Lecture 24: Course Summary

What we've covered in this course

Some underlying principles

Course Evaluation

© 2008 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Course Outline

Modeling
- Sketching vs. Blueprints (vs. programming)
- Structure vs. Behaviour vs. Function
- Abstraction, Decomposition, Projection
- UML

Maintenance and Re-engineering
- Software Evolution
- Program Comprehension
- Reverse Engineering for Design Recovery

Software Architecture
- Conway's Law
- Coupling and Cohesion
- Architectural Patterns

Software Processes
- Agile vs. Disciplined
- Iterative development
- RUP, ICONIX, XP, SCRUM,…
- QA and process improvement

Project Management
- Resources, Time, Product, Risk
- Estimation & Prioritization
- Risk Assessment & Control
- Monitoring and Controling a project
- Organising a team

Requirements Analysis
- Requirements vs. Specifications
- Stakeholders, Goals, Obstacles
- Use Cases
- Robustness Analysis

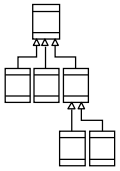
Verification and Validation
- Testing
- Static Analysis
- Inspection
- Prototyping
- Formal model analysis

Software Quality…

© 2008 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

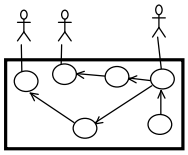
Modeling Notations

UML Class Diagrams



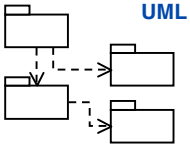
information structure
relationships between data items
modular structure for the system

Use Cases



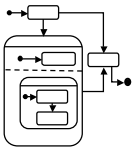
user's view
Lists functions
visual overview of the main requirements

UML Package Diagrams



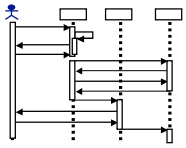
Overall architecture
Dependencies between components

(UML) Statecharts



responses to events
dynamic behavior
event ordering, reachability, deadlock, etc

UML Sequence Diagrams



individual scenario
interactions between users and system
Sequence of messages

Goal Models

Stakeholder's goals and priorities
Means-ends analysis and rationale
dependencies between stakeholders

© 2008 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Why models are important

Abstraction

Ignore detail to see the big picture
Treat objects as the same by ignoring certain differences
(beware: every abstraction involves choice over what is important)

Decomposition

Partition a problem into independent pieces, to study separately
(beware: the parts are rarely independent really)

Projection

Separate different concerns (views) and describe them separately
Different from decomposition as it does not partition the problem space
(beware: different views will be inconsistent most of the time)

Modularization

Choose structures that are stable over time, to localize change
(beware: any structure will make some changes easier and others harder)

© 2008 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Scaling Up

Complexity grows rapidly

"For every 25% increase in problem complexity there is a 100% increase in solution complexity" (Robert Glass)

Why?

- Software development is largely an intellectual task (80% intellectual, 20% clerical)
- To scale up, you need more brains
- Software development becomes a social activity
- Coordinating more people is hard

© 2008 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.

Glass's Facts (slightly refactored)

Adapted from Robert Glass "Facts and Fallacies of Software Engineering"

People

- Most important factor is quality of your developers
- Best programmers are 28 times more effective than the worst

Tools

- There is no silver bullet
- Each tool/technique offers only small improvements
- Any new tool/technique initially causes a reduction in productivity
- Most tools become shelfware

Estimation

- Poor estimation is endemic
- Estimation is done by the wrong people, at the wrong time, and never adjusted…

Re-use

- Re-use in the small is solved;
- Re-use in the large is intractable

Requirements

- Requirements errors are the most expensive to fix during development
- Missing requirements are the hardest errors

Design

- Design is a complex, iterative process
- There is seldom one best design

Testing

- 55-60% branch coverage is typical
- 100% coverage is unachievable
- 100% coverage is insufficient

Defects

- Error removal is the most time-consuming part of software development
- Errors tend to cluster (80:20)
- Most programmers make the same mistakes

Maintenance

- Maintenance is 40-80% of software costs
- Understanding the existing product is the hardest part

© 2008 Steve Easterbrook. This presentation is available free for non-commercial use with attribution under a creative commons license.