
[CS3704] Software Engineering

Dr. Chris Brown
Virginia Tech
8/30/2023

Announcements

- **Homework 1 due Friday (9/8) at 11:59pm**
 - Canvas questions (Upload file or input text)

Software Process II

Unified Process
Agile Methodologies
SE Processes Revisited

Learning Outcomes

By the end of the course, students should be able to:

- **Understand software engineering processes, methods, and tools used in the software development life cycle (SDLC)**
- Use techniques and processes to create and analyze requirements for an application
- Use techniques and processes to design a software system
- Identify processes, methods, and tools related to phases of the SDLC
- **Explain the differences between software engineering processes**
- Discuss research questions and current topics related to software engineering
- Create and communicate about the requirements and design of a software application

What is a SE Process?

- Definitions:
 - a framework for the tasks that are required to build high-quality software to provide stability, control and organization to an otherwise chaotic activity [Pressman]
 - a software development process defines *who* does *what*, *when*, in order to build a piece of software. [Wilson]

Recap

Plan-Driven Models

1. Code-and-Fix*
2. Waterfall
3. V-model

Iterative Models

4. Prototyping
5. Spiral
6. Incremental

- *Agile (this class...)*

Warm-Up: Discuss

- 1. What is the difference between plan-driven and iterative software engineering processes?*
- 2. What do you know about agile development?*

Plan-Driven Models

- Traditional software process models
- Project goes through phases sequentially
 - Possible for feedback across phases (i.e. design problems can be fixed during coding)
- Project requirements are set up front and stable
- Typically few or no iterations
 - Project is “frozen” after a certain time, no changes
- *“Do it right the first time”...*

Iterative Models

- Modern software process models
- Software is developed through repeated cycles (*iterations*).
 - Easier to modify software design, functionality, etc.
 - Faster operational product (weeks vs months)
- Usually more user involvement

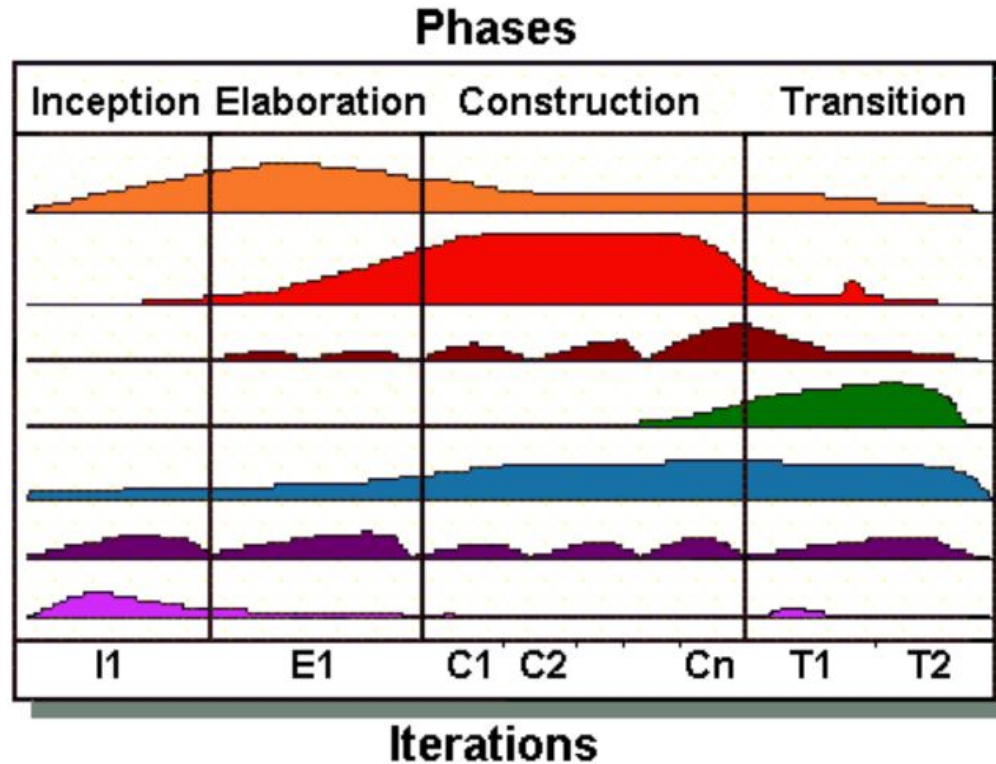
Iterations

- Iterations should be short (2-6 weeks)
 - Small steps, rapid feedback and adaptation
 - Massive teams with lots of communication
- Iterations should be time-boxed (fixed length)
 - Integrate, test and deliver the system by a scheduled date
 - If not possible: move tasks to the next iteration
 - Improves programmer productivity with deadlines
 - Encourages prioritization and decisiveness

Unified Process (UP)

- An example of iterative process for building object-oriented systems
 - Very popular
 - By the same folks who develop UML
- It provides a context for our discussions on analysis and design this semester
- Precursor to agile

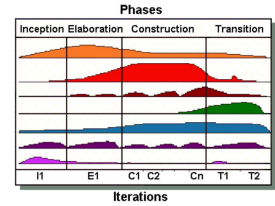
Unified Process



A little history

- “The three amigos”: Grady Booch, Ivar Jacobson, James Rumbaugh
 - **Early 90s**: Separated methodologies for object-oriented analysis and design (OOAD)
 - **1996**: Created the Unified Modeling Language (UML)
 - **1999**: Defined the Unified Process (UP) in Rational Software Inc.
- Refinement: Rational Unified Process (RUP)
 - Adaptable process framework + tools

UP Phases



- **Inception:** preliminary investigation
- **Elaboration:** analysis, design, and some coding
- **Construction:** more coding and testing
- **Transition:** beta tests and development
- Each phase may be enacted in an iterative way, and the whole set of phases may be enacted incrementally

Process Artifacts

- Discipline: an activity and related artifact(s)
- Artifact: any kind of work product
- We will focus on artifacts related to two disciplines:
 - **Requirement modeling**
 - requirement analysis + use-case models, domain models, and specs.
 - **Design**
 - design + design models


UP: Inception Phase

- Investigate approximate, business case, scope, and vague estimates
 - Should we even bother?
- Some basic analysis to decide whether to continue or stop
- Inception is NOT “requirements” in waterfall


UP: Elaboration Phase

- Most requirement analysis **Rq**
- Most design **Ds**
- Some coding and testing **Im** **Ut** **St**
 - Implementation and testing for core architecture and high-risk requirements
- Deeper investigation of scope, risks, and estimates
- Work products
 - Requirement models (UML use cases)
 - An architectural description
 - A development plan

UP: Construction Phase

- More coding and testing 
 - Implementation and testing for the remaining lower risk and easier elements
 - Integration
- Work products ready for delivery
 - A working software system
 - Associated documentation

UP: Transition Phase

- Beta tests and deployment 
 - Moving the system from the development community to the user community
 - This is important but ignored in most software process models
- Work products
 - A documented software system that is working correctly in its operational environment

Roots of Agile

- Direct response to waterfall and plan-driven processes
 - Requirements will change
 - Initial design will be inaccurate
 - Implementation will need to be flexible
 - Risks are inevitable
 - Desire for more *lightweight* approaches (planning, etc.)

Roots of Agile (cont.)

- ***Remember:*** plan-driven methods work great for government, lawyers, managers, etc.
- Agile focuses on developers: the talents and skills of *individuals*, needs of the *team*.
 - Human factors, “People factors” [Pressman]

“Agile development practices are almost as old as programming, but they came into their own with the rise of the web in the late 1990s.” [Wilson]

Data by the Standish Group (1995)

- \$81B on canceled software projects
- \$59B for budget overruns
- Only 1/6 projects were completed on time and within budget
- Nearly 1/3 projects were canceled
- Over half projects were considered “challenged”
- Among canceled and challenged projects
 - Budget overrun: 189% of original estimate
 - Time overrun: 222% of original estimate
 - Only 61% of the originally specified features

The CHAOS Report 1995

“In the United States, we spend more than \$250 billion each year on IT application development...A great many of these projects will fail. Software development projects are in chaos, and we can no longer imitate the three monkeys -- hear no failures, see no failures, speak no failures. The Standish Group research shows a staggering 31.1% of projects will be canceled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates. The cost of these failures and overruns are just the tip of the proverbial iceberg.”

[Standish Group]

The CHAOS Report 1995 (cont.)

Top 3 reasons for project failure:

Project Challenged Factors	% of Responses
1. Lack of User Input	12.8%
2. Incomplete Requirements & Specifications	12.3%
3. Changing Requirements & Specifications	11.8%

Top 3 reasons for project success:

Project Success Factors	% of Responses
1. User Involvement	15.9%
2. Executive Management Support	13.9%
3. Clear Statement of Requirements	13.0%

[Standish group 1995]

The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

- That is, while there is value in the items on the right, we value the items on the left more.

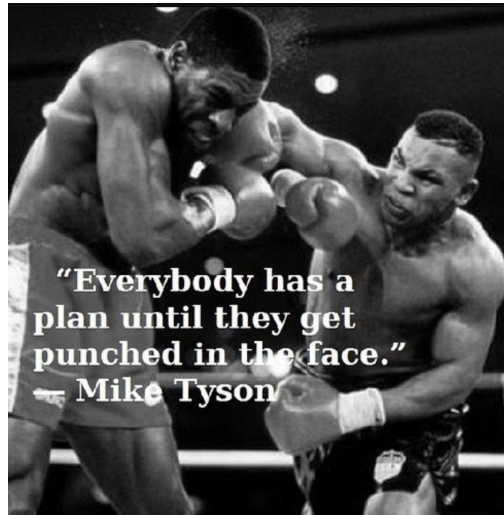
[Kent Beck et al, 2001]

What is Agile?

- Time-boxed, iterative, and evolutionary development framework that promotes:
 - adaptive planning
 - evolutionary development
 - incremental delivery
 - rapid and flexible response to change
- ➔ ***Any iterative process can be applied with an agile spirit!***

Key Points of Agile Planning

“Agile methods derive much of their agility relying on tacit knowledge embodied in the team, rather than writing the knowledge down in plans.” [Boehm]



Agile Planning (cont.)

- The purpose of planning is to understand, not to document.
- Keep is simple
 - Small set of disciplines
 - Avoid unnecessary artifacts
 - Only high-level planning for project
- Plans are inaccurate
 - Only *tested code* demonstrates and verifies good design!

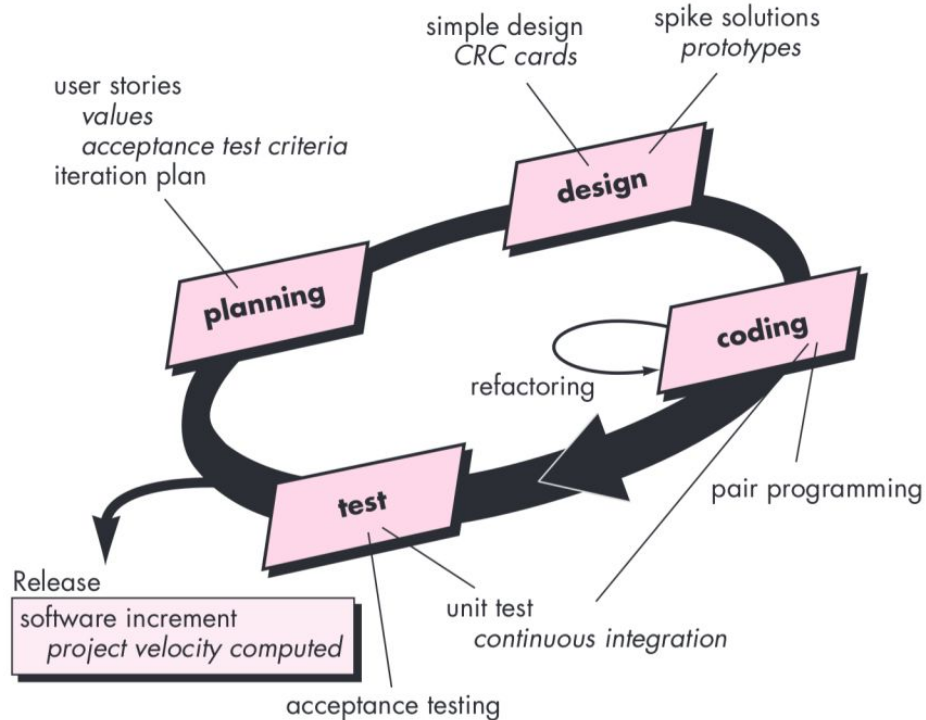
Agile Modeling

- Models are inaccurate
 - ***Only tested code demonstrates true design!***
- Diagrams/models are typically thrown away, if used at all.
 - Used to facilitate creativity
- Modeling should focus on the smaller percentage of unusual, difficult, and tricky parts of the design
 - Developers should do design modeling in pairs (or triads) and in parallel
 - Use simple tools and notation

Agile Development Processes

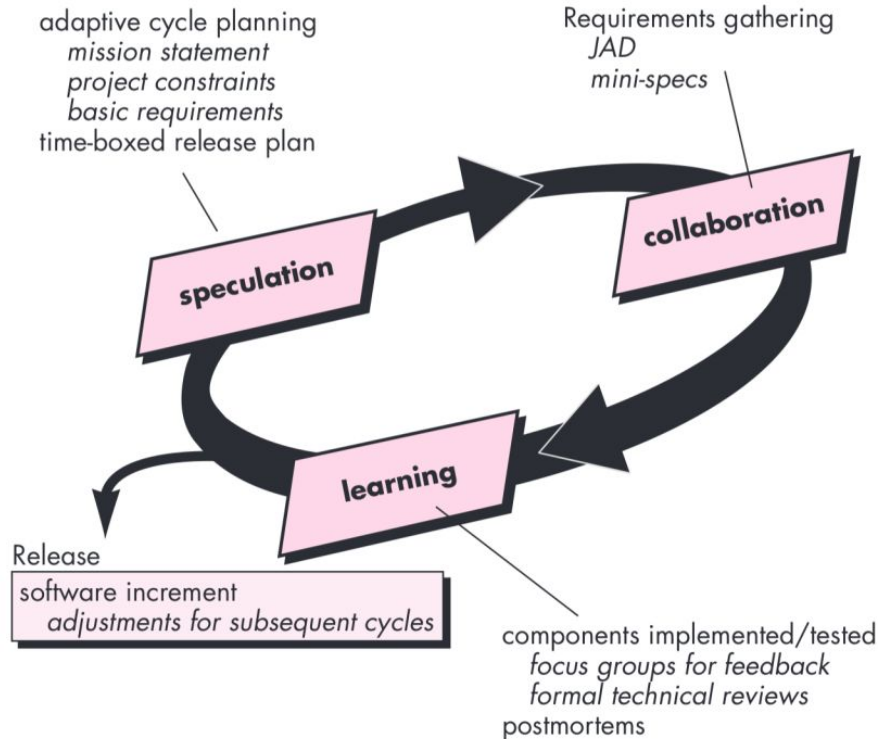
- Agile itself is ***not*** a software engineering process, but has led to the creation of several processes...
 - Extreme Programming
 - Adaptive Software Development
 - ...
- In addition to contributing other software development strategies and frameworks.
 - Kanban & Scrum
 - Lean Software Development
 - Crystal Agile Framework

Extreme Programming (XP)



Adaptive Software Development (ASD)

* More focus on human collaboration, team organization.



[Pressman]

Kanban and Scrum

- Two different strategies for implementing agile development practices
 - Can be combined with other processes, strategies, etc. (i.e. *Scrumban*)
- Brief introduction in SE Basics Workshop

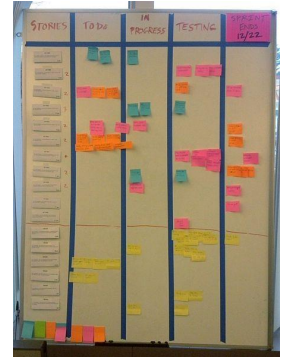
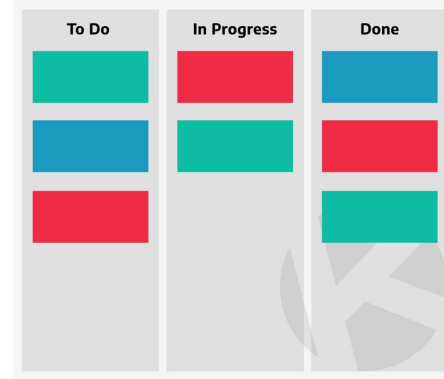


Japanese for *Sign* or *Billboard*

Kanban

Key Kanban concepts:

- Visualize workflow
 - Kanban boards
- Prioritize work
 - Limit work In Progress
 - Measure and manage flow
- Cards
 - Tasks to be assigned and completed during an iteration



Scrum

Key Scrum process flow

- Start

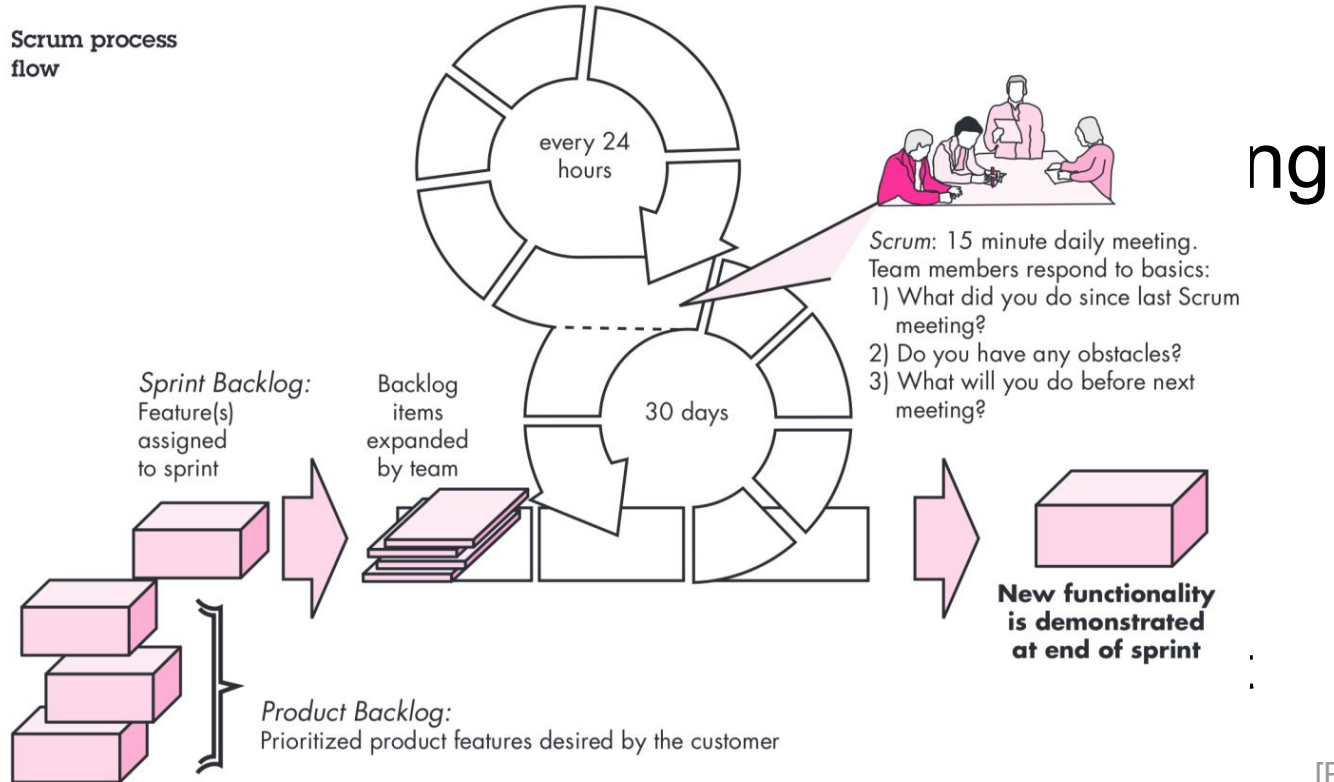
- A
- I

- Sprint

- It

- Backlog

- F



Scrum Meeting

TODO: Find 1 or 2 other students in class to complete another stand-up meeting!

- **What I did.**
- **What I need to do next.**
- **What is blocking me.**

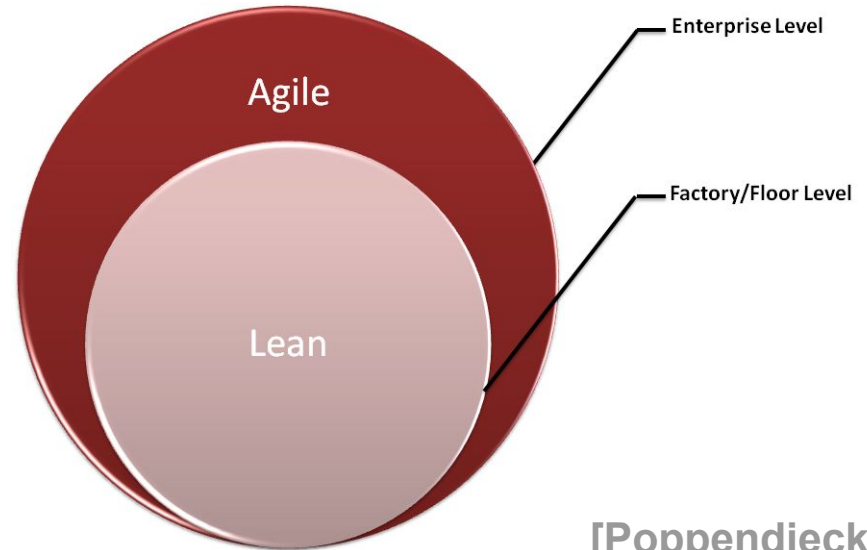
** Share since last stand-up (Basics Workshop), standing optional*

Lean Software Development

- Agile-based framework focused on optimizing development time and resources

7 Principles

1. Eliminate Waste
2. Build Quality In
3. Create Knowledge
4. Defer Commitment
5. Deliver Fast
6. Respect People
7. Optimize the Whole



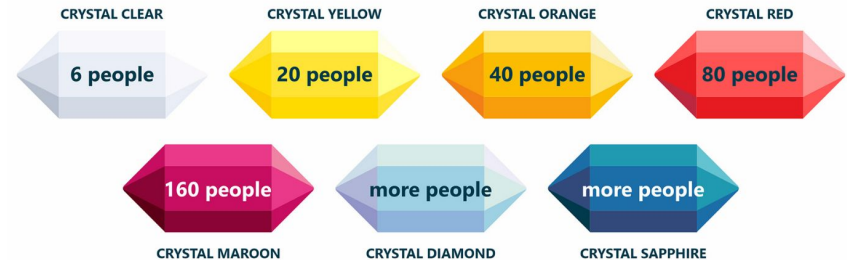
[Poppendieck]

Crystal Agile Framework

- Human-powered, adaptive, and ultra-light (little to no documentation) agile framework

7 Principles

1. Frequent Delivery
2. Reflective Improvement
3. Osmotic Communication
4. Personal Safety
5. Focus (on Work)
6. Easy Access to Expert Users
7. Technical Environment



[Cockburn]

Agile Pros and Cons

Pros:

- Customer satisfaction by rapid, continuous delivery of useful software
- Close, daily cooperation between business people and developers
- Better software quality and lower cost

Cons:

- People may lose sight of the big picture
- Heavy client participation is required

Caution!



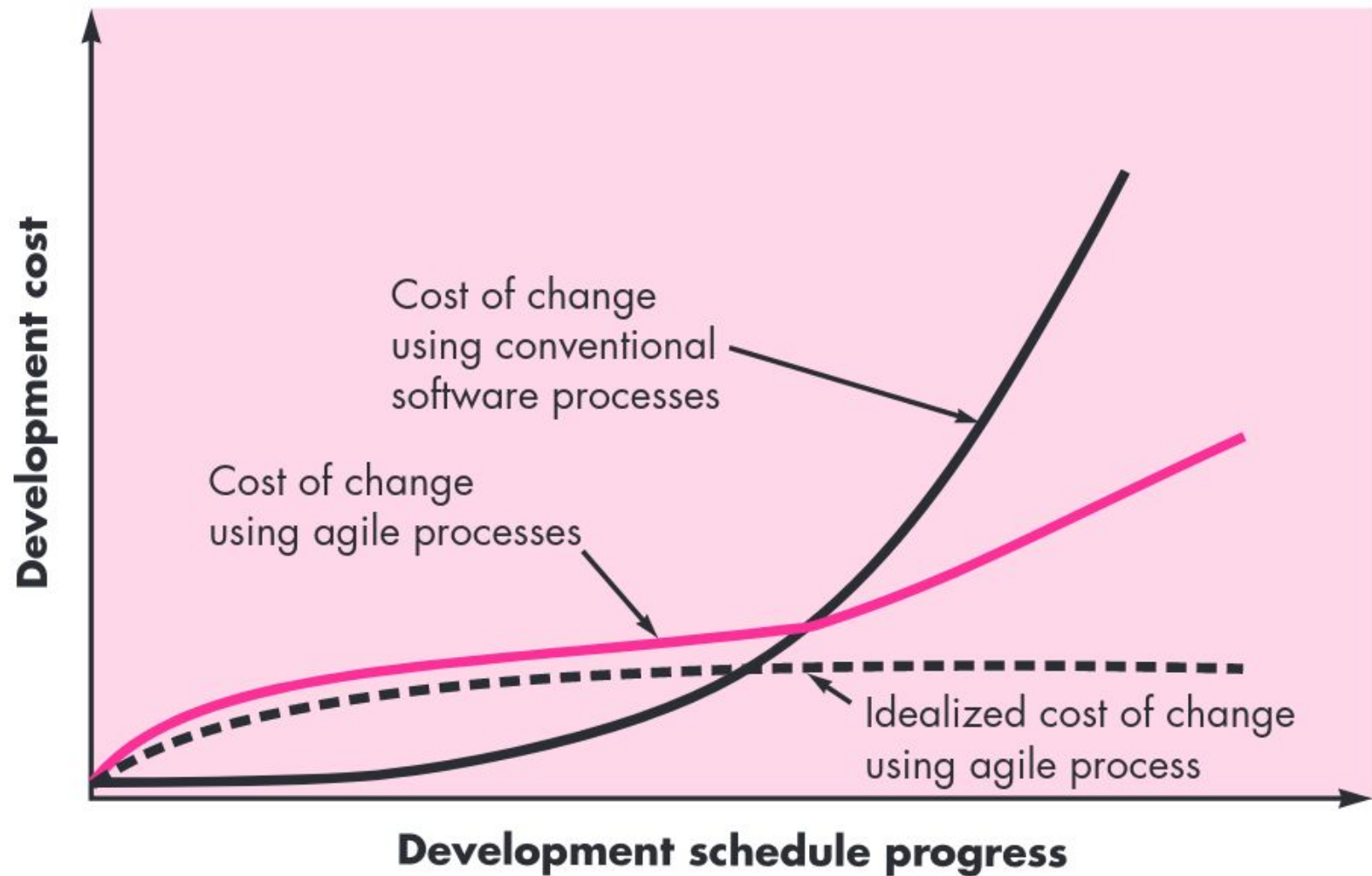
There is no perfect software engineering process...

- *Agile development is not the silver bullet!*

The First Law

*“No matter where you are in the system life cycle, **the system will change, and the desire to change it will persist throughout the life cycle.**”*

[Bersoff, 1980]



Software Crisis

“The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.” [Dijkstra]

Software Crisis

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code is difficult to maintain
- Software was never delivered

Current SE Problems

- Software is too expensive and takes too long to build
 - Frequently over budget and over time!
- Low software quality
 - 3.6 billion users, \$1.7 trillion caused by bugs [Tricentis, 2017]
- Software is more complex to support and maintain
- More users = more difficult to scale applications
- Failing to meet requirements
- Lack of diversity in software development teams
- Inadequate testing and security
- Ethical decisions in software engineering
- ***What else???* [Discuss w/ partner, and in HW1]**



Alexander Rechevskiy • Following

🚀 Land Your Dream Product Role | Grow Your PM Skills & Career | Practical...

5d • 🌐



I was a Group PM at Google for 4 years.

5 truly bizarre things I didn't expect to see at a top tech company:



1. No processes

There are no project management tools.

Projects are tracked and managed in manual spreadsheets and non-synchronized docs.

There are no standalone planning tools.

Most (if not all) processes are created from scratch by each team as they see fit.

Often, this means there are no processes at all.

Every new PM/PgM/lead suggests a different way of doing things and takes a few months to align with the cross-functional team before rolling it out.

Then, a new way is suggested and the cycle repeats.

Software Engineering?

**“ ‘It’s Engineering... but not as we know it’...
Software Engineering - solution to the
software crisis, or part of the problem? ”**

Next Class...

- **Technical Interview Workshop**
- **Requirements on Wednesday**
- **No class on Monday**

- **HW1 (due Friday 9/8 at 11:59pm)**
- **Project Details next week (9/6)**

References

- Standish Group. “*The CHAOS Report*”.
<https://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf>
- Beck, et al. “*Agile Manifesto*”. 2001 <<https://agilemanifesto.org/>>
- RS Pressman. “*Software engineering: a practitioner's approach*”.
- Alistair Cockburn. “*Crystal clear: A Human-powered methodology for small teams*”
- Tom and Mary Poppendieck. “*Implementing Lean Software Development*”
- Na Meng and Barbara Ryder
- Chris Parnin, Software process phase slides based on NCSU CSC216 slides by Sarah Heckman