Lecture 13:

Evaluation of APIs and UI Tools, API Usability, Cognitive Dimensions.

05-431/631 Software Structures for User Interfaces (SSUI)

Fall, 2022

Logistics

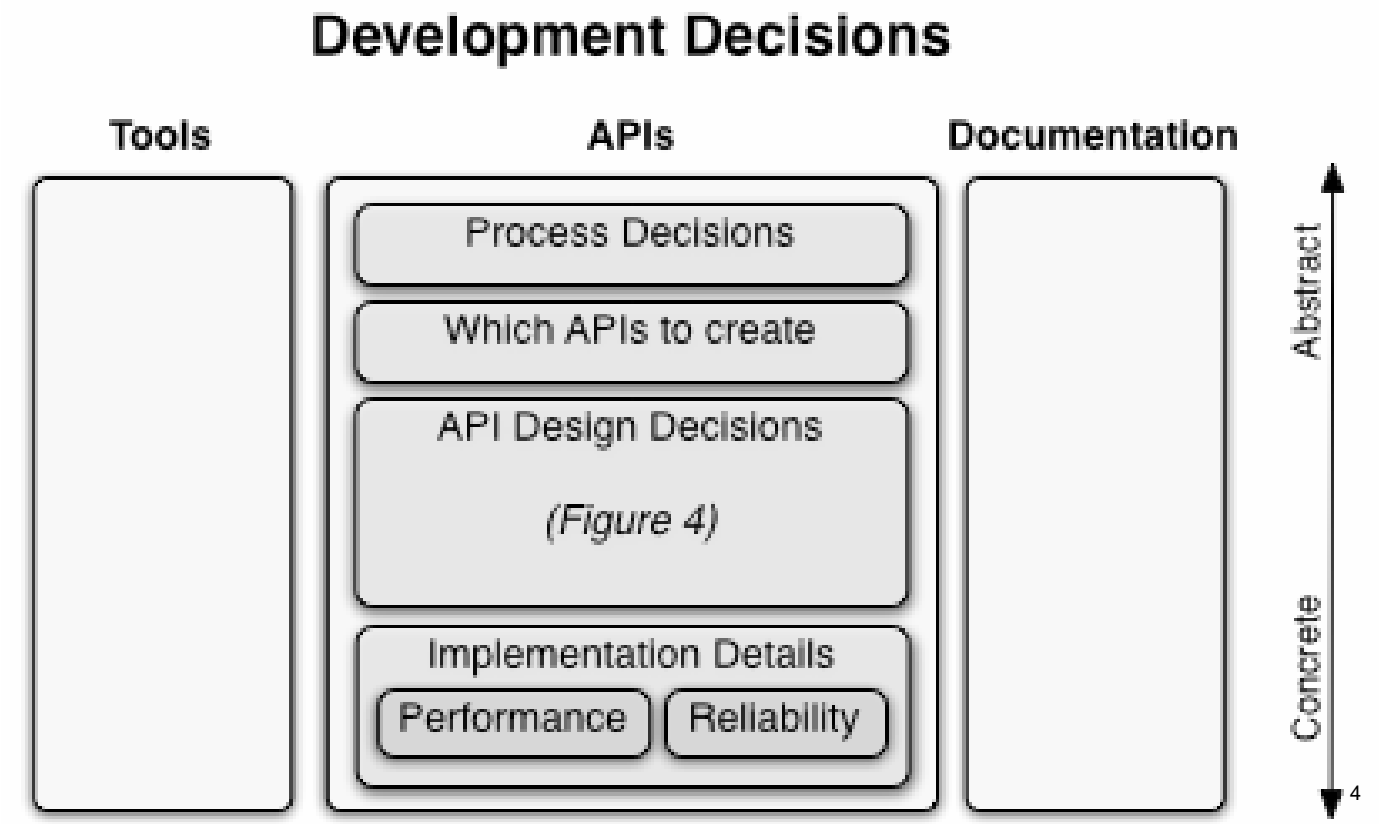
- Midterm starts *tomorrow, Wednesday 3:05 – Friday 3:05*
  - You have 48 hours!
  - No late turn-ins for the midterm
  - Will be in Canvas in the “quizzes” section
  - Note: not this lecture (covers **lectures 1-12)**
  - ***You should still come to class on Thursday!***

How Can UI Tools be Evaluated?

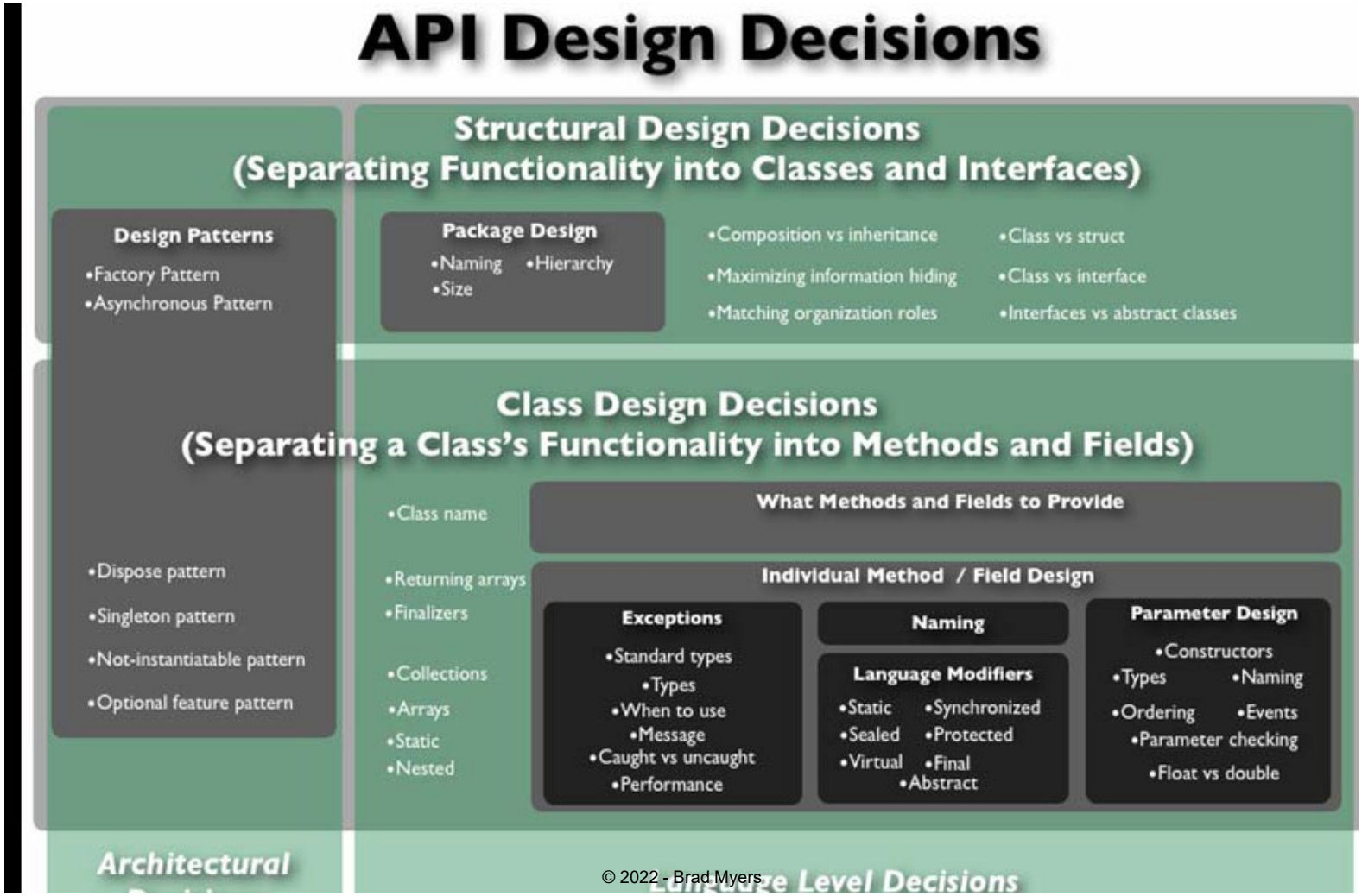
- Same as any other software
- Software Engineering Quality Metrics
  - Power (expressiveness, extensibility and evolvability)
  - Performance (speed, memory)
  - Robustness
  - Complexity
  - Defects (bugginess), …
- Same as other GUIs
  - Tool users (programmers) are people too
    - Effectiveness
    - Errors
    - Satisfaction
    - Learnability
    - Memorability
    - …

API Design Decisions

- Jeffrey Stylos and Brad Myers, "Mapping the Space of API Design Decisions," 2007 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC'07. Sept 23-27, 2007, Coeur d'Alene, Idaho. pp. 50-57. pdf



API Design Decisions, cont.



UI Evaluation of UI Software Tools:

Some Usability Methods

- Contextual Inquiry
- Contextual Analysis
- Paper prototypes
- Think-aloud protocols
- Heuristic Evaluation
- Affinity diagrams
- Personas
- Wizard of Oz
- Task analysis
- A/B testing
- Cognitive Walkthrough
- Cognitive Dimensions
- KLM and GOMS (CogTool)
- Video prototyping
- Body storming
- Expert interviews
- Questionnaires
- Surveys
- Interaction Relabeling
- Log analysis
- Storyboards
- Focus groups
- Card sorting
- Diary studies
- Improvisation
- Use cases
- Scenarios
- “Speed Dating”
- Journey Maps
- …

Dangers of *Not* Applying Human Centered Approaches

- Tools may prove to be not useful
  - Useful = solves an important problem
    - Happens frequently
    - Difficult to solve otherwise
    - Developers believe academic too s solve unimportant problems
      [How do practitioners perceive Software Engineering Research? http://catenary.wordpress.com/2011/05/19/how-dopractitioners-perceive-software-engineering-research/]
  - Tools may not actually solve the problem
    - Example: a study suggested that Tarantula tool identifying potentially faulty statements for debugging was not helpful
      - Changed the task, but telling if the identified statement was *actually* faulty not easier than finding the bug
      - Parnin, C. and Orso, A. 2011. Are Automated Debugging Techniques Actually Helping Developers International Symposium on Software Testing and Analyisis (2011), 199–209.

Dangers of Not Applying Human Centered Approaches

- Tools may show no measurable impact
  - Desired advantage overwhelmed by problems with other parts
  - Example: Emerson Murphy-Hill found that refactoring tools are under-utilized and programmers do not configure them due to usability issues
    - Emerson Murphy-Hill, Chris Parnin, Andrew P. Black. How we refactor, and how we know it. In *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering* (2009), pp. 287-297.

Study of API Usability

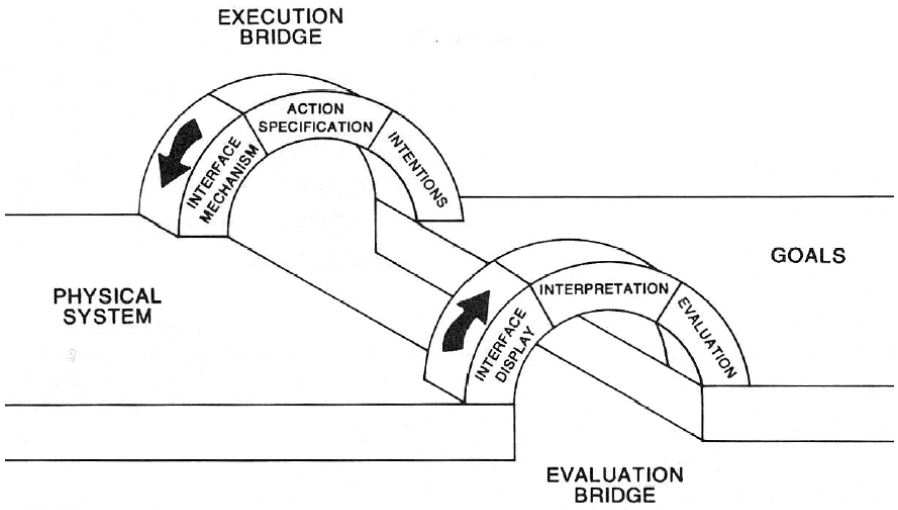
- Duala-Ekoko, E., Robillard, M.P., 2012. Asking and answering questions about unfamiliar APIs: An exploratory study. In: Proceedings of the 34th International Conference on Software Engineering. ICSE '12, pp. 266–276. http://dl.acm.org/citation.cfm?id=2337223.2337255
- Think-aloud protocols, screen captures and interviews
- 20 participants working on two programming tasks on different APIs
- 5 hardest problems:
  - Which keywords best describe a functionality provided by the API?
  - How is the type X related to the type Y?
  - Does the API provide a helper-type for manipulating objects of a given type?
  - How do I create an object of a given type without a public constructor?
  - How do I determine the outcome of a method call?

Coordination / Dependencies

- "Discovering Relevant Dependencies" among the classes
- A. J. Ko, Brad A. Myers, and Htet Htet Aung. "Six Learning Barriers in End-User Programming Systems." *VL/HCC'04: IEEE Symposium on Visual Languages and Human-Centric Computing*, Rome, Italy, September 26-29, 2004. pp. 199-206. pdf. **(Winner, Most Influential Paper Award for important influences on VL/HCC research or commerce over the last 10+/-1 years in 2013.)**
  - **Design** - I don't know what I want the computer to do...
  - **Selection** - I think I know what I want the computer to do, but I don't know what to use
  - **Coordination** - I think I know what things to use, but I don't know how to make them work together
  - **Use** - I think I know what to use, but I don't know how to use it
  - **Understanding** - I thought I knew how to use this, but it didn't do what I expected
  - **Information** - I think I know why it didn't do what I expected, but I don't know how to check

Don Norman's "Gulfs"

- Donald A. Norman and Stephen W. Draper. 1986. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Assoc. Inc., Hillsdale, NJ, USA.
- "Gulf of Evaluation"
- "Gulf of Execution"
- 7 stages:
  - Form goal
  - Form intention
  - Specify action
  - Execute action
  - Perceive state
  - Interpret state
  - Evaluate outcome



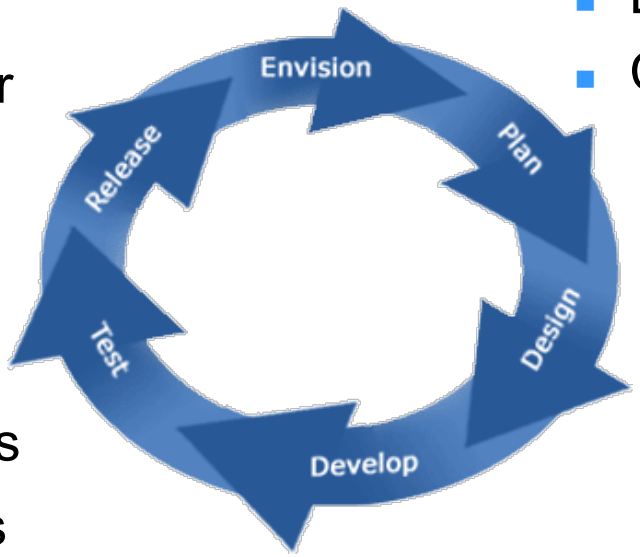
Product Lifecycle

Exploratory Studies

- Contextual Inquiries
- Interviews
- Surveys
- Lab Studies
- Corpus data mining

Field Studies

- Logs & error reports



Evaluative Studies

- Expert analyses
- Usability Evaluation
- Formal Lab studies

Design Practices

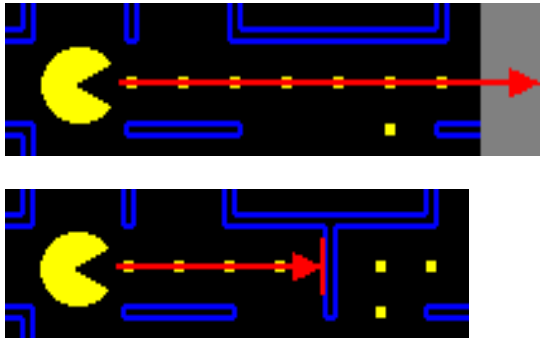
- "Natural programming"
- Graphic & Interaction Design
- Prototyping
- Wizard of Oz

Design and Development

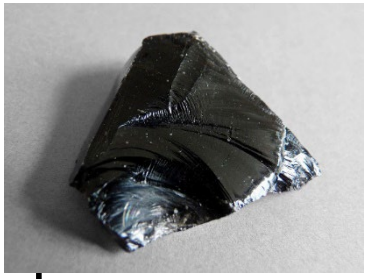
- Use CIs, other field studies and surveys to find problems to solve
  - Ko, A.J., Myers, B.A., and Aung, H.H. “Six Learning Barriers in End-User Programming Systems,” in *IEEE VL/HCC’2004*. pp. 199-206.
  - Ko, A.J. and DeLine, R. “A Field Study of Information Needs in Collocated Software Development Teams,” in *ICSE'2007.*
  - Thomas D. LaToza and Brad Myers. "Developers Ask Reachability Questions", *ICSE'2010: 32nd International Conference on Software Engineering*, Cape Town, South Africa, 2-8 May 2010. pp. 185-194. pdf
  - Also interviews and surveys, etc.: Myers, B., Park, S.Y., Nakano, Y., Mueller, G., and Ko, A. “How Designers Design and Program Interactive Behaviors,” in *IEEE VL/HCC‘2008.* pp. 185-188.
- Iterative design and usability testing of versions
- Summative testing at end

"Natural Programming" Elicitation Method

- Technique developed by my group to discover developer's "natural" expressions
  - Mental models of tasks, vocabulary, etc.
- A form of participatory design
- Blank paper tests
- Must prompt for the tasks in a way that doesn't bias the answers
- Examples:
  - PacMan before and after
    - Mostly rule-based (if-then)
  - API designs
    - Architecture, names used, which methods are on which classes

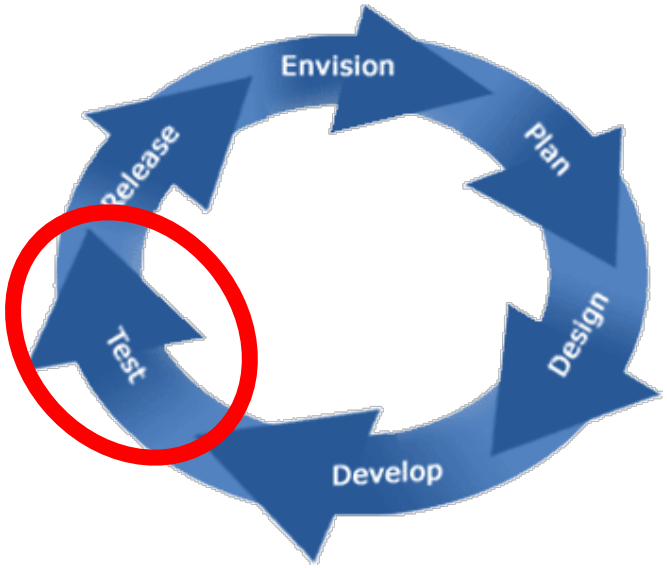


Example of use of Natural Programming



- Obsidian is a domain specific language (DSL) for blockchains [Coblenz, et al, 2019]
  - **O**bject-oriented **B**lockchain **S**tate **I**nteraction and **D**evelopment **I**mplementation **A**nd **N**otation
- Combining state transition language *("TypeStates")* with *resources ("linear types")* all checked statically
- 11 different NatProg studies on how to present these complex concepts

Evaluation Methods



- Does my tool work?
- Does it solve the developer's problems?
- "If the user can't use it, it doesn't work!"
  - Susan Dray

Dray & Associates
Human Centered Innovation

Expert Analyses

- Usability experts evaluating designs to look for problems
  - Heuristic Analysis – [Nielsen] set of guidelines
  - Cognitive Dimensions – [Green] another set
  - Cognitive Walkthroughs – evaluate a task
- Can be inexpensive and quick
- However, experienced evaluators are better
  - 22% vs. 41% vs. 60% of errors found [Nielsen]
- Disadvantage: “just” opinions, open to arguments
- [Nielsen] Jakob Nielsen. Usability Engineering. Boston, Academic Press. 1993.
- [Green] T.R.G. Green and M. Petre. “Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework,” Journal of Visual Languages and Computing. 1996. vol. 7, no. 2. pp. 131-174.

Heuristic Evaluation Method

- Named by Jakob Nielsen
- Expert evaluates the user interface using guidelines
- "Discount" usability engineering method
  - One case study found factor of 48 in cost/benefit:
    - Cost of inspection: $10,500. Benefit: $500,000 [Nielsen]

10 Basic Principles

From Nielsen's web page:
http://www.useit.com/papers/heuristic/heuristic_list.html

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and Documentation

- Slightly different from list in Nielsen's text

Cognitive Dimensions

- 12 different dimensions (or factors) that individually and collectively have an impact on the way that developers work with an API and on the way that developers expect the API to work. (from Clarke'04)
  - Abstraction level. The minimum and maximum levels of abstraction exposed by the API
  - Learning style. The learning requirements posed by the API, and the learning styles available to a targeted developer.
  - Working framework. The size of the conceptual chunk (developer working set) needed to work effectively.
  - Work-step unit. How much of a programming task must/can be completed in a single step.
  - Progressive evaluation. To what extent partially completed code can be executed to obtain feedback on code behavior.
  - Premature commitment. The amount of decisions that developers have to make when writing code for a given scenario and the consequences of those decisions.
  - Penetrability. How the API facilitates exploration, analysis, and understanding of its components, and how targeted developers go about retrieving what is needed.
  - Elaboration. The extent to which the API must be adapted to meet the needs of targeted developers.
  - Viscosity. The barriers to change inherent in the API, and how much effort a targeted developer needs to expend to make a change.
  - Consistency. How much of the rest of an API can be inferred once part of it is learned.
  - Role expressiveness. How apparent the relationship is between each component exposed by an API and the program as a whole.
  - Domain correspondence. How clearly the API components map to the domain and any special tricks that the developer needs to be aware of to accomplish some functionality.

Example: Consistency Issues in html/CSS/JavaScript?

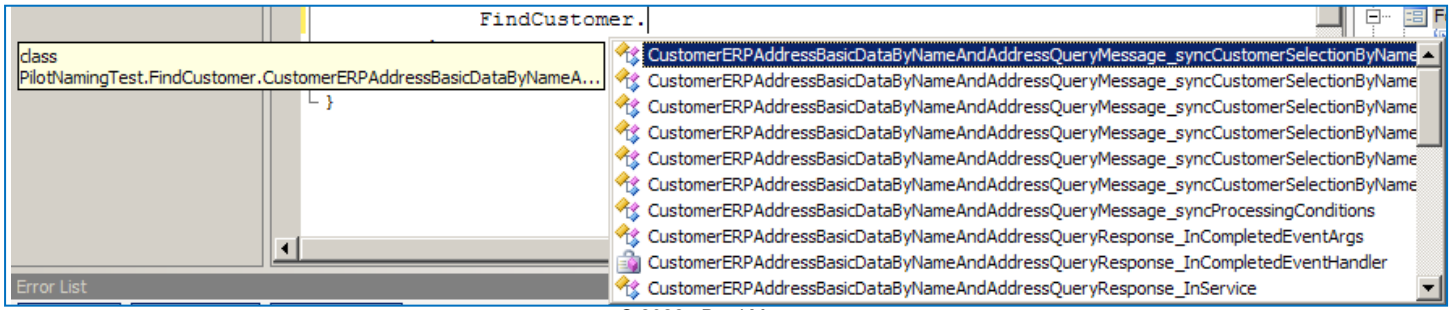
Our Use of Expert Analyses

- Study APIs for Enterprise Service-Oriented Architecture - eSOA ("Web Services")
  - Cite: Jack Beaton, Sae Young Jeong, Yingyu Xie, Jeffrey Stylos, Brad A. Myers. "Usability Challenges for Enterprise Service-Oriented Architecture APIs," *VL/HCC'08*. Sept 15-18, 2008, Herrsching am Ammersee, Germany. pp. 193-196.
- HEs and Usability Evaluations
- Naming problems:
  - Too long

```
MaterialSimpleByIDAndDescriptionQueryMessage_syncMaterialSimpleSelectionByIDAndDescriptionSelectionByMaterialDescription
```

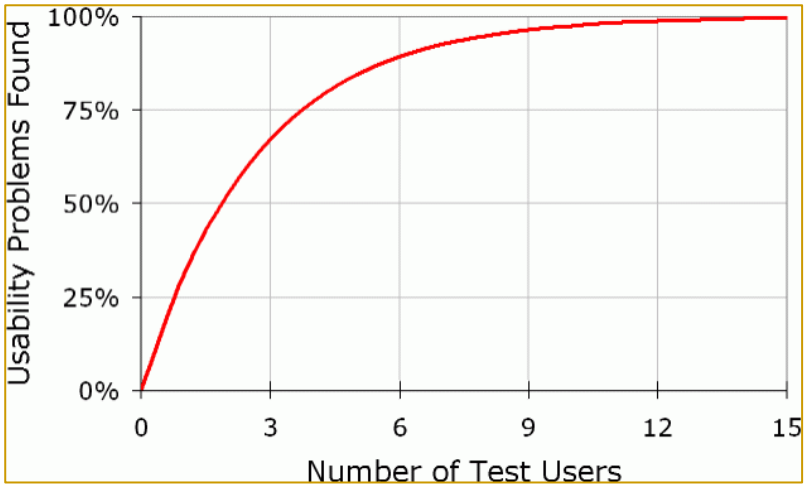
  - Not understandable
  - Differences in *middle* are frequently missed

CustomerAddressBasicDataByNameAndAddressRequestMessageCustomerSelectionCommonName
CustomerAddressBasicDataByNameAndAddressResponseMessageCustomerSelectionCommonName



Usability Evaluations with users

- Different from formal A vs. B "user studies"
  - Understand usability issues
  - Should be done early and often
    - Doesn't have to be "finished" to let people try it
- "Think aloud" protocols
  - "Single most valuable usability engineering method" -- [Nielsen]
  - Users verbalize what they are thinking
    - Motivations, *why* doing things, what confused about
  - Don't need many users



Why Usability Analysis

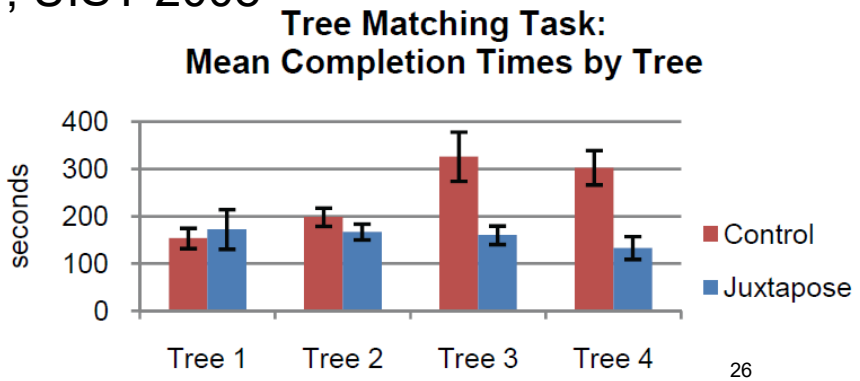
- Improve the user interface prior to:
  - Deployment
  - A vs. B testing (as a “pilot” test)
- Demonstrate that users can use the system
  - Show that novel features of the UI are understandable

Formal A vs. B "User Studies"

- Formal *A vs. B* lab user studies are "gold standard" for academic papers – to show something is better
- But many issues in the study design
- Issues:
  - Vast differences in programmer productivity
    - 10X often cited (cites: Sackman, 1968, Curtis 1981, Mills 1983, DeMarco and Lister 1985, Curtis et al. 1986, Card 1987, Boehm and Papaccio 1988, Valett and McGarry 1989, Boehm et al 2000)
  - Difficulty of controlling for prior knowledge
  - Usually really care about expert performance, which is difficult to measure in a user test
  - "Confounding" factors which were not controlled and are not relevant to study, but affect results
  - Tasks or instructions are mis-understood
    - Use prototypes & pilot studies to find these
- Statistical significance doesn't mean real savings
- Be sure to collect qualitative data too
  - Strategies people are using
  - Why users did it that way
  - Especially when unexpected results

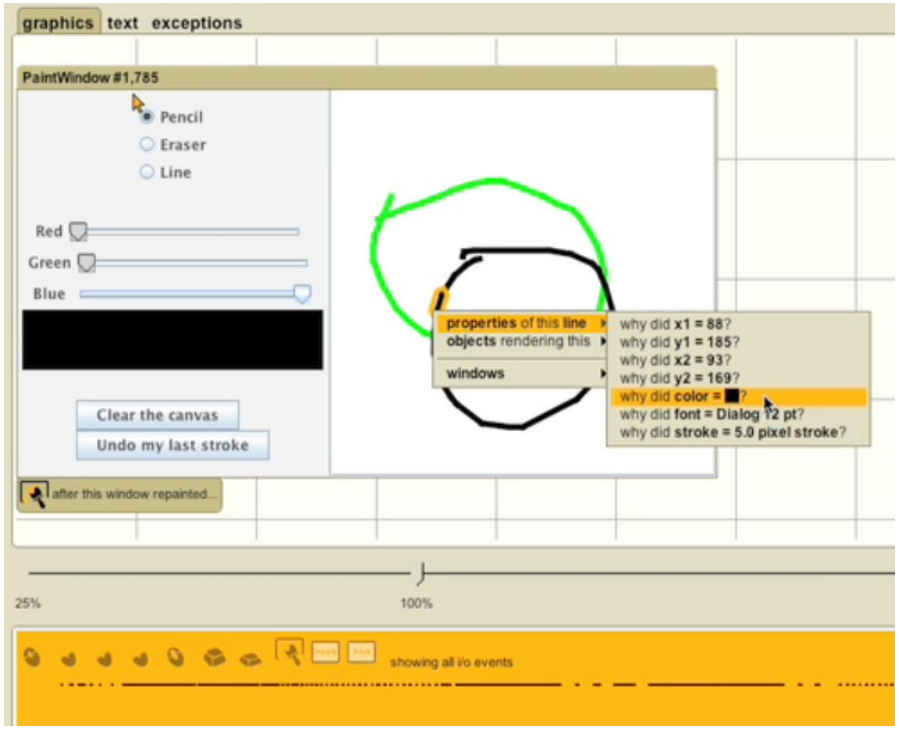
Examples of UI Tests

- Many tool papers have user tests
  - Especially at CHI conference
    - E.g.: Ellis, J. B., Wahid, S., Danis, C., and Kellogg, W. A. 2007. Task and social visualization in software development: evaluation of a prototype. *CHI '07.* http://doi.acm.org/10.1145/1240624.1240716
      - 8 participants, 3 tasks, within subjects: Bugzilla vs. SHO, observational
  - Backlash? at UIST conference
    - Olsen, 2007: "Evaluating user interface systems research"
    - But: Hartmann, Björn,Loren Yu, Abel Allison, Yeonsoo Yang, and Scott Klemmer. "Design As Exploration: Creating Interface Alternatives through Parallel Authoring and Runtime Tuning", UIST 2008 – **Best Student Paper Award**
      - 18 participants, within subjects, full interface vs. features removed, "(one-tailed, paired Student's t-test; $p < 0.01$)"



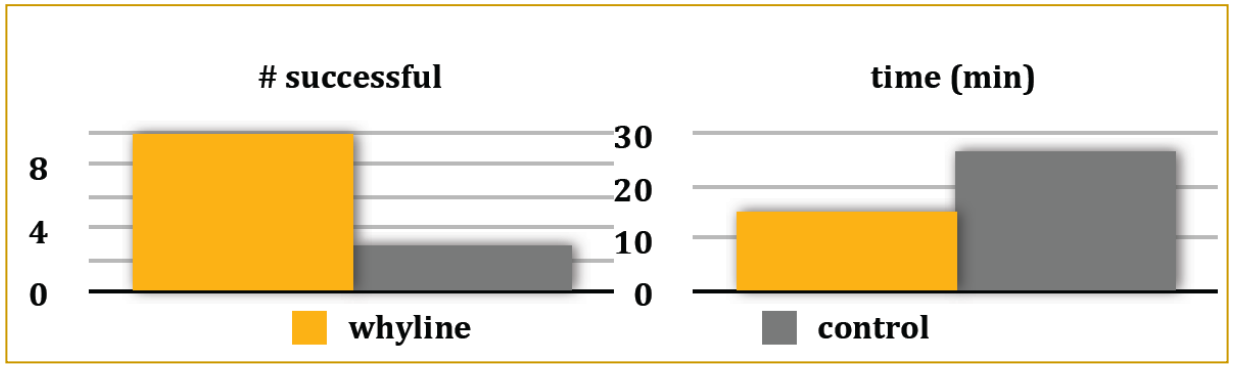
Our use of *A vs. B* Study: Whyline

- PhD work of A.J. Ko
- Allow users to directly ask “Why” and “Why not”



Whyline User Studies

- Initial study:
  - Whyline with novices outperformed experts with Eclipse
  - Factor of 2.5 times faster
- Formal study:
  - Compared to Whyline with key features removed (rather than Eclipse)
  - Tasks: 2 real bug reports from real open source system (ArgoUML)
  - Whyline was over 3 times as successful, in ½ of the time

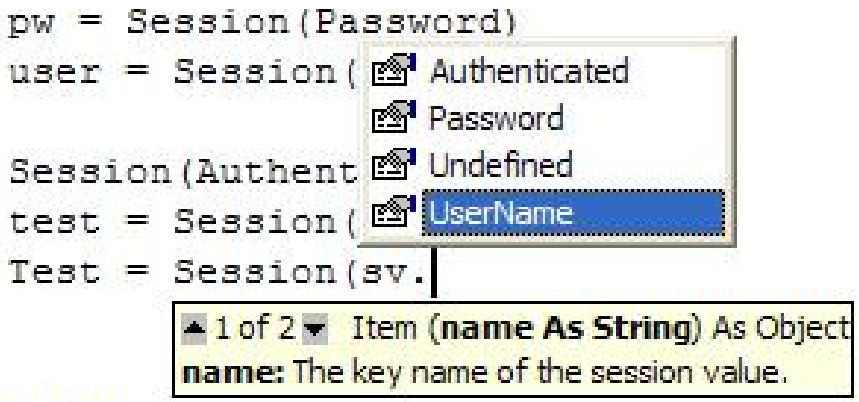


Steven Clarke's "Personas"

- Classified types of programmers he felt were relevant to UI tests of Microsoft products (Clarke, 2004) (Stylos & Clarke 2007)
- Capture different *work styles*, not experience or proficiency
- **Systematic** - work from the top down, attempting to understand the system as a whole before focusing on an individual component. Program defensively, making few assumptions about code or APIs and mistrusting even the guarantees an API makes, preferring to do additional testing in their own environment. Prefer full control, as in C, C++
- **Opportunistic** - work from the bottom up on their current task and do not want to worry about the low-level details. Want to get their code working and quickly as possible without having to understand any more of the underlying APIs than they have to. They are the most common persona and prefer simple and easy to use languages that offer high levels of productivity at the expense of control, such as Visual Basic.
- **Pragmatic** - less defensive and learn as they go, starting working from the bottom up with a specific task. However, when this approach fails, they revert to the top-down approach used by systematic programmers. Willing to trade off control for simplicity but prefer to be aware of and in control of this trade off. Prefer Java and C#.

Usability Evaluations of APIs

- PhD work of Jeff Stylos (extending Steven Clarke's work)
- Which programming patterns are most usable?
  - Default constructors
  - Factory pattern
  - Object design
  - E-SOA APIs
- Measures: learnability, errors, preferences
- Expert and novice programmers
- Fix by:
  - Changing APIs
  - Changing documentation
  - Better tools in IDEs
    - E.g., use of Code completion ("IntelliSence") for exploration



Required Constructors

- Compared create-set-call (default constructor)

```
var foo = new FooClass();
foo.Bar = barValue;
foo.Use();
```

- vs. required constructors:

```
var foo = new FooClass(barValue);
foo.Use();
```

- All participants assumed there would be a default constructor
- Required constructors interfered with learning
  - Want to experiment with what kind of object to use first
- Did *not* ensure valid objects – passed in `null`
- Preferred to *not* use temporary variables

"Factory" Pattern

- (Ellis, Stylos, Myers 2007)
- ***Covered in Lecture 9, slide 5***
- Instead of "normal" creation: `Widget w = new Widget();`
- Objects must be created by *another* class:

```
AbstractFactory f =  AbstractFactory.getDefault();
Widget w = f.createWidget();
```

- Used frequently in Java (>61) and .Net (>13) and SAP
- Lab study with expert Java programmers
  - Five programming and debugging tasks
  - Within subject and between subject measures
- Results:
  - When asked to design on "blank paper", no one designed a factory
  - Time to develop using factories took 2.1 to 5.3 times longer compared to regular constructors (20:05 v 9:31, 7:10 v 1:20)
  - All subjects had difficulties getting using factories in APIs
- Implications: avoid the factory pattern!

Summary

- CIs and Iterative Design to help *design and develop* better tools
- User testing is still the “gold standard” for user interface tools
- HE and CD are useful for evaluations