Software Development Processes

Designing the development process

Plans & Processes

We make a *plan* for an individual project

But we seldom start from scratch ...

A "process" or "process model" is a pattern for planning and managing projects

- *May follow a pattern used by many organizations, e.g., "Extreme programming", "Rational unified process (RUP)", "scrum", "waterfall", "spiral model"*

Styles & Instances

In cooking: North Italian > risotto > risotto ala Milanese > tonight's risotto ala Milanese with chicken and chantarelle

In airplanes: Jet airliner > wide body twin-engine > Dreamliner 787

In software processes

- Waterfall > Waterfall as practiced at XXX corp > This project
- Agile > Scrum > EA's Scrum > Zelda meets Godzilla
- Process family > process model > adapted process > project plan

Typical Goals

Intellectual manageability

Predictability

- ability to make a reasonably accurate plan

Visibility

- ability to monitor ("how are we doing?")

Flexibility, Feedback

- ability to acquire and adjust to new information and circumstances

Relative priority of these goals will vary by domain and organization

Process Models in Other Fields

Reliable, efficient production

- Process improvement for quality, efficiency

Predictable production

- Ability to plan, schedule, and budget production

Standardization

- Economic advantage of standard processes and components

Automation

The “Waterfall” model

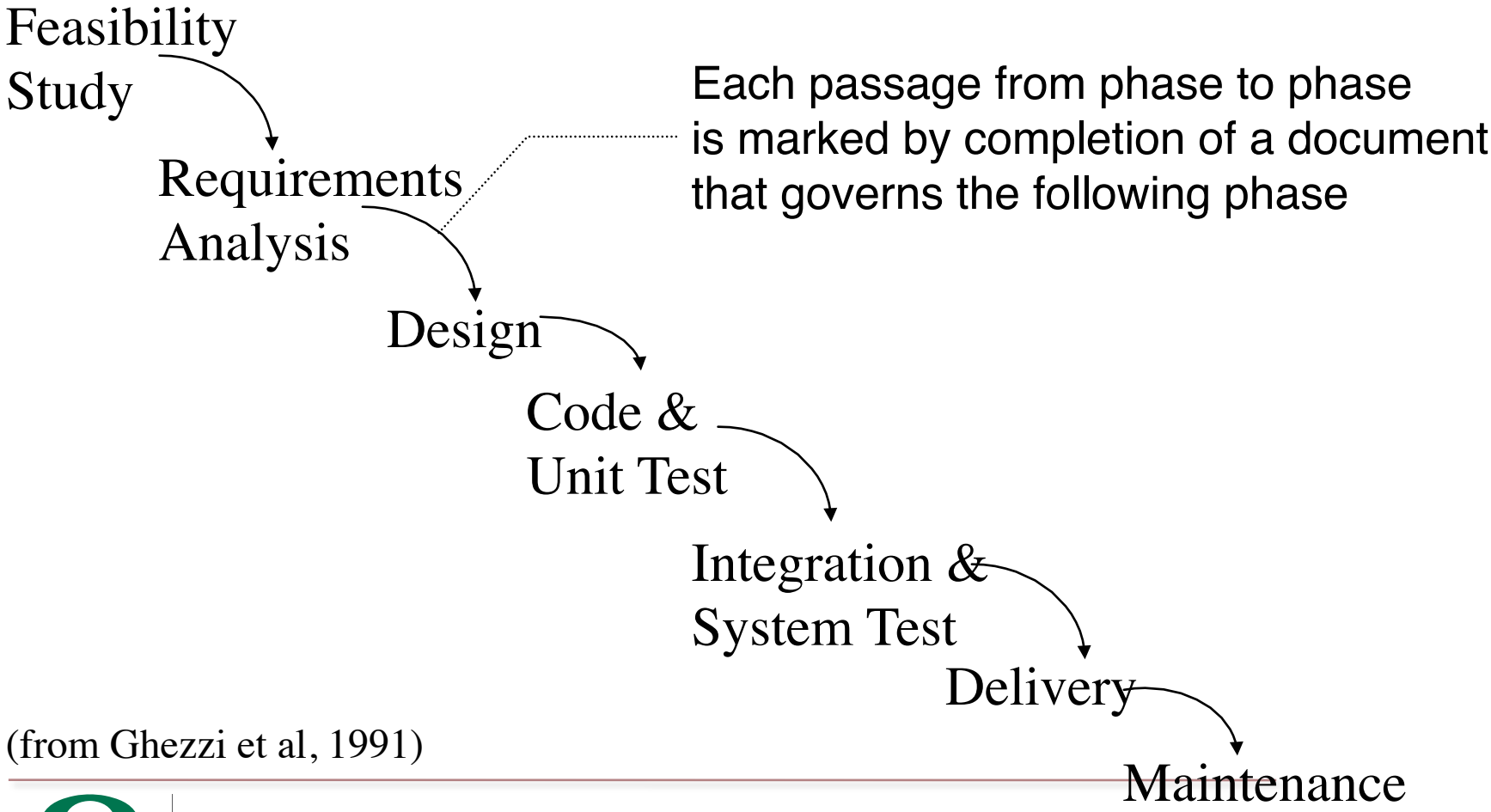
Inspired by industrial product development cycles, esp. aircraft

A document-based model

- Stages in development are marked by completion of documents
- Feedback and feed-forward are through documents

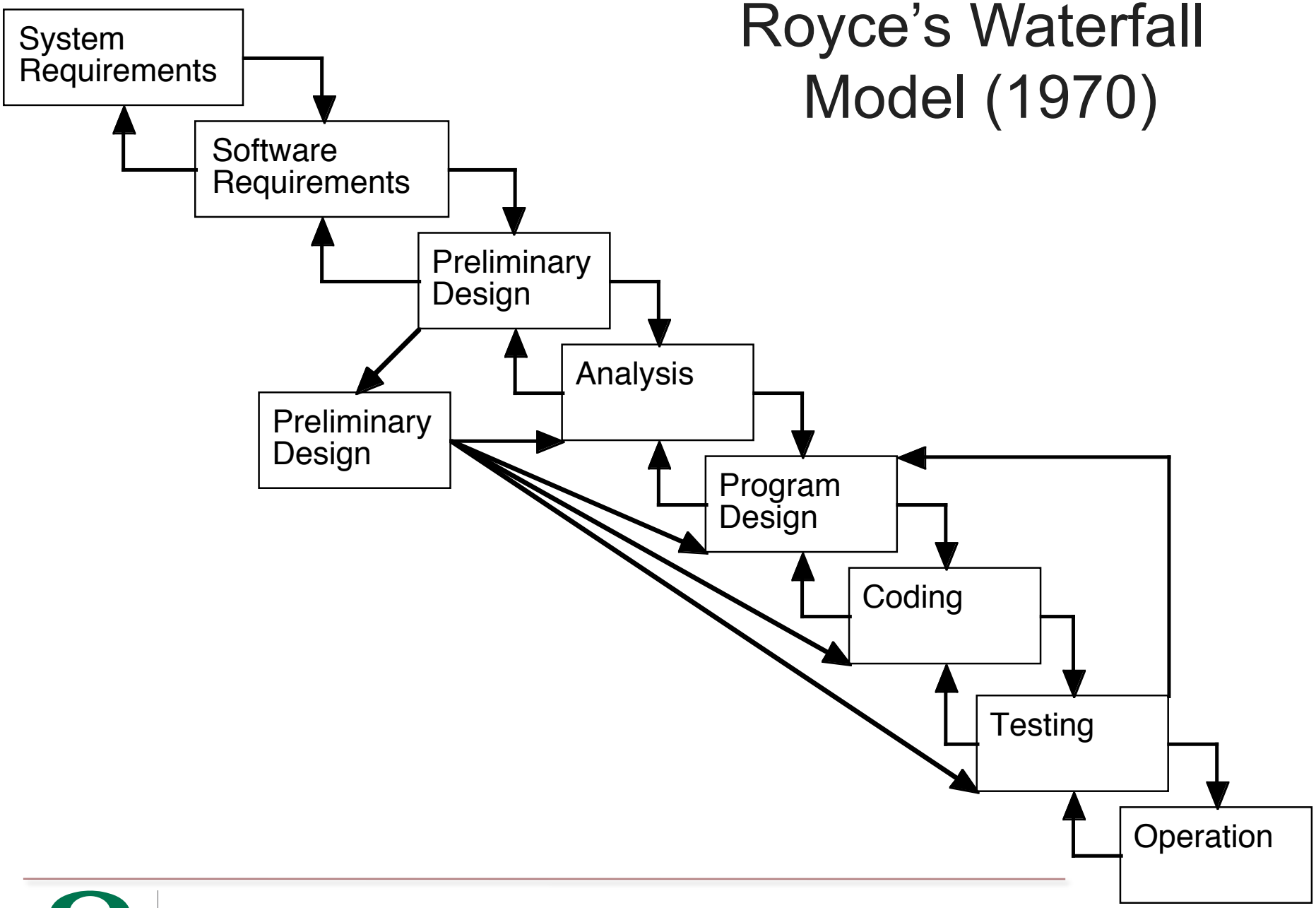
Several variations

Waterfall Model (example)

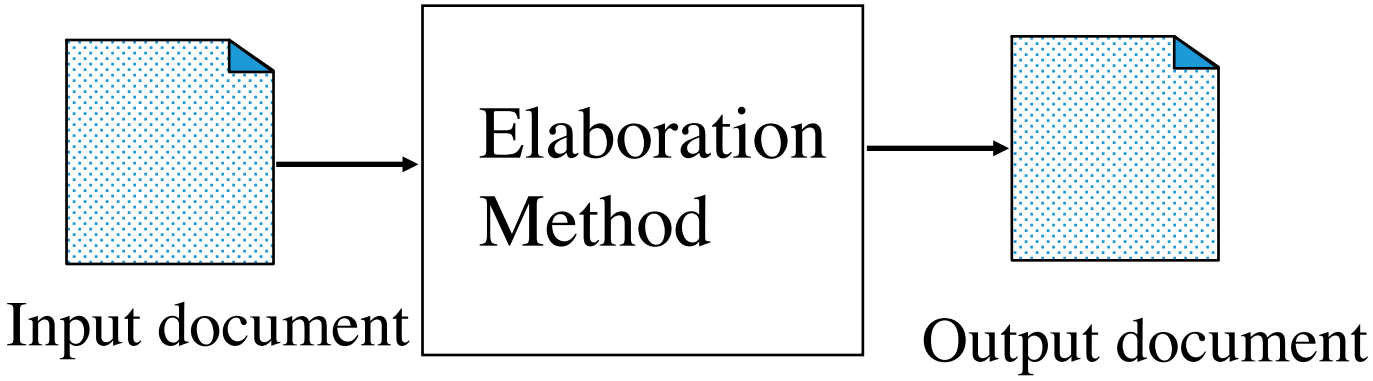


(from Ghezzi et al, 1991)

Royce's Waterfall Model (1970)



Waterfall Model Phase



Goal is an output document consistent with the input document; an "error" is an inconsistency

Phase is complete when document is finished

Each phase has specific methods

Characteristics of the Waterfall Model

- Limited iteration
  - Naive version is purely sequential; more commonly there is some iteration and adjustment, but the model is highly sequential
- "Big bang" development
  - Beginning from nothing
  - Ending with a single delivery of a single product

How does waterfall satisfy goals of a process model?

Intellectual manageability

Predictability

- ability to make a reasonably accurate plan

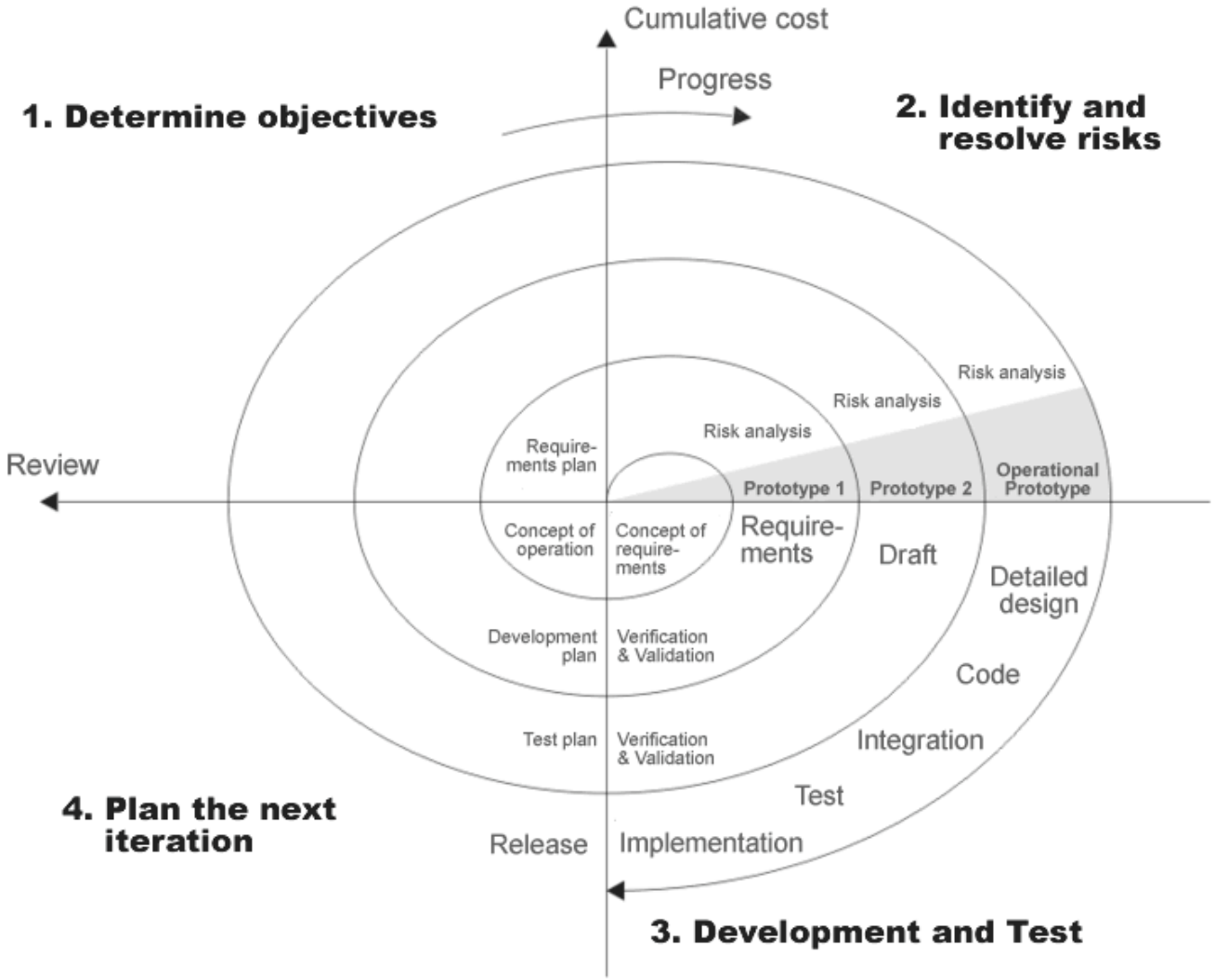
Visibility

- ability to monitor ("how are we doing?")

Flexibility, Feedback

- ability to acquire and adjust to new information and circumstances

Spiral Model
(Risk-driven evolutionary development)



In each “turn” of the spiral

- Problem definition
  - Determine objectives (qualities to achieve)
  - Identify alternatives and constraints
- Risk analysis
  - Determine risks
  - Gain information (typically through prototyping)
- Develop & verify next level “product”
  - may be only requirements, or design
- Plan next phase

Phased Projects

Develop & Deliver in Increments

- May repeat entire waterfall model in each increment

Goals:

- Keep clients/customers happy
- Improve requirements through feedback
- Improve process visibility through more frequent milestones

Prototypes vs. Incremental Deliveries

The primary goal of a prototype is *information*

- Should address the most significant risks

Incremental deliveries should be useful

- May avoid the highest risks

These goals are in conflict!

- It is sometimes possible to serve both purposes
- *but* ... Many "prototypes" fail to serve either purpose, because developers fail to distinguish goals and plan accordingly

Prototyping for Information

- Requirements clarification
  - Users “learn what they want” by using the prototype
  - Implicit requirements are identified through failure
  - Human interface can be assessed and refined
- Design alternatives
  - Performance, complexity, capacity, ...
  - Requires evaluation plan *before* implementation

"Agile methods"

A reaction to problems with "waterfall" methods: Same goals, different means

XP, Scrum, RAD, ...

Predictability, Visibility: Through incremental development

- Rapid feedback, continuous adjustment

Agile? Huh?

Agile:

> marked by ready ability to move with quick easy grace <an agile dancer>
>
> having a quick resourceful and adaptable character <an agile mind>

(Merriam-Webster)

As versus: perceived slow, clumsy movement of conventional software development processes

Cycle time, adaptability

Waterfall model: Freeze requirements early, then be consistent

- Boehm: "Plan the flight, fly the plan"
- Problem: "Now that I see it, that's not what I wanted"

Spiral, iterative: Multiple cycles of requirements, design, implementation

Agile: Radically shortened, with skipped steps

Plan vs Adapt (per Martin Fowler)

Classic engineering is based on planning

- Carpenter's rule: Measure twice, cut once (a good rule if you're cutting something physical)
- Change (new requirements, unanticipated difficulties) are a problem. Avoid it if you can.

Agile methods welcome change

- Resistance is futile. Don't try to predict, don't try to prevent, just adapt. Take one useful step, then plan the next.
- Assume competence and good will.

Code vs Design

Conventional view:

- Requirements and design are creative.
- Code is a fabrication activity. Train some monkeys to write it.

Agile view

- Code is design. It's creative and respectable.
- We have computers, not trained monkeys, for the fabrication step

Long before XP and Scrum ...

Rapid Application Development (IBM)

No written requirements: Build, demo, repeat

Intense client participation

- "Workshops" for goals and (especially) scope
- Client as collaborator: rapid cycle of choosing next step

Timeboxing

Small, flat teams, using standard frameworks

"Agile" process characteristics

- Very rapid build/evaluate/design cycle
  - Days or weeks; not months
- Requirements are minimal and informal
  - Typically "user stories" (scenarios)
  - Requirements are assumed to be incomplete and evolving: We don't know till we see it
- Little architectural design; lots of refactoring
  - Design is also evolving; commit "as late as possible"

Conventional	Agile
Documents record decisions (requirements, design)	Replace most documents with meetings
Plan carefully, design for change	Do something useful now. Refactor tomorrow. Don't anticipate or generalize.
A document (requirements, design, ...) marks progress	Progress is working code doing something useful. Nothing else counts.
Work products are reviewed (code reviews, design reviews, ...)	Pair programming.
Estimate schedule for planned features.	Select features for schedule.
Partition responsibility: I own this code, you own that code.	Joint ownership: Anybody can change anything.

Reduced Paper Documentation

- Emphasis on rapid delivery and change
  - Not on preserving information for a longer period
  - Fixed personnel (including user representatives) reduces need for documents as orientation and communication
  - Active, intense user participation
- Reliance on computerized documentation
  - CASE tools, databases and application generators
  - The test cases are design "documentation"
- Developer "logs" of design rationale

Practices: Developers are human

Stand-up meetings

Daily scrum: What did you do yesterday, what do you plan to do today, and is there anything in your way?

Pigs & chickens: Only pigs speak.

Limited overtime

Pair programming

Test first

Timeboxing

Only developers estimate effort

Timeboxing

If functionality not delivered by date, scale back or abandon

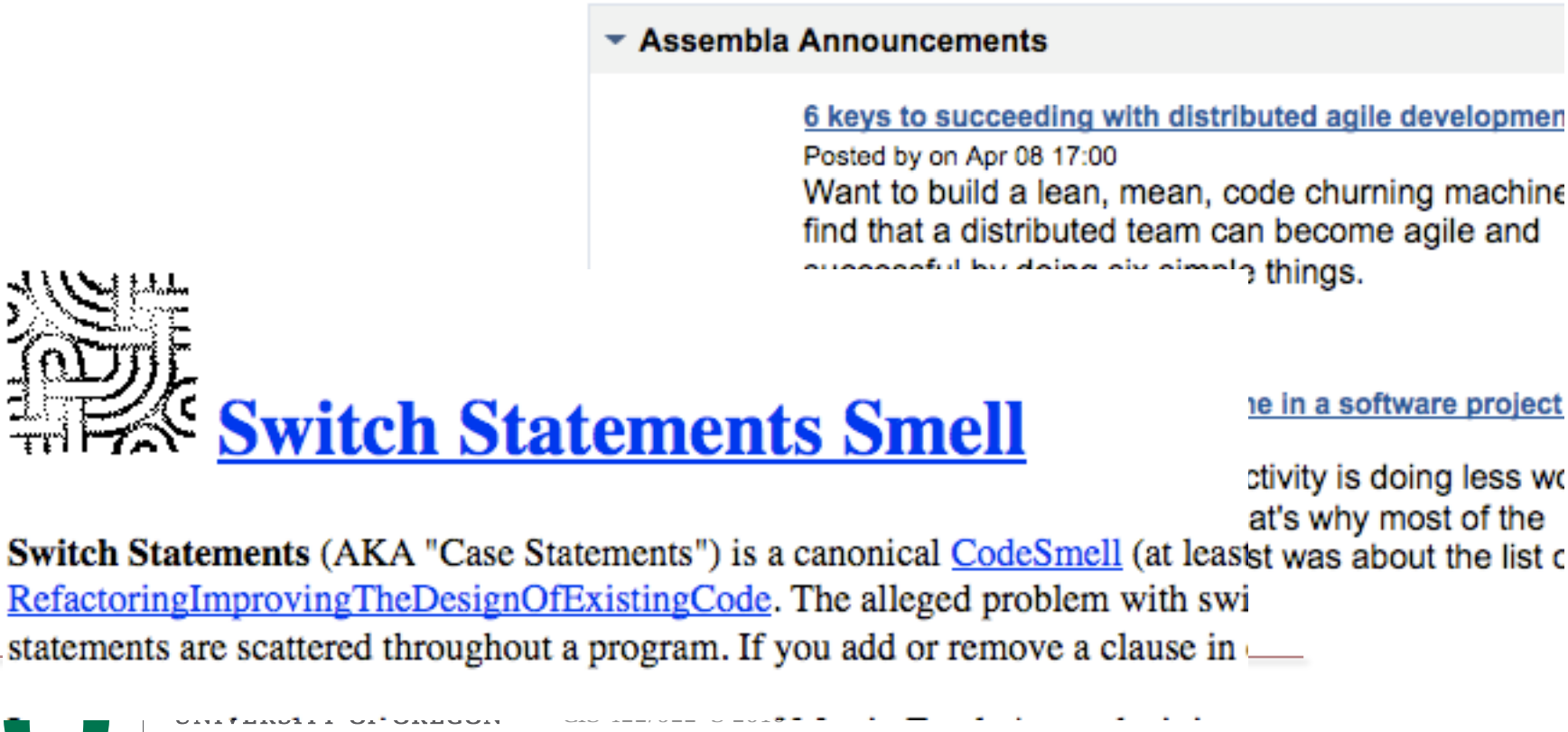
- Radical application of “design-to-schedule”

The build-plan is stable; the product functionality is fluid within bounds of project scope

- What is actually built depends on technical feasiblity as well as user wants

Community of Practice

Learn from experience, share experience; not a fixed process “by the book”



Agile vs. Just Hacking

It's easy to *just hack* and call it "agile"

- Agile development is adaptive but disciplined

Each process (XP, Scrum, Crystal) has well-defined rules and practices

- Irony? Lots of strict rules of practice, because we're humans and need discipline.

Next step is chosen by customer and developer together

Example: Scrum Process

Two cycles:

- 24 hours, from "daily scrum" meeting to next
- 2 weeks to 30 days: the "sprint"

Sprint results in delivered functionality (shippable)

- Something from the prioritized feature backlog
- Selected for importance, *and feasibility*
- "Burndown" chart is current time-to-completion estimate

Continuous Process Improvement

Retrospectives

- After each sprint: what did we do well, what can we improve
- Note analogy to Toyota processes: just-in-time, transparent, constant improvement. Agile and Toyota model are both reactions against *Taylorism*.

Goals (again)

Intellectual manageability

Predictability

- ability to make a reasonably accurate plan

Visibility

- ability to monitor ("how are we doing?")

Flexibility, Feedback

- ability to acquire and adjust to new information and circumstances

Relative priority of these goals will vary by domain and organization

Choosing a model, designing a process

What would you choose, and why?

- Context: Flight control software for Boeing Dreamliner
- Context: Spore
- Context: Amazon Kindle version x
- Context: Yahoo new advertising program (compete with Google AdWords)