- I've argued several times for tools that provide multiple views on a concept and allow for some kind of automatic analysis of the consistency of those views
  - Model checking, Alloy-style checks, type checking (to some degree), etc.
- The iterative process of this "compare and contrast" style allows software engineers to build a model of these concepts that is understood with significant depth and care
- Or maybe it's Reagan-esque: "trust, but verify"
- "Learning research tells us that the time lag from experiment to feedback is critical …" --Kent Beck

What I learned from p503 this term

Actionable Principles

- They are hard to get, for whatever reasons
  - "I always wanted to be somebody, but now I realize I should have been more specific."
    –Lily Tomlin
- But since they are what people seem to want, it would be good to continue thinking about what they might look like

Denial

- Michael Jackson must have missed some forms of denial – we are all exceptionally creative about this
  - Why did Cleopatra always say "no"?
- In reading p503 papers, I think a major form of denial was "this seems like a good idea to me so I'll buy almost everything the advocates say about it"
  - This is curious to me, since researchers generally complain that practitioners demand "too much" information before buying into something

We still want silver bullets

- Agile
- Aspect-oriented approaches
- Model-driven testing, etc.
- …

- Related to the previous form of denial?
- Remember, Brooks: “there may not be a royal road, but there is a road”

Measuring improvement in software productivity would indeed be nice

- But how?

Back to you…

- Comments?
- Questions?
- Discussions?

Thank you

- Sorry for the slowness in grading
- Please fill out the evaluation forms … and, if you wish, get in touch with me by email, phone or in person if you have other feedback