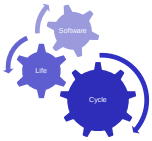
CSE403 ● Software engineering ● sp12

Week 1				
Monday	Tuesday	Wednesday	Thursday	Friday
• Overview • Course plans & expectations	• Tools & tool questions (section)	• Lifecycle & project milestones • KNOW project overview • *JB office hours 2-3PM Atrium*	• No section • *JB office hours 11:30AM-12:30 PM Atrium* • *DN office hours 9:00-10:00AM & 11:30-noon*	*Proposal descriptions & slides by 9:30AM* • Proposal presentations *Project & team preferences by 11PM* Teams announced by 11PM Saturday



Software development lifecycle
the power of process

KNOW
Knowledge Network of World Events & News

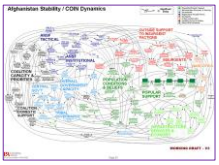
Two goals of software engineering
Barry Boehm

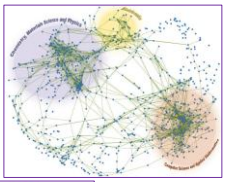
- Building the right system
  - *Validation*: Does the program meet the users' needs?
- Building the system right
  - *Verification*: Does the program meet the specification?

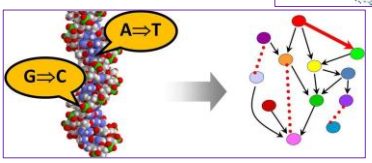
The software lifecycle

- These goals – software that works as specified and software that meets users' needs – are hard to achieve for substantive systems
- The lifecycle is a series of steps or phases through which software is produced – usually over months or years, from womb to tomb
- It is a process by which teams of people can create complex software systems
- The lifecycle helps teams deal with complexity by laying out a clear set of steps to perform and associate tangible artifacts that can be assessed to determine progress, quality, etc.

What is complexity?







"The state or quality of being intricate…"

Dictionary.com. *Collins English Dictionary - Complete & Unabridged 10th Edition*. HarperCollins (accessed: March 27, 2012).

Complexity in computation

- How much resource – usually time or space – is needed, based on the size of the input, to solve a specific problem using a precise model?
  - Lower bound: best possible
  - Upper bound: best known
- Kolmogorov complexity represents the shortest possible representation of a (often a program to compute) value
  - 2^{31542} vs. 4522134566.32323354252567888885322122323423423466661 93
- Fred Brooks: Essential vs. incidental complexity

How complex is software?
Possible measures include

- lines of code
- # classes
- # modules
- # module interconnections and dependencies
- # paths: cyclomatic complexity takes a control flow graph (roughly, a "flow chart") of a program and computes $E - N + 2P$
  - E = the number of edges of the graph
  - N = the number of nodes of the graph
  - P = the number of connected components (exit nodes)
- time to understand
- # of authors
- … many more

Lines of Code

- LOC (SLoC – *source lines of code*) are frequently used to characterize the size of a software system
  - Often used for cost estimation
- May be as good a proxy for complexity as anything else

MSLoC (MLOC) ≡ Million Lines of Source Code	
Windows Server 2003	50 MSLoC
Debian 5.0	324 MSLoC

Downside
- Paying people per LOC isn't smart
- If they refactor and make the code better but smaller, should they pay you?

How big is 324 MLOC?

Two minutes

Left side of the room: in small groups estimate how high 324 MLOC would be if you printed it
(assume 50 LOC/page, two-sided)

~13,000 inches ≈ [illegible] of the Allen Center

Ideas about how to think about this?

Right side of the room: in small groups estimate how long it would take you to type 324 MLOC
(assume 5 words/LOC @ 50 wpm)

~32,000,000 min ≈ 61 years
no thinking • no sleeping • no breaks

How to get to 324 MLOC?

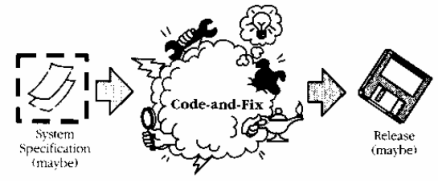
- …or even 1MLOC … or even 100KLOC …?
- Especially adding in some expectations of what the program does, its correctness, etc.

Ad-hoc development

Creating software without any formal guidelines or process; ever done this for a project or assignment?

- Advantages
  - Easy to learn
  - Easy to use
- Disadvantages
  - May ignore some important tasks like testing and design
  - Unclear when to start or stop each task
  - Scales poorly to teams
  - Hard to review/evaluate work
  - Code may not match users' needs – no requirements!
  - Code likely to be inflexible
  - No way to assess progress, quality or risks
  - Unlikely to accommodate changes without a major design overhaul
  - Unclear delivery features (scope), timing, and support
  - …

Also know as code-and-fix



- It doesn't manage or tame complexity
- And the later a problem is found in software, the more costly it is to fix

Managing complexity: break it down

- We identify different *activities* that we know – from experience – we will have to do (such as testing)
- We identify different *milestones* that we know – from experience – we will have to produce (such as requirements)
- We identify different *roles* – from experience – that people will perform in a project (such as designers)
- We identify a *process* that increases our confidence that people in those roles will address all the activities effectively and produce a quality set of milestones
- The specifics of these dimensions – people, activities, milestones – and the way they interact characterizes different lifecycles

Parts of speech

Milestones are also points in time – deadlines – but they are accompanied by deliverables

Subject *Role*	Verb *Activity*	Object *Milestone*
A designer	designs	a design
A programmer	implements	modules
A tester	plans and runs	tests
…		

- These are largely different sides of the same (three-sided) coin
- It's never 1:1:1
  - An individual may be a programmer and a tester
  - Multiple milestones will include designs
  - …

Benefits of using a lifecycle

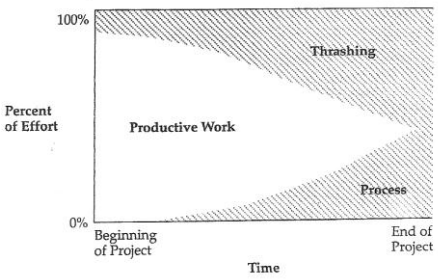
- It provides a structure for organizing work
- It forces us to think of the "big picture" and follow steps so that we reach it without glaring deficiencies
- Without it you may make decisions that are individually on target but collectively misdirected
- It is a management tool

Drawbacks?

Extra credit

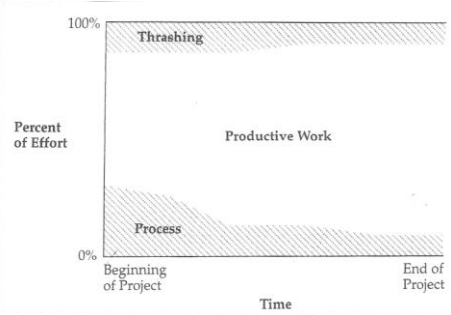
"…I have always found that plans are useless, but planning is indispensable." –D.D. Eisenhower

Project with little attention to process



Survival Guide: McConnell p. 24

With early attention to process

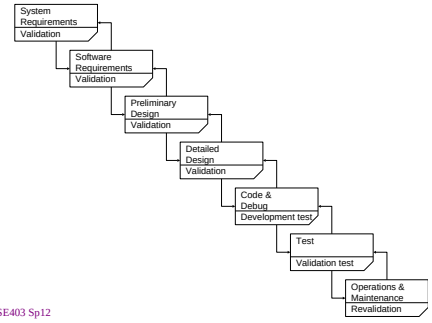


Survival Guide: McConnell p. 25

Some lifecycle models (past code-and-fix)

- waterfall: standard phases – requirements, design, code, test, … – in order
- spiral: assess risks at each step; do most critical action first
- staged delivery: build initial requirement specifications for several releases, then design-and-code each in sequence
- evolutionary prototyping: build an initial small requirement specification, code it, then "evolve" the specification and code as needed
- agile: very flexible, customer-oriented variations of evolutionary prototyping (more coming next week)

Waterfall model

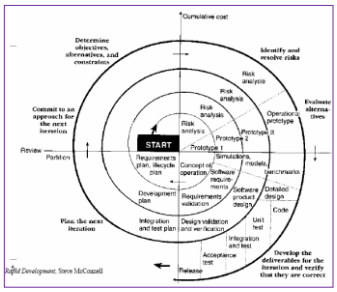


Waterfall model tradeoffs

- Can work well for well-understood but complex projects
  - Tackles all planning upfront
- Supports inexperienced teams
  - Orderly, easy-to-follow sequential model
  - Reviews at each stage determine if the product is ready to advance
- Hard to specify requirements of a stage completely and correctly upfront
- Rigid, linear, not adaptable to change in the product
  - Costly to "swim upstream"
- No sense of progress until end
  - Nothing to show until almost done ("we're 90% done, I swear!")
- Integration occurs at the very end
  - Defies "integrate early and often" rule
  - No feedback until end to customer

Spiral model – risk-oriented

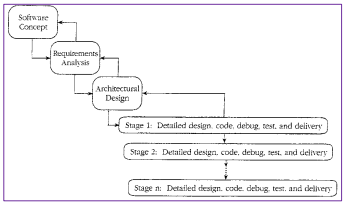
- Determine objectives and constraints
- Identify and resolve risks
- Evaluate options to resolve risks
- Developer and verify deliverables
- Plan next spiral
- Commit (or not) to next spiral



Spiral model – tradeoffs

- A lot of planning and management
- Frequent changes of task
- Requires customer and contract flexibility
- Must be able to assess risk properly
- Take on the big risks early, make decisions
  - Right product?
  - Can we implement?
- Progresses carefully to a result – clearer tasks each spiral
- As costs increase, risks decrease!

Staged delivery model



- Waterfall-like beginnings
- Then, short release cycles: plan, design, execute, test, release, with delivery possible at the end of any cycle

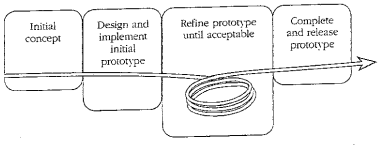
Staged-delivery tradeoffs

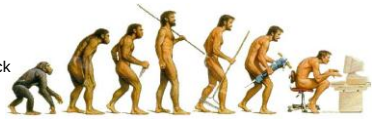
- Can ship at the end of any release cycle
- Intermediate deliveries show progress, satisfy customers, and lead to feedback
- Problems are visible early (e.g., integration)
- Facilitates shorter, more predictable release cycles
- Prioritize features
- Requires tight coordination with documentation, management, marketing
- Product must be decomposable
- Extra releases cause overhead
- Feature creep

Very practical, widely used and successful

Evolutionary prototyping model

- Develop a skeleton system and evolve it for delivery
- Staged delivery: requirements are known ahead of time
- Evolutionary: discovered by customer feedback on each release





Evolutionary process

- Requires close customer involvement
- Assumes user's initial specification is flexible
- Problems with planning
  - Feature creep, major design decisions, use of time, etc.
  - Hard to estimate completion schedule or feature set
  - Unclear how many iterations will be needed to finish
- Integration problems
- Temporary fixes become permanent constraints

Why are there so many models?

- The choice of a model depends on the project circumstances and requirements
- A good choice of a model can result in a vastly more productive environment than a bad choice
- A cocktail of models is frequently used in practice to get the best of all worlds – models are often combined or tailored to environment
- "Models" are as often descriptive as they are prescriptive
  - Parnas and Clements. A rational design process: How and why to fake it. *IEEE Trans. Software Eng. 12*, 2 (Feb 1986), 251-257.

The "best" model depends on…

- The task at hand
- Risk management
- Quality / cost control
- Predictability
- Visibility of progress
- Customer involvement and feedback
- Team experience
- …

Better question: best model for …

- A system to control anti-lock braking in a car?
- A hospital accounting system that replaces an existing system?
- An interactive system that allows airline passengers to quickly find replacement flight times (for missed or bumped reservations) from terminals installed at airports?
- A specific 403 project?

Today	Tomorrow	Wednesday	Thursday	Friday
• Overview • Course plans & expectations	• Tools & tool questions (section)	• Lifecycle & project milestones • KNOW project overview • **Form project proposal groups NOW** (if you haven't yet) • **KNOW++: JB office hours**	• No section • **Meet with your project proposal groups** • **KNOW++: JB office hours** • **DN office hours 9-10 & 11:30-12**	**Proposal descriptions & slides by 9:30AM** **Posted on web ASAP** • Proposal presentations **Project & team preferences by 11PM** Teams announced by 11PM Saturday

Any questions?