Lecture 7: Software Processes

- ➜ What is a Software Development Process
- ➜ The Lifecycle of a Software Project
- ➜ Agile vs. Disciplined
- ➜ Some common approaches:
  - RUP, SCRUM, XP, ICONIX,…
- ➜ Where UML fits in

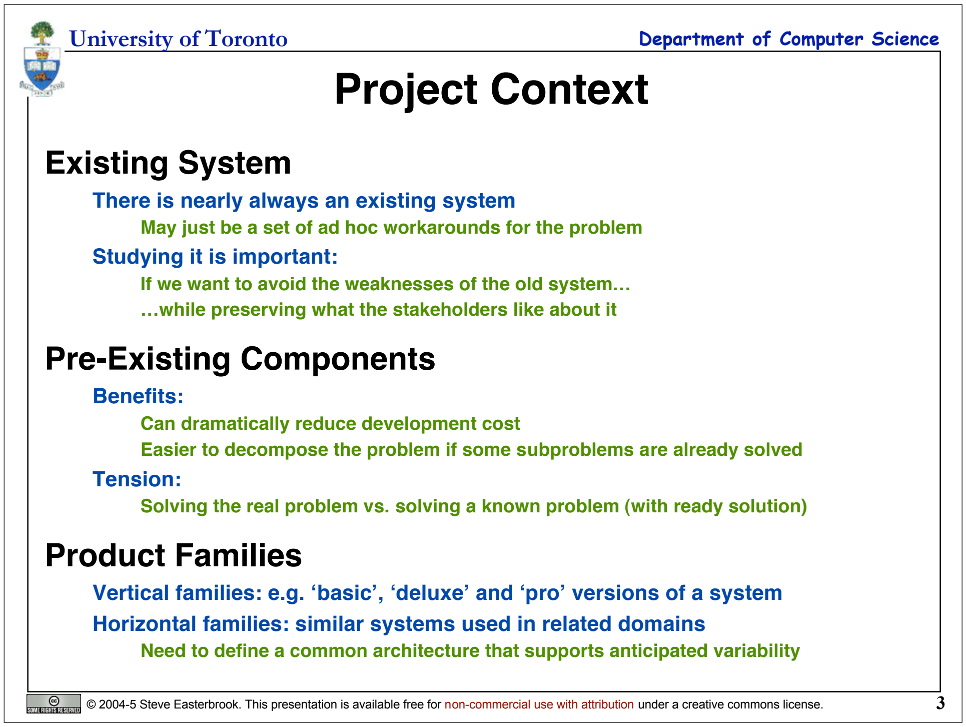
Project Types

Reasons for initiating a software development project

- Problem-driven: competition, crisis,…
- Change-driven: new needs, growth, change in business or environment,…
- Opportunity-driven: exploit a new technology,…
- Legacy-driven: part of a previous plan, unfinished work, …

Relationship with Customer(s):

- Customer-specific - one customer with specific problem
  - May be another company, with contractual arrangement
  - May be a division within the same company
- Market-based - system to be sold to a general market
  - In some cases the product must generate customers
  - Marketing team may act as substitute customer
- Community-based - intended as a general benefit to some community
  - E.g. open source tools, tools for scientific research
  - funder ≠ customer (if funder has no stake in the outcome)
- Hybrid (a mix of the above)



Project Context

Existing System

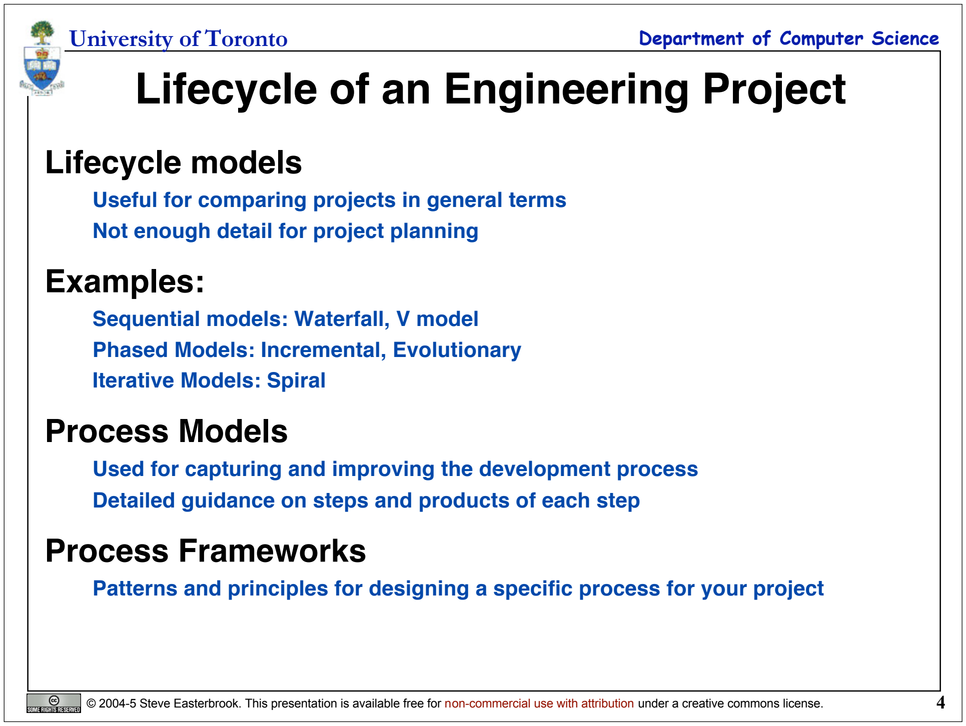
- There is nearly always an existing system
  - May just be a set of ad hoc workarounds for the problem
- Studying it is important:
  - If we want to avoid the weaknesses of the old system…
  - …while preserving what the stakeholders like about it

Pre-Existing Components

- Benefits:
  - Can dramatically reduce development cost
  - Easier to decompose the problem if some subproblems are already solved
- Tension:
  - Solving the real problem vs. solving a known problem (with ready solution)

Product Families

- Vertical families: e.g. ‘basic’, ‘deluxe’ and ‘pro’ versions of a system
- Horizontal families: similar systems used in related domains
  - Need to define a common architecture that supports anticipated variability



Lifecycle of an Engineering Project

Lifecycle models

- Useful for comparing projects in general terms
- Not enough detail for project planning

Examples:

- Sequential models: Waterfall, V model
- Phased Models: Incremental, Evolutionary
- Iterative Models: Spiral

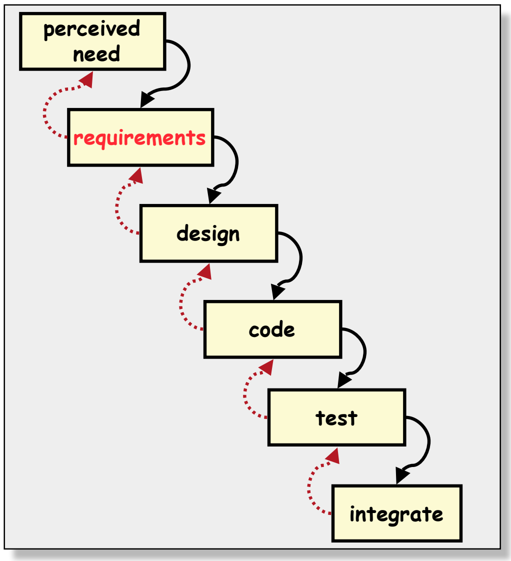
Process Models

- Used for capturing and improving the development process
- Detailed guidance on steps and products of each step

Process Frameworks

- Patterns and principles for designing a specific process for your project

Waterfall Model



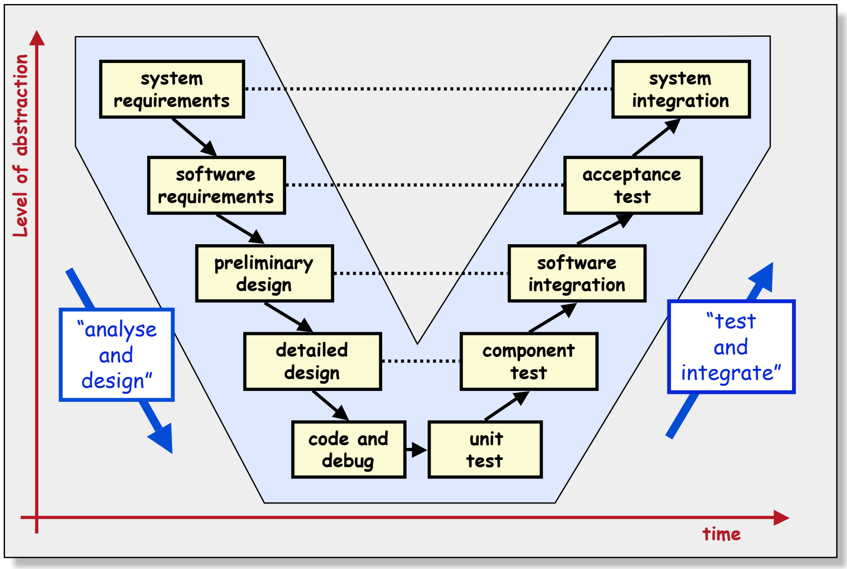
View of development:

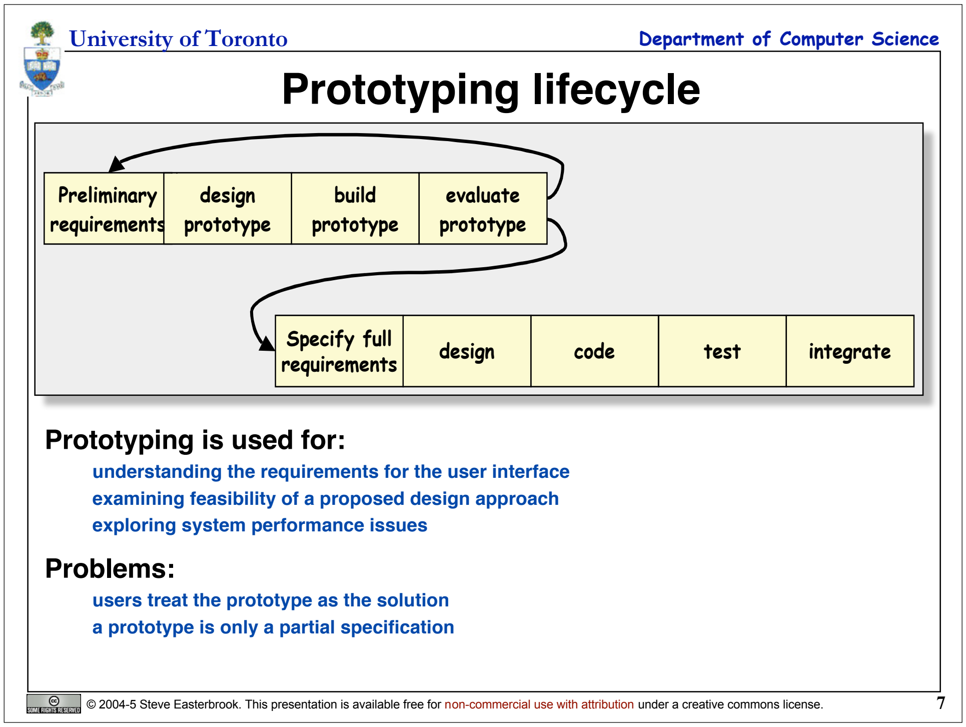
- a process of stepwise refinement
- largely a high level management view

Problems:

- Static view of requirements - ignores volatility
- Lack of user involvement once specification is written
- Unrealistic separation of specification from design
- Doesn't accommodate prototyping, reuse, etc.

V-Model





Prototyping lifecycle

Preliminary requirements | design prototype | build prototype | evaluate prototype

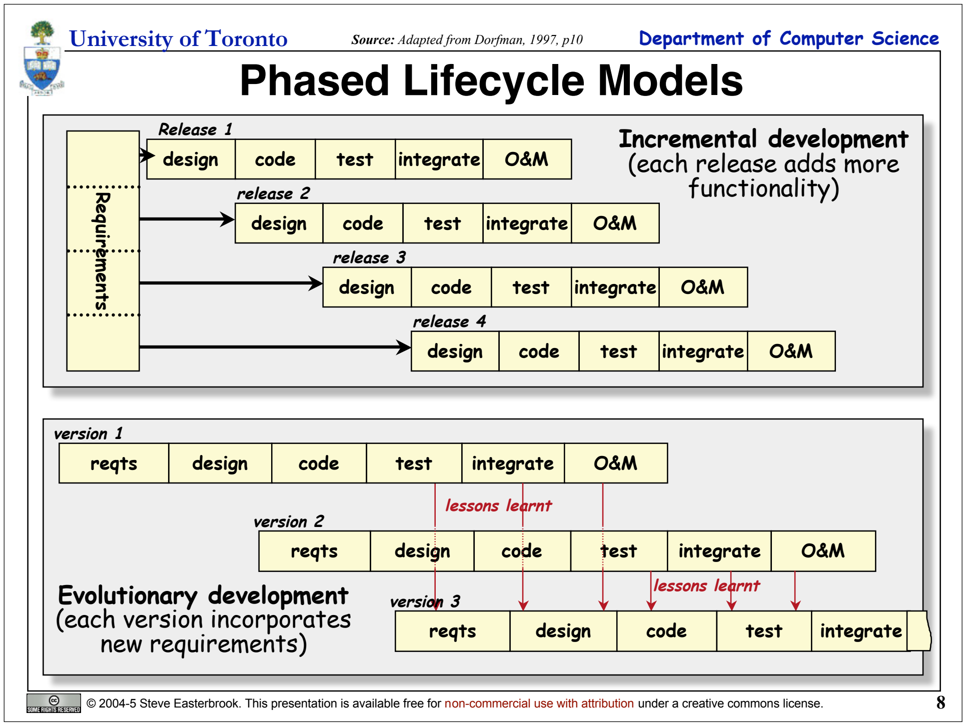
Specify full requirements | design | code | test | integrate

Prototyping is used for:

- understanding the requirements for the user interface
- examining feasibility of a proposed design approach
- exploring system performance issues

Problems:

- users treat the prototype as the solution
- a prototype is only a partial specification



Source: Adapted from Dorfman, 1997, p10

Phased Lifecycle Models

Requirements

Release 1: design | code | test | integrate | O&M

release 2: design | code | test | integrate | O&M

release 3: design | code | test | integrate | O&M

release 4: design | code | test | integrate | O&M

Incremental development (each release adds more functionality)

version 1: reqts | design | code | test | integrate | O&M

lessons learnt

version 2: reqts | design | code | test | integrate | O&M

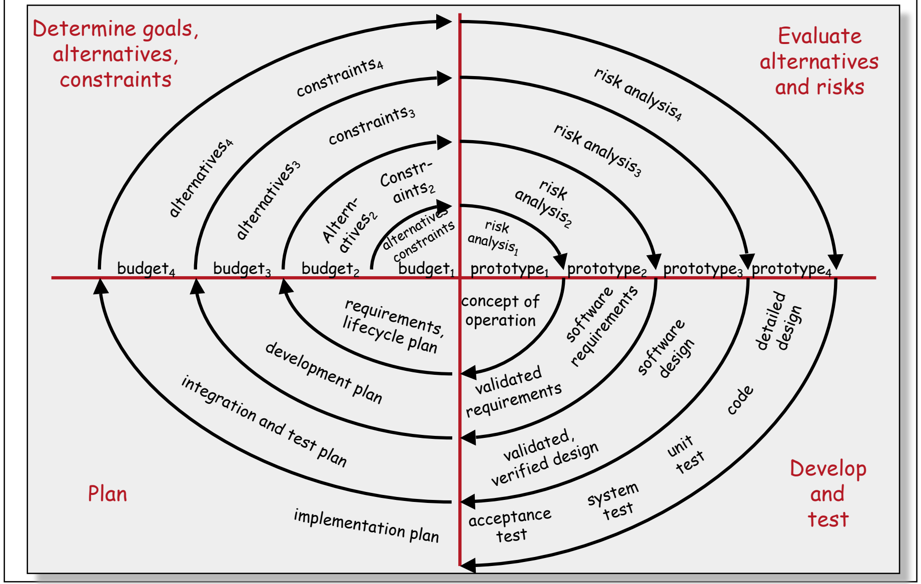
lessons learnt

version 3: reqts | design | code | test | integrate

Evolutionary development (each version incorporates new requirements)

***Source:** Adapted from Pfleeger, 1998, p57*

The Spiral Model



“Agile” vs “Disciplined”

Iterative	**Planned**
Small increments	**Analysis before design**
Adaptive planning	**Prescriptive planning**
Embrace change	**Control change**
Innovation and exploration	**High ceremony**
Trendy	**Traditional**
Highly fluid	**Upfront design / architecture**
Feedback driven	**Negotiated requirements**
Individuals and Interactions	**Processes and Tools**
Human communication	**Documentation**
Small teams	**Large teams**

Rational Unified Process (RUP)

Inception

- Establish Scope
- Build a business case
- Get stakeholder buy-in

Elaboration

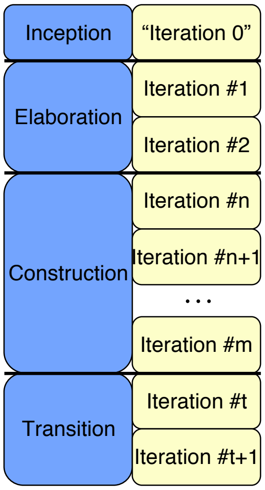
- Identify and manage risks
- Build an executable architecture
- Focus only on high risk items

Construction

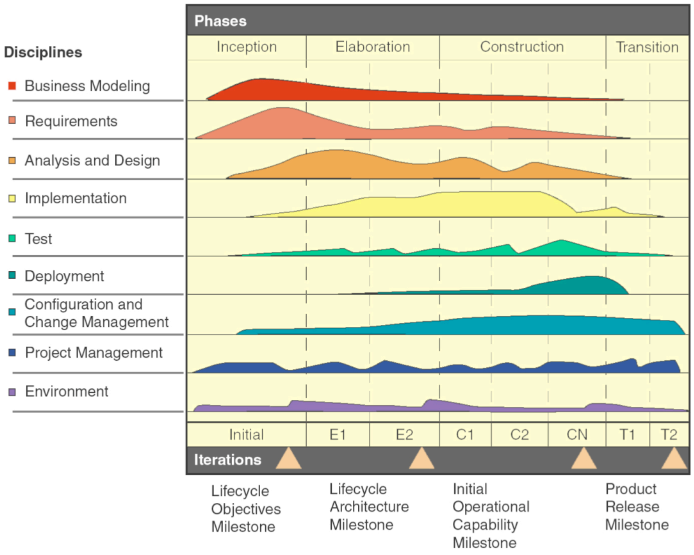
- Iteratively build operational version
- Develop support docs and training materials

Transition

- Fine-tune
- Resolve configuration, installation and usability issues



RUP Activities



SCRUM

Sprint - 30 day iteration

- Starts with 1/2 day planning meeting
- Starts with Prioritized Product Backlog (from product owner)
- Builds a Sprint Backlog - items to be done in this sprint
- 29 days of development
- 1/2 day Sprint review meeting - inspect product, capture lessons learnt

Daily Scrum

- 15 minute team meeting each day.
- Each team member answers:
  - What have you done since last meeting?
  - What will you do between now and the next meeting?
  - What obstacles stood in the way of doing work?
- Scrum master keeps meeting on track

Scrum teams

- Cross-functional, 7 (±2) members
- Teams are self-organising

Extreme Programming

Fine Scale Feedback

- Pair Programming
- Planning Game
- Test-driven Development
- Whole team (customer part of team)

Continuous Process

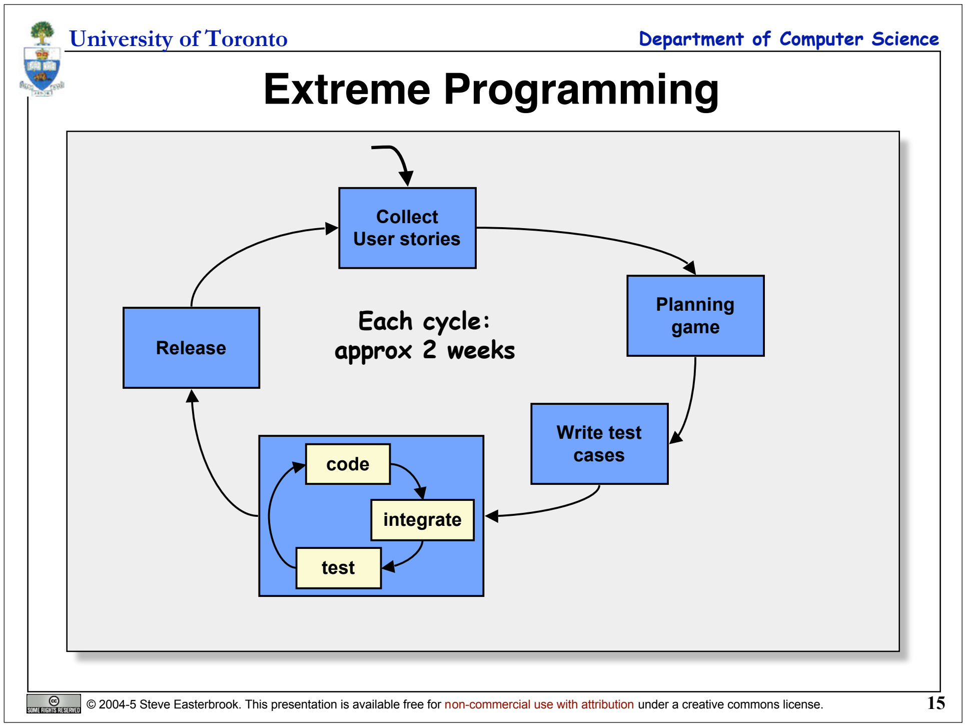
- Continuous Integration
- Design Improvement (refactoring)
- Small Releases

Shared Understanding

- Coding Standards
- Collective Code Ownership
- Simple Design
- System Metaphor

Programmer Welfare

- Sustainable pace (40 hour week)



Extreme Programming

Collect User stories

Planning game

Write test cases

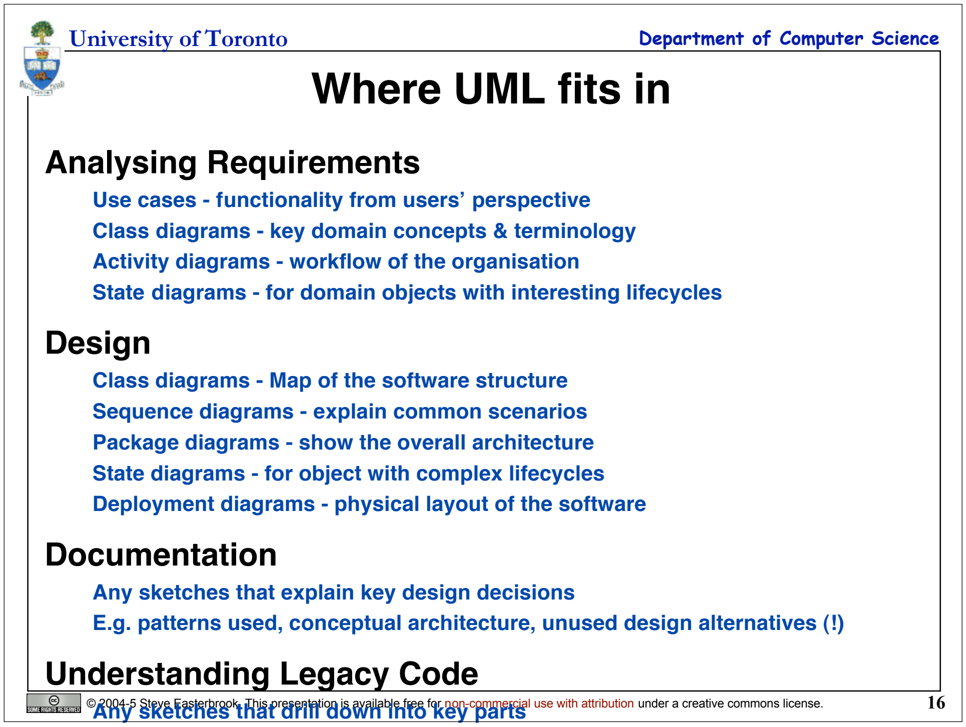
code

integrate

test

Release

Each cycle: approx 2 weeks



Where UML fits in

Analysing Requirements

- Use cases - functionality from users' perspective
- Class diagrams - key domain concepts & terminology
- Activity diagrams - workflow of the organisation
- State diagrams - for domain objects with interesting lifecycles

Design

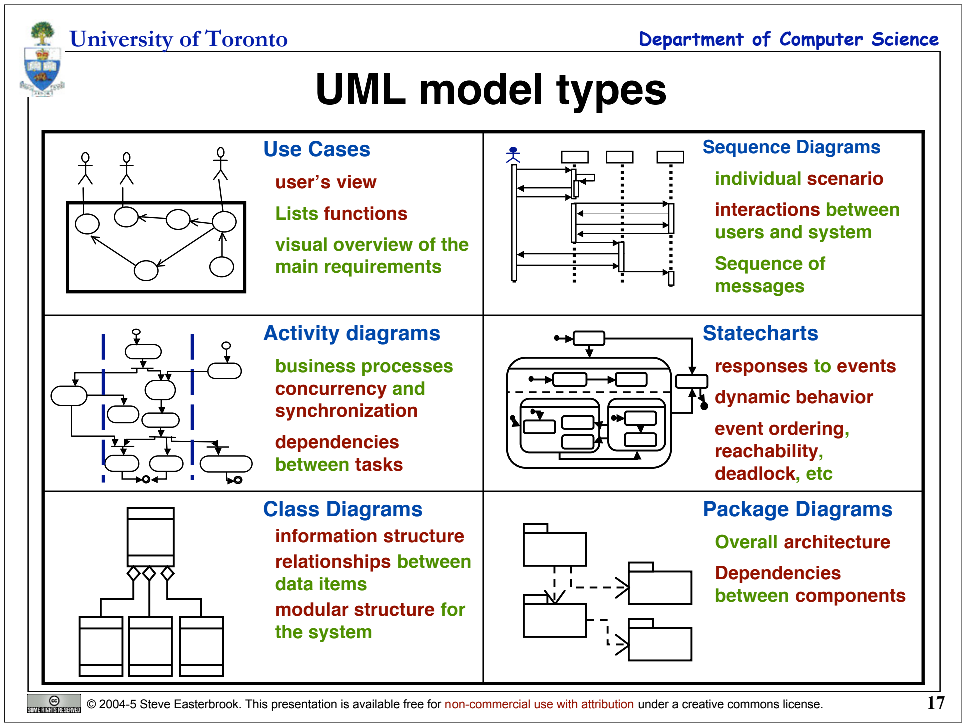
- Class diagrams - Map of the software structure
- Sequence diagrams - explain common scenarios
- Package diagrams - show the overall architecture
- State diagrams - for object with complex lifecycles
- Deployment diagrams - physical layout of the software

Documentation

- Any sketches that explain key design decisions
- E.g. patterns used, conceptual architecture, unused design alternatives (!)

Understanding Legacy Code

- Any sketches that drill down into key parts



UML model types

Use Cases

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

Sequence Diagrams

- individual scenario
- interactions between users and system
- Sequence of messages

Activity diagrams

- business processes
- concurrency and synchronization
- dependencies between tasks

Statecharts

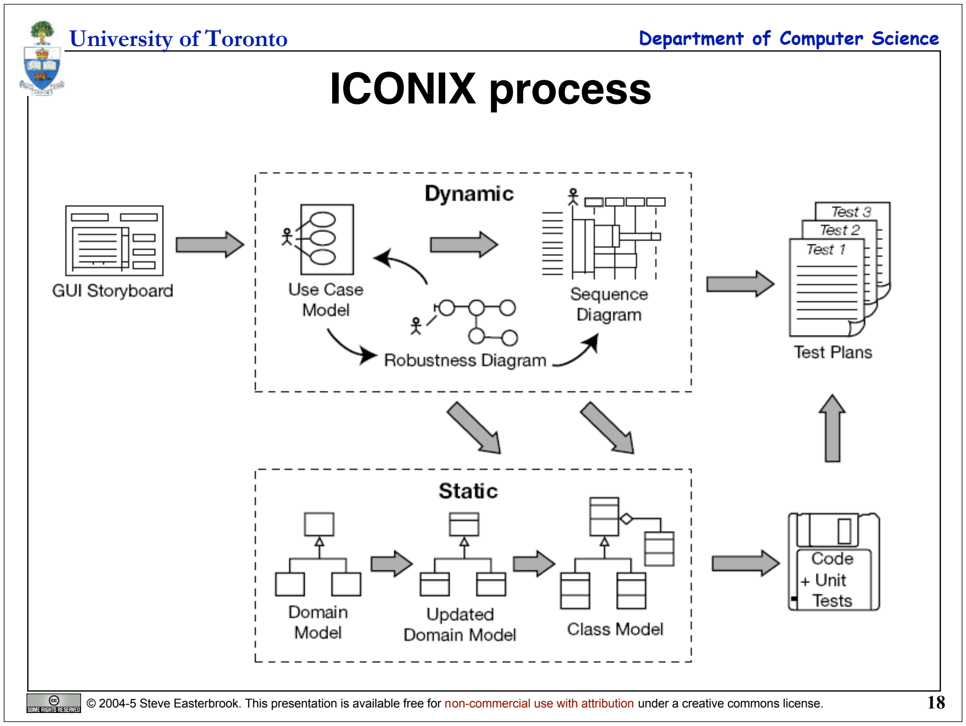
- responses to events
- dynamic behavior
- event ordering, reachability, deadlock, etc

Class Diagrams

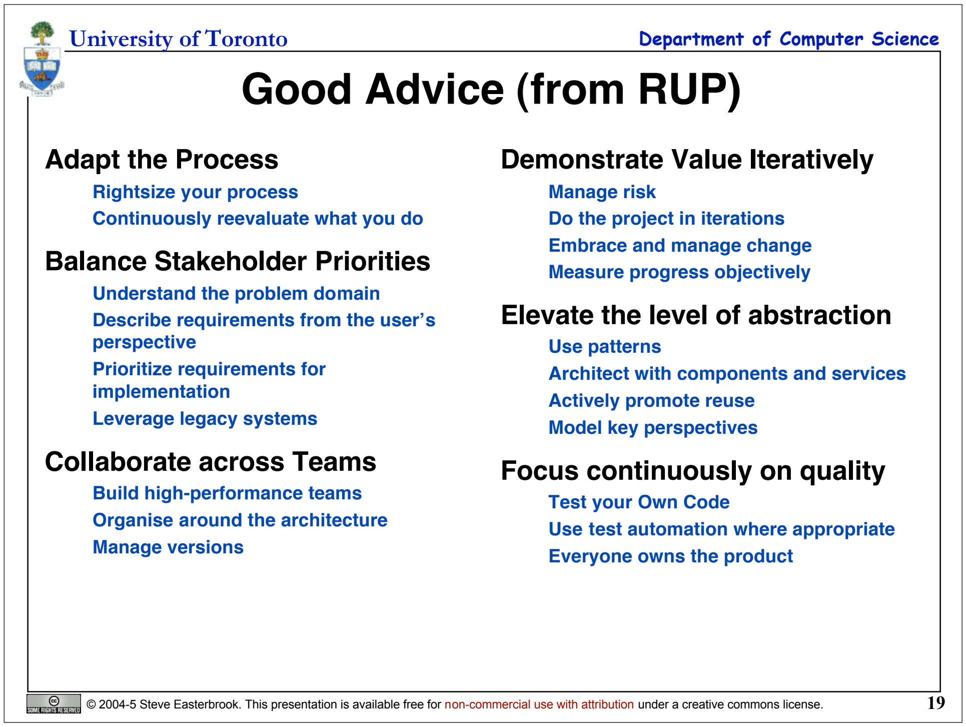
- information structure
- relationships between data items
- modular structure for the system

Package Diagrams

- Overall architecture
- Dependencies between components



ICONIX process



Good Advice (from RUP)

Adapt the Process

- Rightsize your process
- Continuously reevaluate what you do

Balance Stakeholder Priorities

- Understand the problem domain
- Describe requirements from the user’s perspective
- Prioritize requirements for implementation
- Leverage legacy systems

Collaborate across Teams

- Build high-performance teams
- Organise around the architecture
- Manage versions

Demonstrate Value Iteratively

- Manage risk
- Do the project in iterations
- Embrace and manage change
- Measure progress objectively

Elevate the level of abstraction

- Use patterns
- Architect with components and services
- Actively promote reuse
- Model key perspectives

Focus continuously on quality

- Test your Own Code
- Use test automation where appropriate
- Everyone owns the product