CS1530, Lectures 3 - 4:
Software Development Methodologies

Definition

- Software Engineering Methodology: A system of dividing software development into distinct phases, responsibilities, and deliverables.

- There are many ways to do this!

- This allows better planning, management, and (arguably) better software.

Why Not Just Throw Coders At a Problem And Hope For The Best?

- Remember there are other aspects of development besides coding, and there needs to be communication with/from, and preparation for, these other aspects

- External constraints (e.g. deadlines, legislative or regulatory)

- Need for planning and estimation

- Interactions with others

- Guidelines for development (avoiding the "so... now what?" problem)

We've discovered a few things in software development; this is one!

- Projects get accomplished with higher quality and less surprise if the team follows some sort of structured methodology, as compared to none

- Now, what methodology to use, that is the question!

  - Domain-dependent

  - Team-dependent

  - Project-dependent

  - Schedule-dependent

Heavyweight vs Lightweight Methodologies

- One of the key dividing lines

- Lightweight = fewer rules, more flexibility, less documentation, less up-front work, less restrictive

- Heavyweight = more rules, less flexibility, more documentation, more up-front work, more restrictive

-

Heavyweight vs Lightweight

- Lightweight = pick-up game with your friends
- Heavyweight = Olympic sports
- Which is "better"? Depends on your goals!

Cowboy Coding

- The lightest of lightweight "processes"
- Lack of any sort of formal (or even informal) software project management
- Lack of estimation, testing, integration
- Think "hackathon" or "assignment due in two hours" work

Cowboy Coding

- You'd expect a Software Engineering professor to rail against this - and it IS overused - but it does have some benefits!

  - Fast

  - Flexible

  - Efficient for very small teams or projects

  - Efficient if correctness is not critical

Cowboy Coding

- Many, many drawbacks:
    - Tends to use "quick and dirty" solutions, a.k.a. kluges or hacks
    - Often leads to technical debt
    - Not good for writing code without defects
    - Very difficult to write non-trivial software systems in this style
    - Very difficult to converse with other stakeholders about system

The Culmination of the Manichaean Worldview in Software: Waterfall

- The closest to the "idealized" SDLC covered in the last class

- Understand and document all customer requirements first, then understand and document entire design, then understand and document all code before testing, etc.

- Stability before moving on, never moving back

Waterfall

- Very "wall-friendly" - easy to throw design "over the wall" to developers, or code "over the wall" to testers

- Why? Heavy documentation, software phase should be "complete" before moving on

- Communications can then be at a minimum

- This can be good - what if team is geographically disparate or very large?

- Helps eliminate the $O(n^2)$ growth of comm pathways

Waterfall

- "Gates" between development phases

- Ensure correctness as much as possible before moving on

- Not meant to be flexible - you have specific goals you are trying to reach

- Change is minimized, but lots of work is done ahead to time when change should be "easier"

  – BDUF ("Big design up front")

Benefits of Waterfall

- Often maligned, but it (or a variant like Incremental Development) actually can be good fit if requirements are stable and well-understood, and if developers understand the system and domain well

- I view it as a lumbering ogre - good if you want simple, well-understood tasks done

- Creates lots of documentation for others to be able to come on to team

Drawbacks of Waterfall

- Very inflexible

- If problem domain is complex or not well-understood, can end up at the end with entirely the wrong product, or non-functional software

- Extremely slow

- Prone to failure - think "Christmas lights in sequence" as opposed to in parallel

Spiral

Determine Objectives

Identify / Resolve Risks

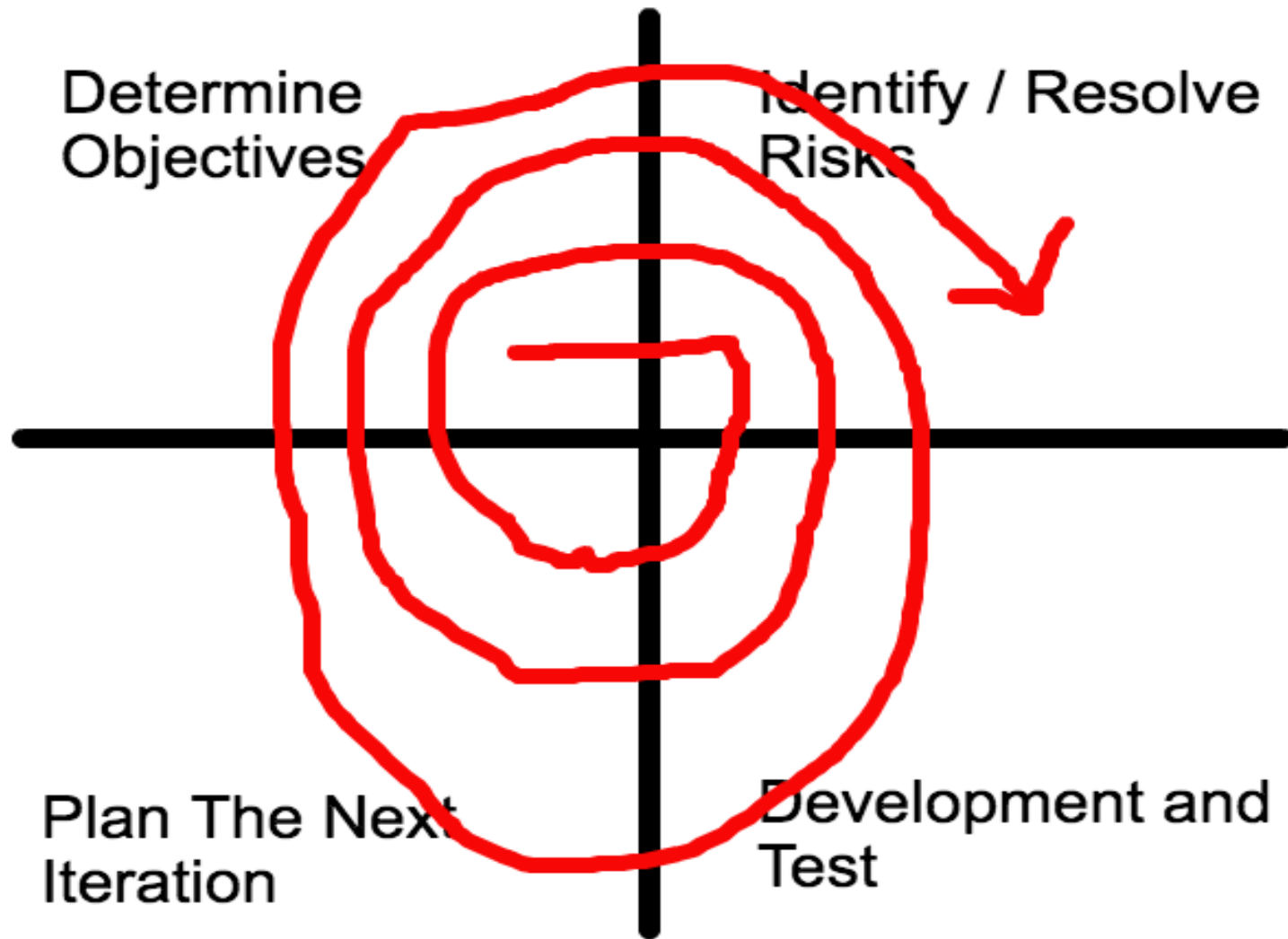Plan The Next Iteration

Development and Test

Spiral

Determine Objectives

Identify / Resolve Risks

Plan The Next Iteration

Development and Test

Benefits of Spiral Metholodogy

- Allows more precise estimation, albeit closer temporally

- More flexible than agile

- Allow focus on risk and trade-offs multiple times in the development process

Drawbacks of Spiral Metholodogy

- Still limits us in flexibility

- Not as detailed as some other methodologies; Barry Boehm described it as a meta-process which could be used for specific methodologies

  - In fact, most methodologies discussed in class are "special cases" of spiral development

- Focus is on risk reduction as opposed to feature development

Cleanroom Development

- Software Development based on formal methods (mathematical description of a system before coding)

- If a system fails QA, defects are not just fixed; system is moved back to design phase!

- Testing is done statistically, not haphazardly

- Not to be confused with "cleanroom" as in developing a workalike system without access to the underlying code (e.g. BSD)

Benefits of Cleanroom Development

- Allows us to get close (never quite reaching, of course) defect-free software

- Confidence in software

- Reductions in cost of development compared to similar systems, according to the SEI

Drawbacks of Cleanroom Development

- Not as flexible - you need to understand a system well to mathematically model it

- Lack of people with requisite skills in industry

- If not done properly (and see last point), can be slow

Iterative and Incremental

- Iterative - System is developed through repeated cycles

- Incremental - Small pieces of the software are done during each iteration

- Always have working software, just keep adding features

Kinds of Iterative and Incremental

- Unified Process
    - and its variants RUP, OpenUP, etc.

- Chaos model

- Various "incremental Waterfall" models

- XP

- "Agile" methodologies can be considered variants

Benefits of Iterative and Incremental

- Very flexible!

- Always have working software; can easily reduce scope by just stopping work

- Focus is on small pieces of the software and getting each part to work correctly - less risk of catastrophic failure

Drawbacks of Iterative and Incremental

- Less up-front design and analysis; could mean more technical debt later

- Difficult for products which are hard to split into feature sets

- Can be difficult to plan ahead

RAD (Rapid Application Development)

- Less emphasis on planning, less emphasis on design/architecture

- "Many projects fail because they did not meet the user's needs, and users have trouble verbalizing their needs, so let's focus on the UI and having a prototype first"

- Sometimes called "prototyping methodology"

- Prototypes get better - can evolve to the final product!

Benefits of RAD

- Higher quality software which more closely aligns with the user's needs

- Very, very flexible!

- Risk control - allows development team to focus on "what matters most", cut features which won't be necessary, etc.

Drawbacks of RAD

- Poor design - as "prototypes" evolve into production systems, they can often degenerate into the "Big Ball of Mud" architecture anti-pattern

- Rarely works well for large, complex, or interdependent systems

- Focus is on user-facing systems, not back-end

Agile

- A broad term encompassing several different, but related, methodologies

- Focus is on communication - dev to dev, dev to customer, manager to customer, QA to dev..

- "Information radiators" - information on a system should be dynamic, not static documentation

- Goal is to meet the needs of the customer as opposed to meeting requirements

Agile

- Assumption is that software development is a "never-ending" process

- Features can be added or removed, defects fixed, etc.

- Relatively lightweight processes - focus is on development of the software, with enough process to "point people in the right direction"

Benefits of Agile Development

- Very flexible, very good for work on tight timelines without rigid deadlines

- Kaizen - "continuous improvement" baked into the system for the most part

- Focus on user means all stakeholders focus on end goal

Drawbacks of Agile Development

- Less up-front understanding of system, so development team can go down blind alleys or accumulate technical debt

- More heavyweight than some, although still very lightweight

- Adaptive vs prescriptive schedule estimation (often a problem when dealing with management)

CMMI

- Capability Maturity Model Integration (evolved from CMM, which was software-only)

- Developed at CMU! Now run by a spin-off company...

- NOT a software methodology, but a way to determine how well a software development group is aligned with the process

- That is, for whatever process you follow, how well do you follow it?

CMMI Levels

- Initial - Processes are unpredictable, ad hoc (where you start)

- Managed - Processes is defined and characterized for projects. The process is at least documented.

- Defined - Processes is defined and characterized for the entire organization.

- Quantitatively Defined - Processes are measured and controlled. There are metrics that can allow one to know how well the processes are working.

- Optimizing - Focus is on process improvement

Our Methodology: Agile / Scrum

- Perhaps the most popular version of "Agile development"

- We will see some of its drawbacks, benefits, and how it works next time!