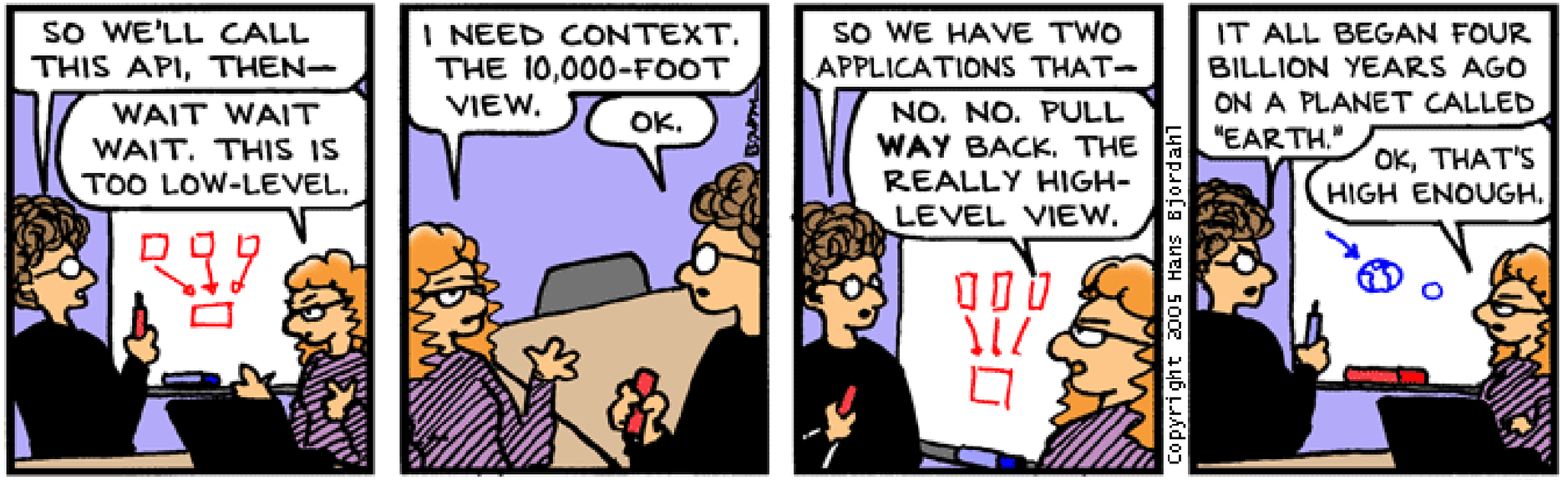
Homework and OH

- HW0: due this Sunday
- Autograder and Brightspace
- Team up or not?
- Environment
  - Can I use XYZ instead of Ubuntu 22.04 or VirtualBox/EC2?
    - Nothing “bad” will happen: I will not fail you in this class; you are responsible for the development environment



The Software Development Lifecycle (+HW1)

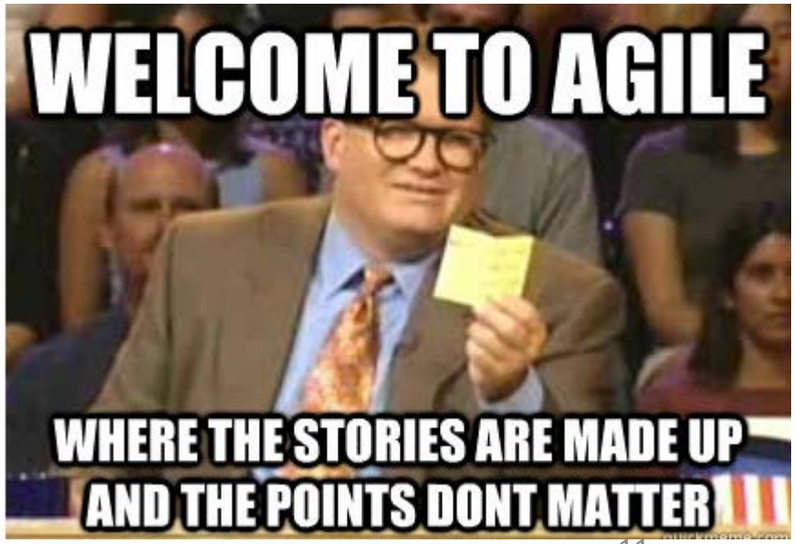
One-Slide Summary

- A **software development process** organizes activity into distinct *phases* (e.g., design, coding, testing, etc.).
  - Processes can increase efficiency, but are often implemented poorly.

- **Effort estimation** is based on historical information
  - **Modeling** or **experience** both used for planning
  - **Risk** leads to **uncertainty,** mitigated by *identification* and *minimization*
  - A **project plan** (milestones, deliverable) includes all considerations of risk management
  - Measuring **progress** is difficult

Process

- A **software development process** (a.k.a **software development life cycle** or **software development model**) *divides* software development into distinct **phases** to improve design, product, and project management.
- Process is *the set of activities and associated results that produce a software product*.
- Examples include the **waterfall model**, **spiral development**, **agile development**, and **extreme programming**.

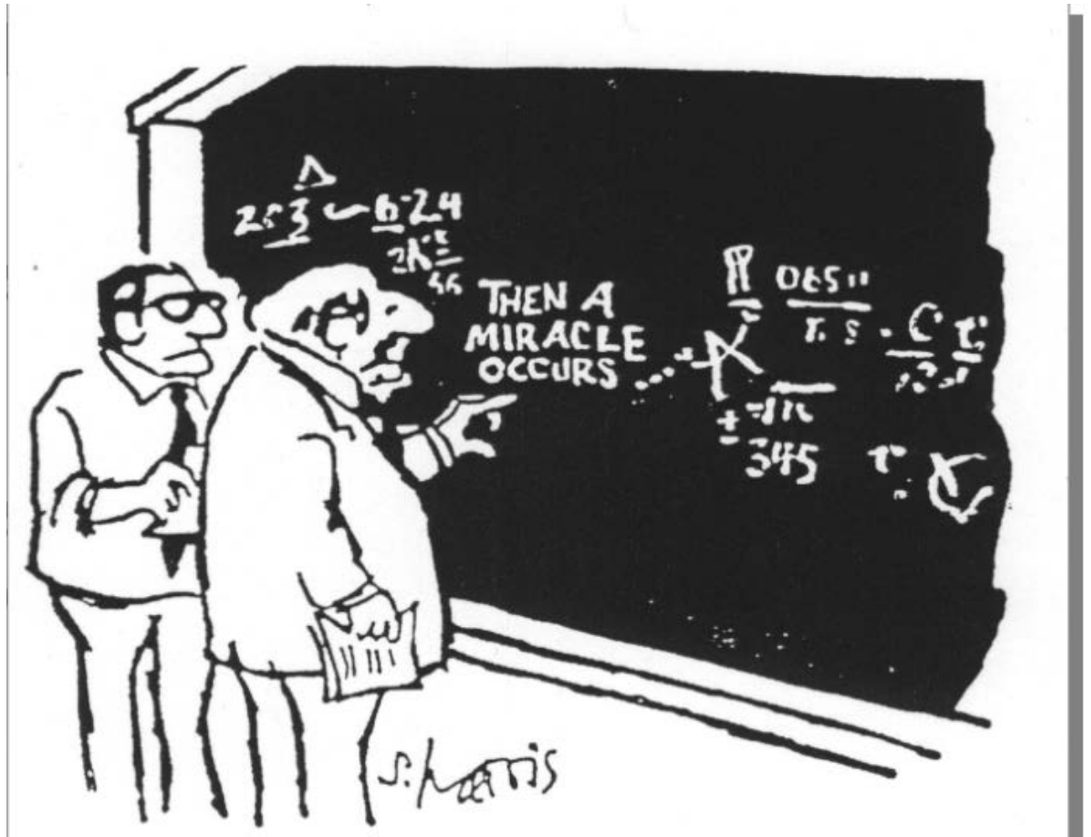


Richard Feynman's Problem Solving Algorithm

1. Write down the problem.
2. Think real hard.
3. Write down the solution.

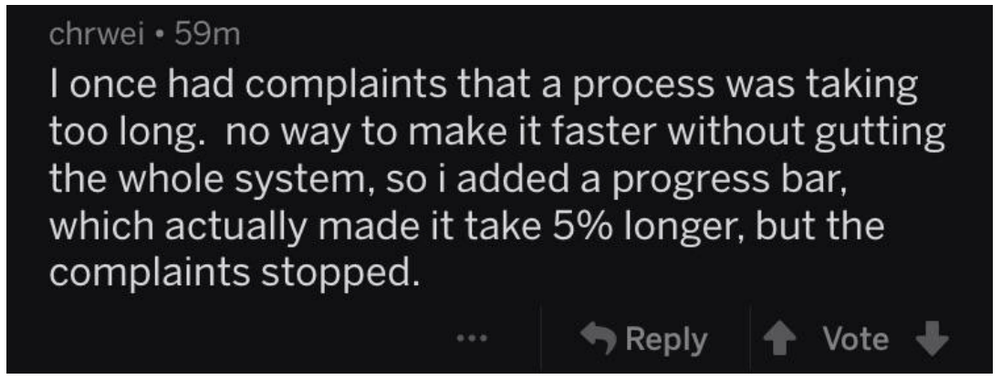
– As facetiously suggested by Murray Gell-Mann, a colleague of Feynman, in the New York Times

https://wiki.c2.com/?FeynmanAlgorithm



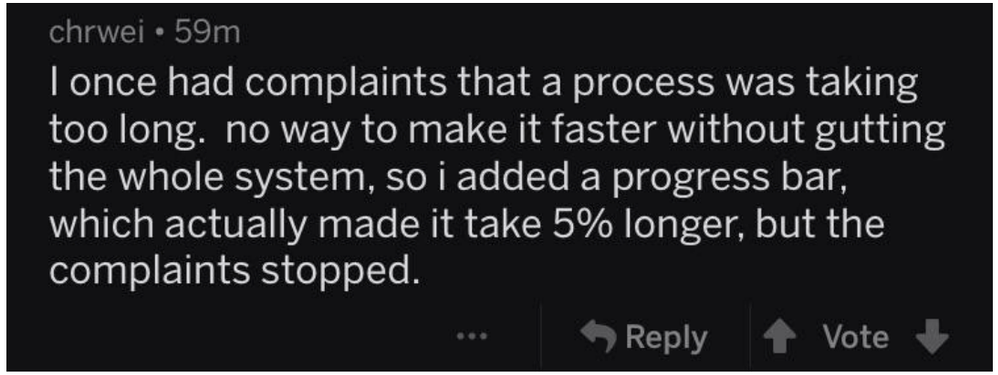
A **Straw** Software Process

1. Discuss the software that needs to be written



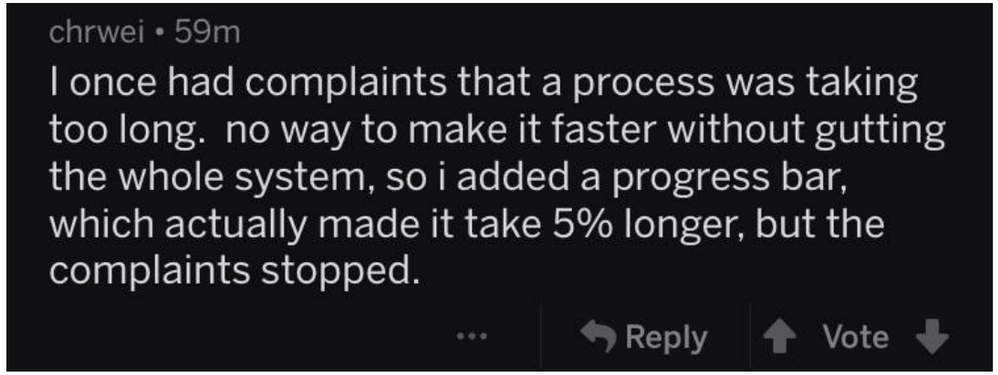
A **Straw** Software Process

1. Discuss the software that needs to be written
2. Write some code



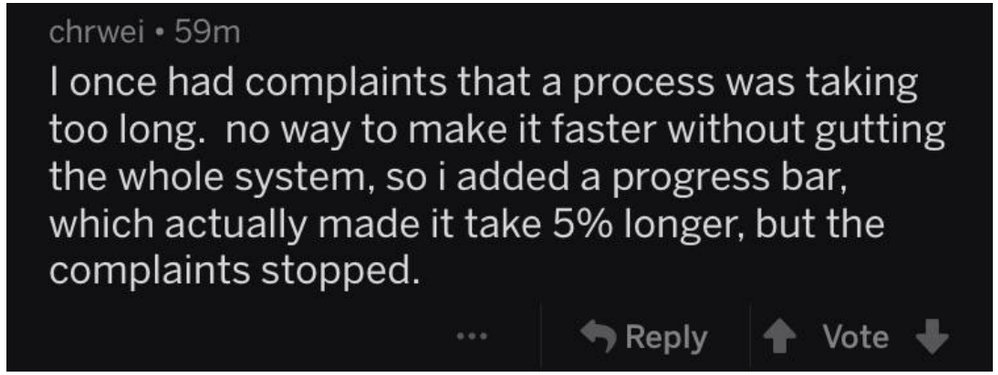
A **Straw** Software Process

1. Discuss the software that needs to be written
2. Write some code
3. Test the code to identify the defects



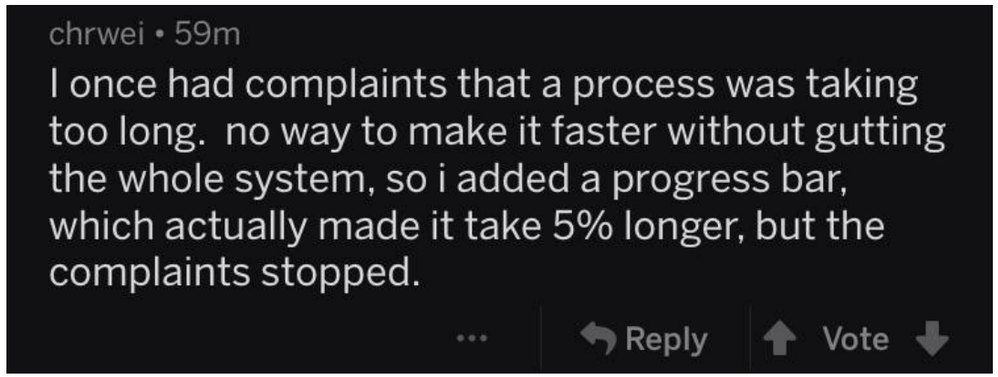
A **Straw** Software Process

1. Discuss the software that needs to be written
2. Write some code
3. Test the code to identify the defects
4. Debug to find causes of defects



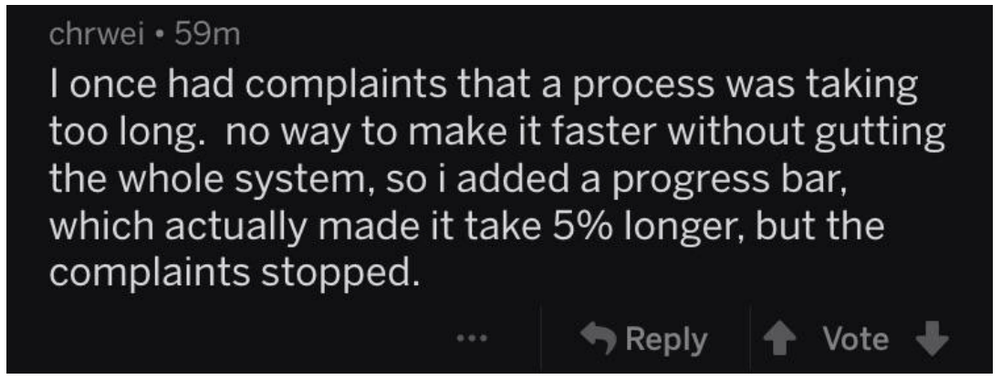
A **Straw** Software Process

1. Discuss the software that needs to be written
2. Write some code
3. Test the code to identify the defects
4. Debug to find causes of defects
5. Fix the defects



A **Straw** Software Process

1. Discuss the software that needs to be written
2. Write some code
3. Test the code to identify the defects
4. Debug to find causes of defects
5. Fix the defects
6. If not done, return to Step 1



Waterfall Model

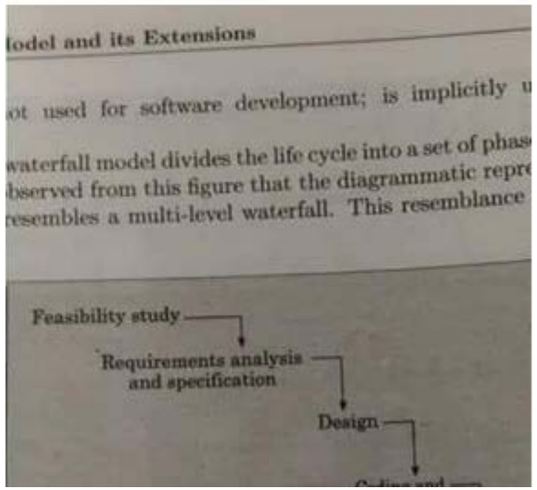
- In the **waterfall** software development model, the following phases are carried out *in order*:

- System and software requirements
  - Elicited from customer, captured in a document
- Analysis
  - Derive models, schema, and business rules
- Design
  - Software architecture
- Coding
  - Development, proving, and integration of software
- Testing
  - Systematic discovery and debugging of defects
- Operations
  - Installation, migration, support, and maintenance of complete systems

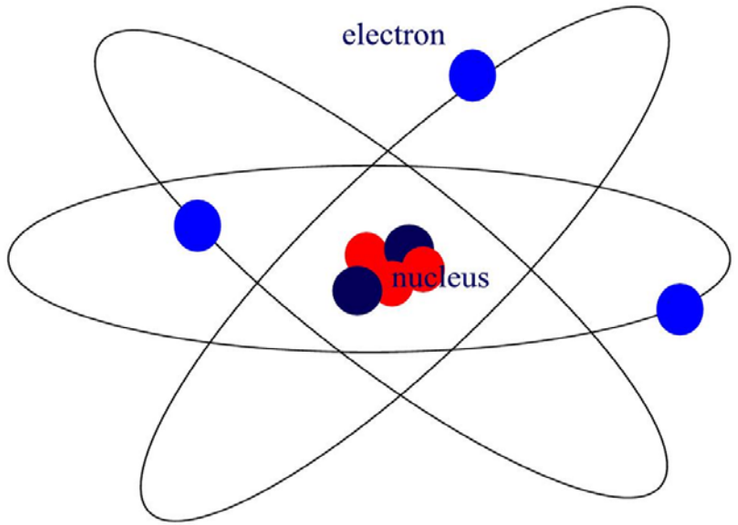
Waterfall Model

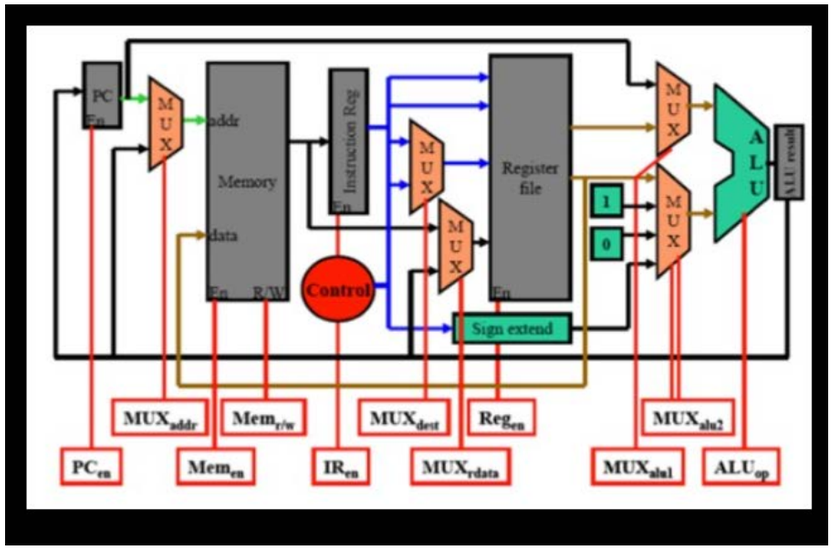
Expectation

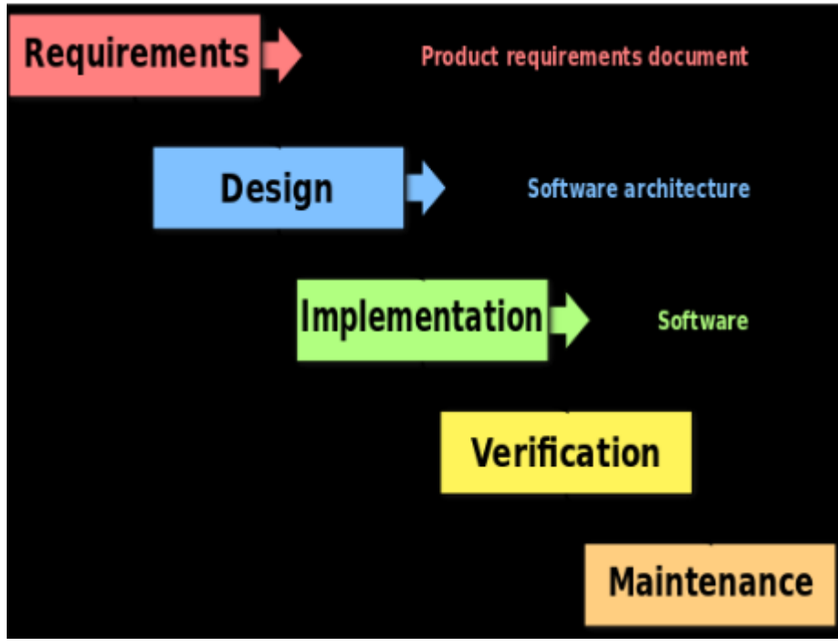
Reality



Tell Me Lies

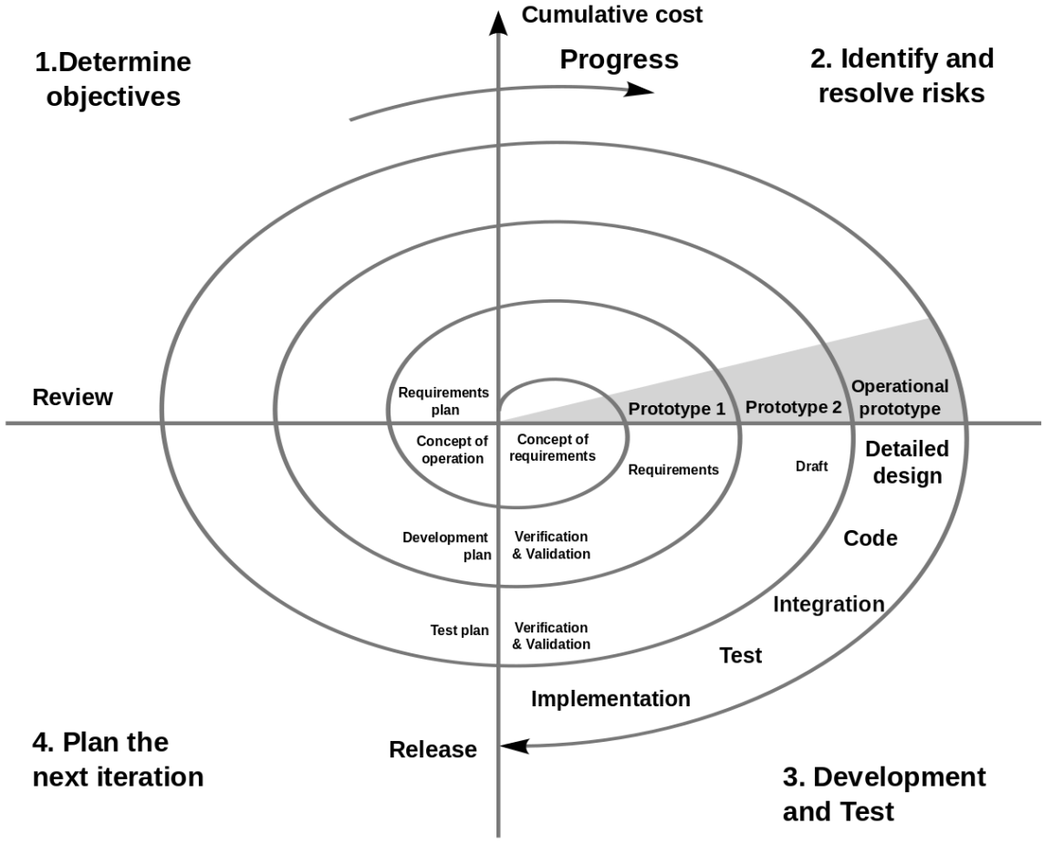

electron
nucleus


PC
En
M
U
X
addr
Memory
data
En
R/W
Instruction Reg
En
Control
Register file
En
Sign extend
1
0
A
L
U
MUXaddr
Memr/w
MUXdest
Regen
MUXalu2
PCen
Memen
IRen
MUXrdata
MUXalu1
ALUop


Requirements
Product requirements document
Design
Software architecture
Implementation
Software
Verification
Maintenance

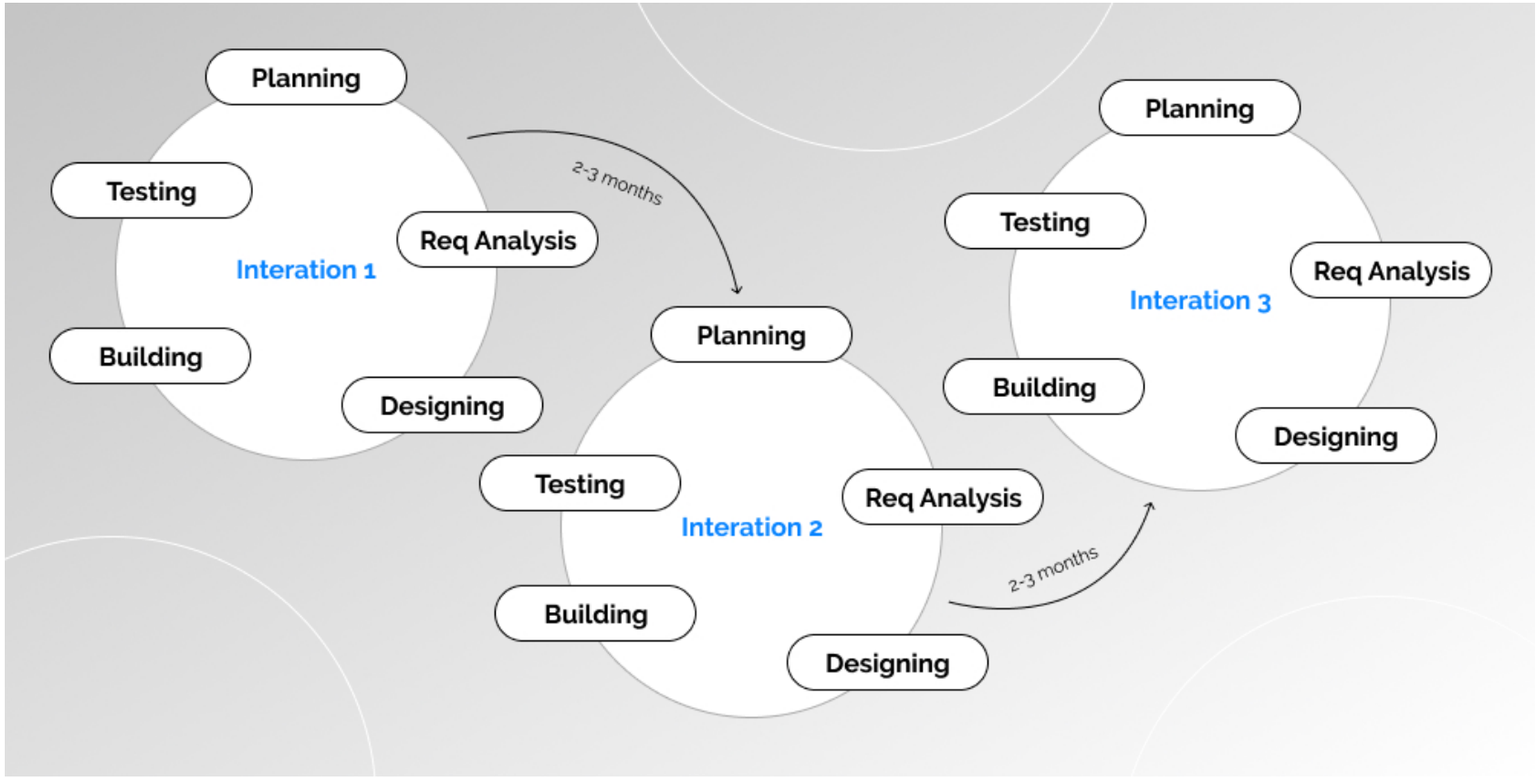
Spiral Development Model

- The **spiral** software process model focuses on the construction of an increasingly-complete series of prototypes while accounting for risk.



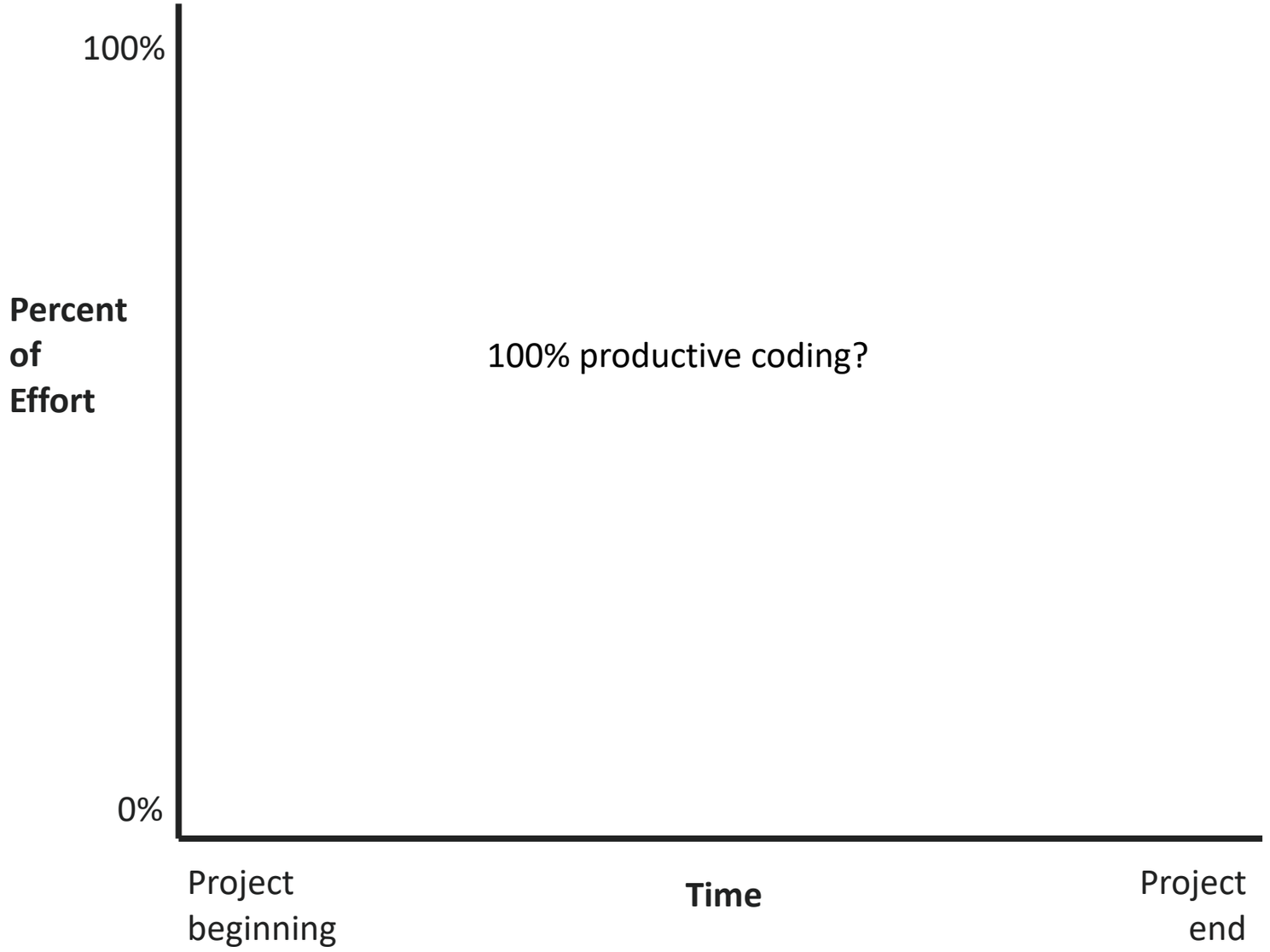
Agile Development Model

- Agile: ability to create and respond to change, deal with uncertain and turbulent environment
- Agile software development is an umbrella term for a set of frameworks and practices
  - scrum
  - extreme programming
  - sprints/iteration
  - stand-ups
  - kanban
  - pair programming
  - test-driven development

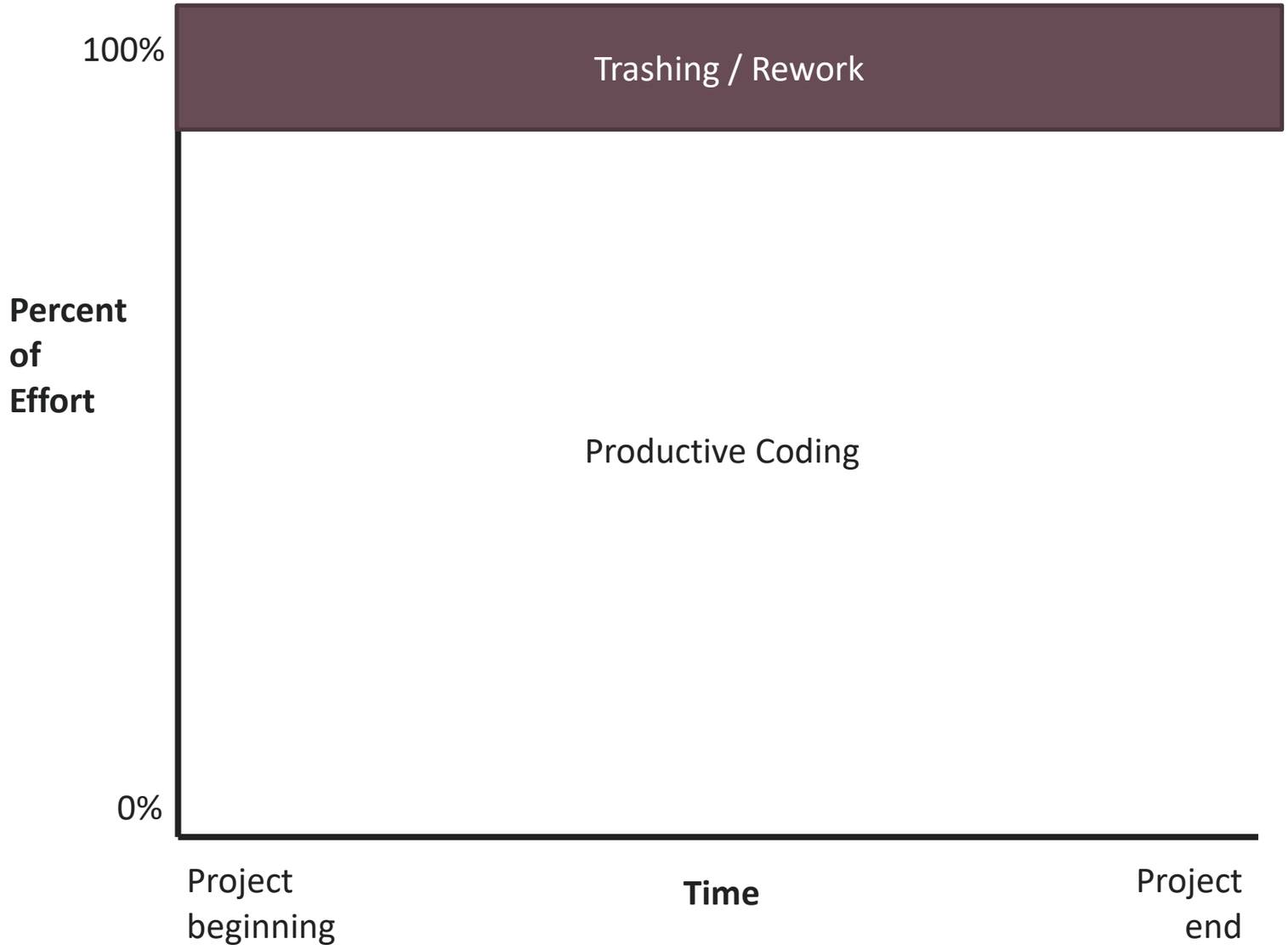


https://www.agilealliance.org/agile101/

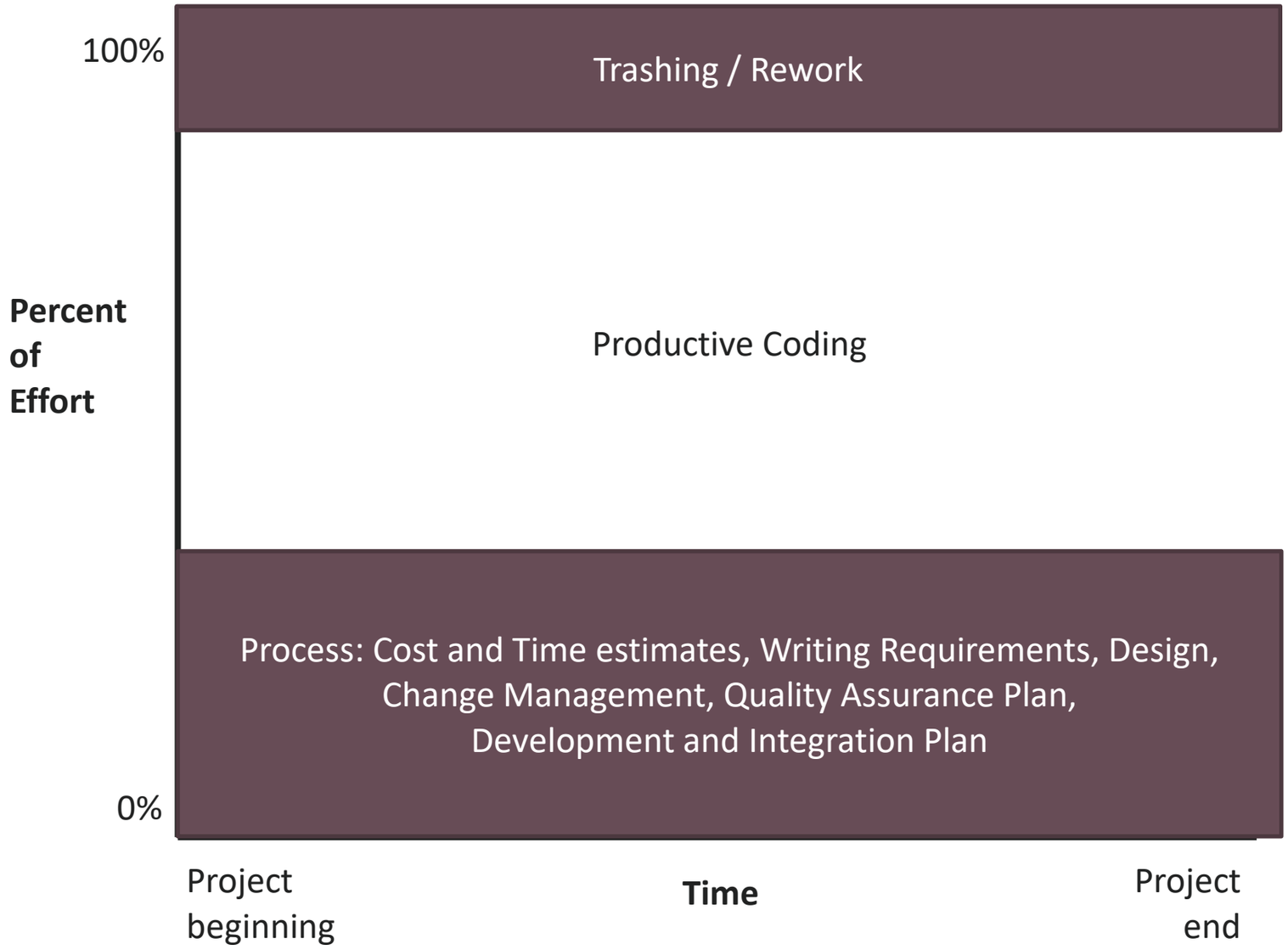
Activity Effort over Time



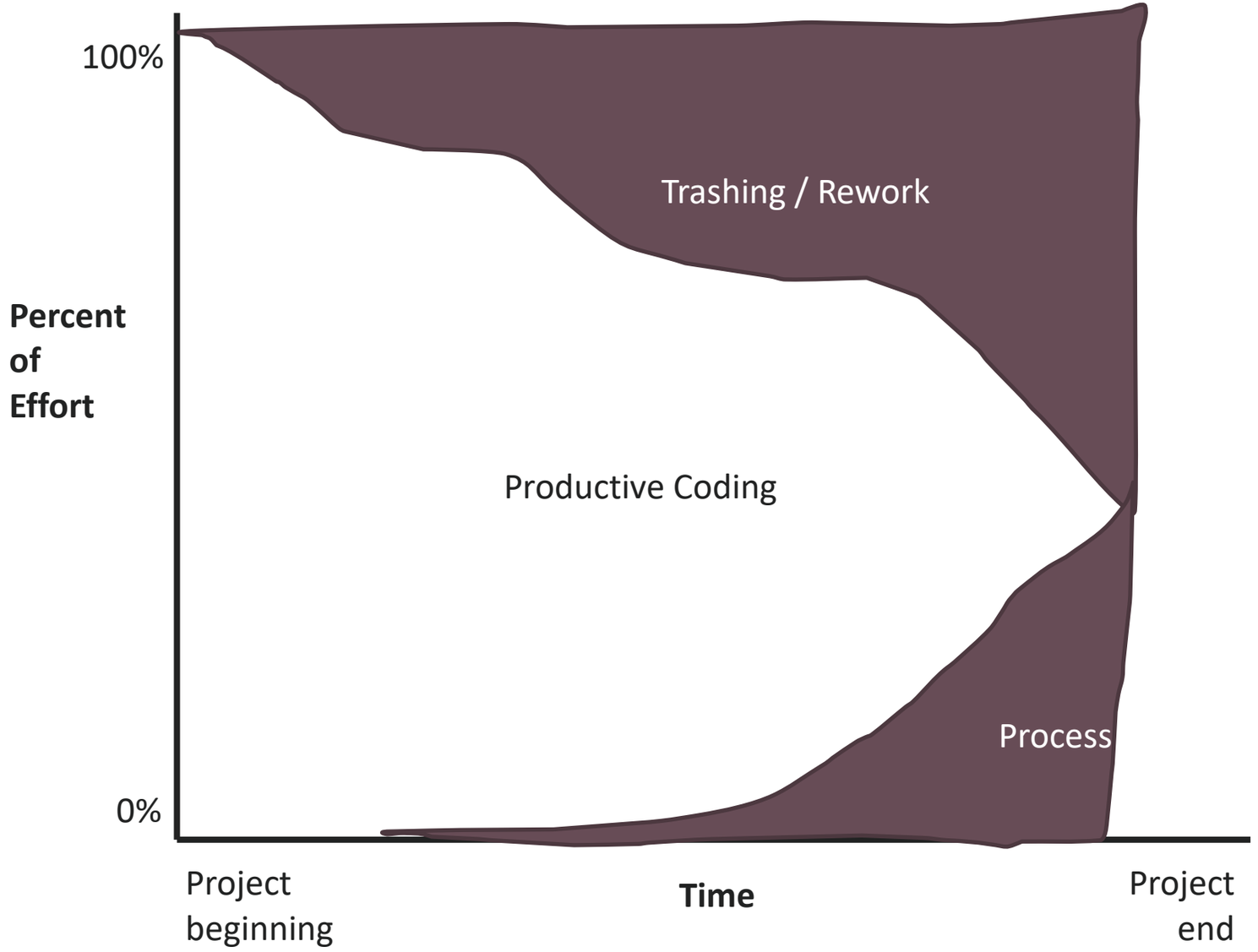
Activity Effort over Time



Idealized View



Result of Failing to Plan



Example Process Issues and Outcomes

- Requirements: Mid-project informal agreement to changes suggested by customer or manager
  - → Project scope expands 25-50%
- Quality Assurance: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features.
  - → Release with known defects
- Defect Tracking: Bug reports collected informally
  - → Bugs forgotten
- System Integration: Integration of independently-developed components at the end of the project
  - → Interfaces out of sync
- Source Code Control: Accidentally overwritten changes
  - → Lost work
- Scheduling: When project is behind, developers are asked weekly for new estimates
  - → Project falls further behind

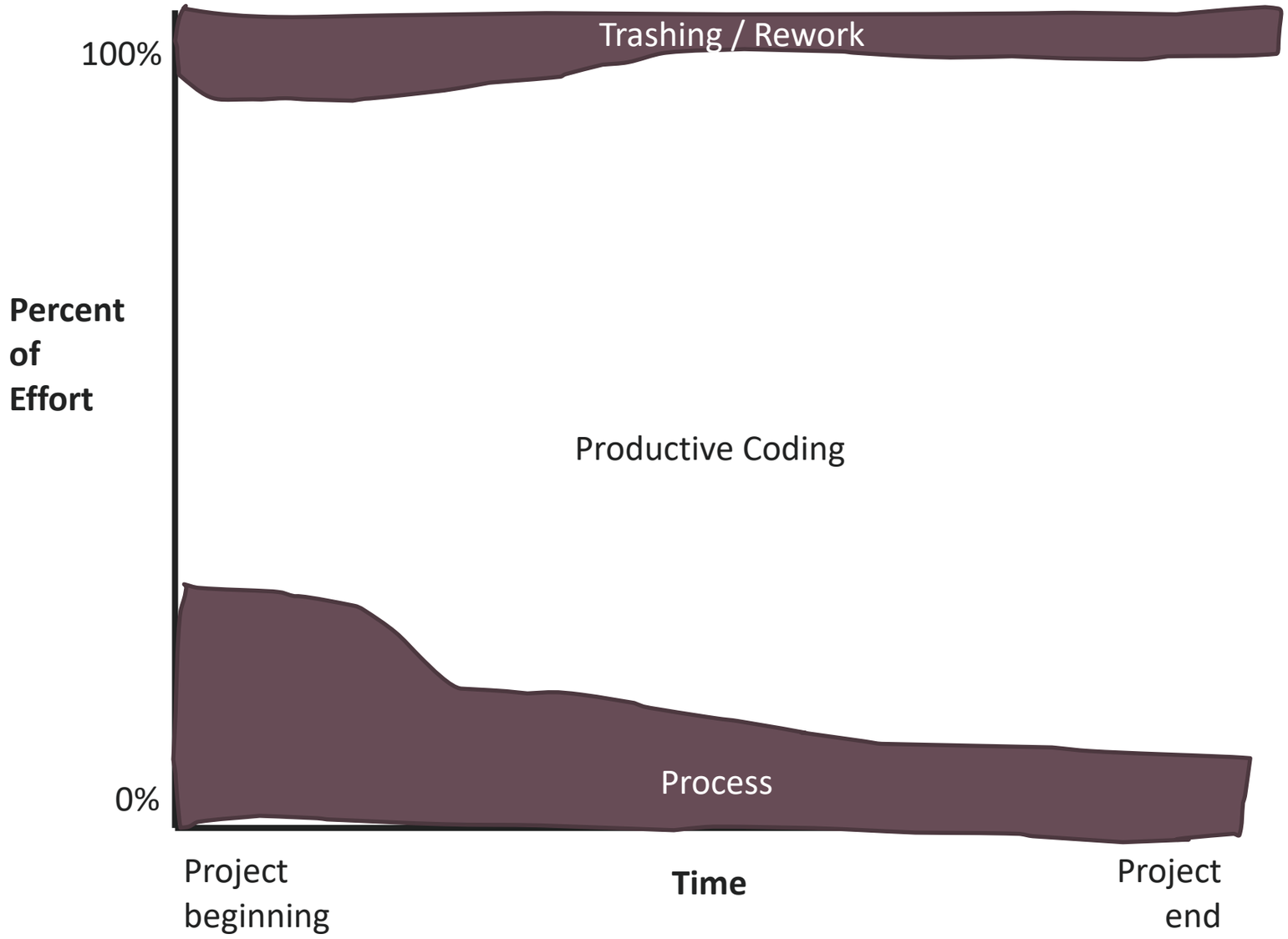
Survival Mode

- Missed deadlines → “solo development mode”, **developers stop interacting** with testers, technical writers, managers, etc.
- “The producers even set a **deadline**; they gave **a specific date** for the **end of the crunch**, which was still *months away* from the title's shipping date, so it seemed safe. *That date came and went*. And went, and went. When the next news came it was not about a reprieve; it was another acceleration: twelve hours six days a week, 9am to 10pm.

  Weeks passed. Again the producers had given a **termination date on this crunch** that again they **failed**. Throughout this period the project *remained on schedule*. The long hours started to take its toll on the team; people grew irritable and some started to get ill. People *dropped out in droves* for a couple of days at a time, but then the team seemed to reach equilibrium again and they plowed ahead. The managers stopped even talking about a day when the hours would go back to normal.” – *EA: The Human Story*

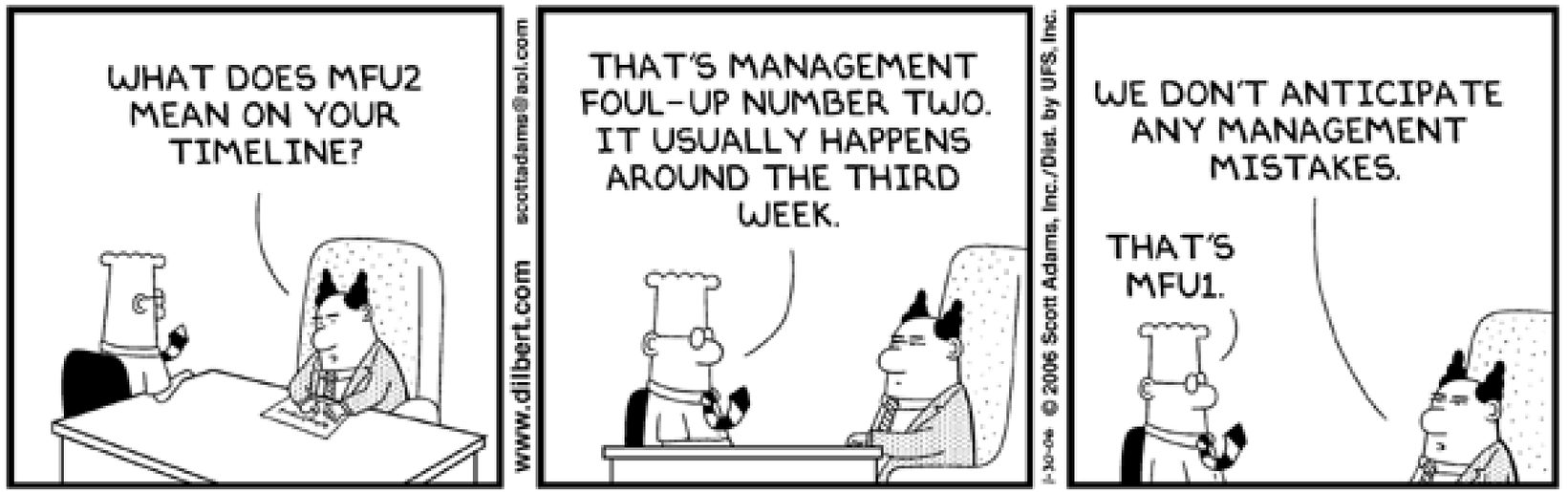
  https://ea-spouse.livejournal.com/274.html

Desired Allocation

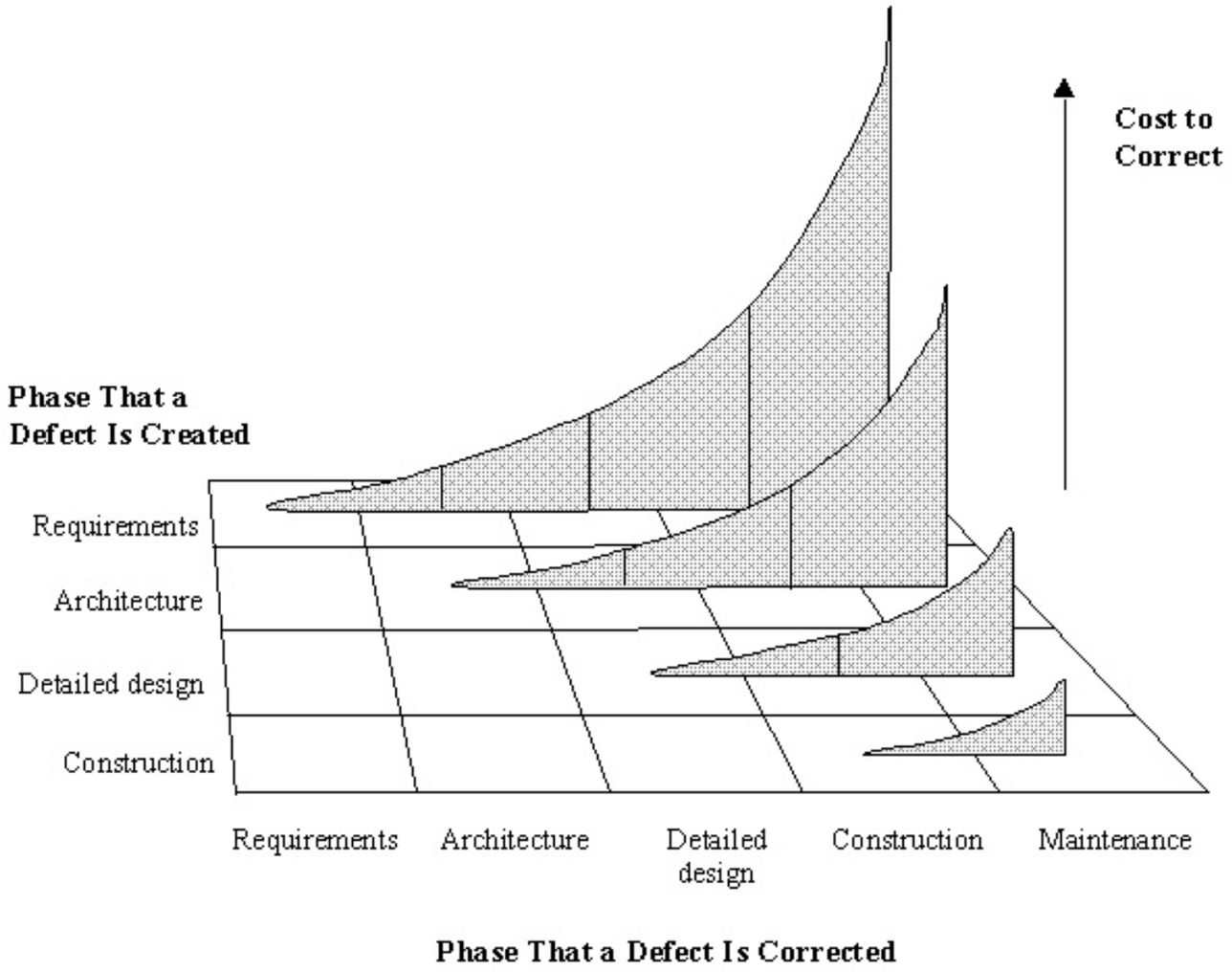


Process Hypothesis

- A process can increase flexibility and efficiency for software development
- If this is true, an up-front **investment** (of resources, e.g., “time”) in process can yield greater **returns** later



Efficiency: Defect Cost vs. Creation Time



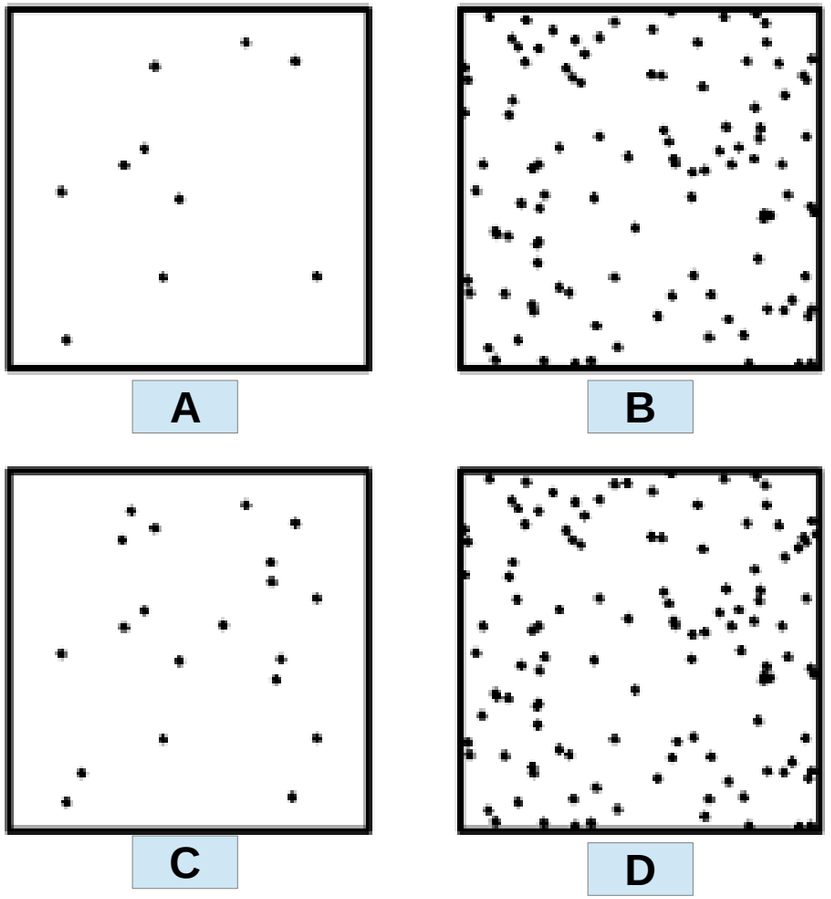
Efficiency: Defect Cost vs. Detection Time

- An IBM report gives an average defect repair cost of
  - $25 during **coding**
  - $100 at **build** time
  - $450 during **testing**/QA
  - $16,000 ***post-release***

- [L. Williamson. *IBM Rational software analyzer: Beyond source code.* 2008.]

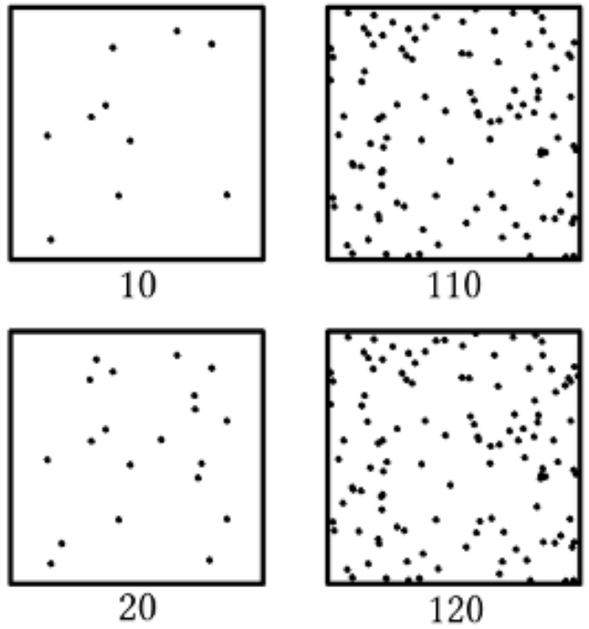
Psychophysics

- Which two figures have the same # of dots?



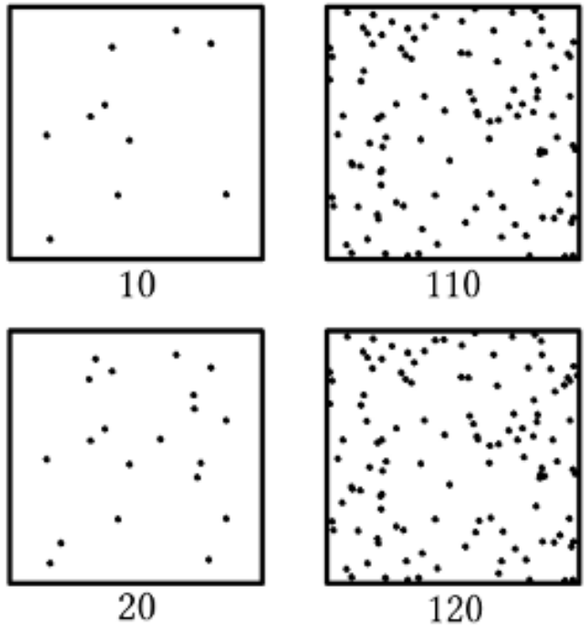
Psychophysics: Weber's Law

- Weber's Law states that “simple differential sensitivity is inversely proportional to the size of the components of the difference; relative differential sensitivity remains the same regardless of size."
- That is “the perceived change in stimuli is proportional to the [size of] initial stimuli.”



Psychophysics: Weber's Law

- Implication for SE: Things you could notice on small-scale projects are harder to notice on large-scale projects. Your intuitions (“I can spot bugs in this”) from small class projects do *not* carry over.



Psychology

- Consider a hypothetical cleanup scenario involving two hazardous waste sites X and Y.
  - X causes 8 cases of cancer annually (large city)
  - Y causes 4 cases of cancer annually (small city)

- Rank these three cleanup approaches:
  - A. X → 4. Y → 2.
  - B. X → 7. Y → 0.
  - C. X → 3. Y → 3.

Psychology: Zero-Risk Bias

- Three cleanup approaches:
  - A. X → 4. Y → 2.
  - B. X → 7. Y → 0.
  - C. X → 3. Y → 3.

- “The bias was defined as not ranking the complete-reduction option [B] as the worst of the three options. (It should be ranked worst because it saves fewer cancer cases.) 42% of the subjects exhibited this `**zero-risk**' bias.”

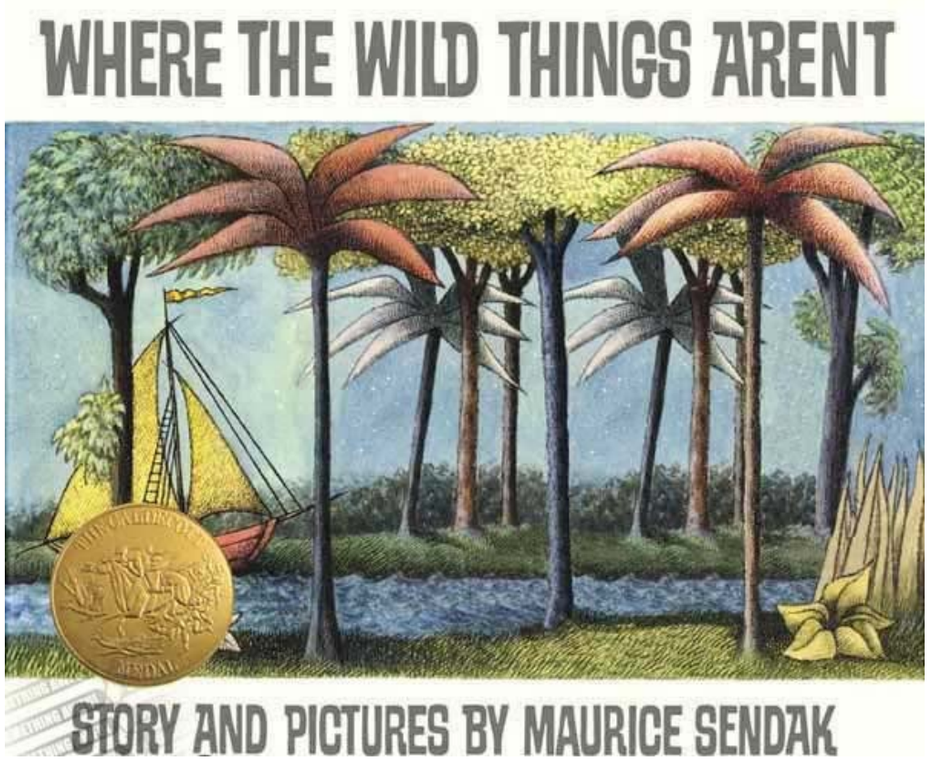
- [Baron; Gowda; Kunreuther (1993). "Attitudes toward managing hazardous waste: What should be cleaned up and who should pay for it?". Risk Analysis. 13: 183–192.]

Psychology: Zero-Risk Bias

- **Zero-risk bias** is a tendency to prefer the complete elimination of a risk even when alternative options produce a greater reduction in risk (overall).
- “42% of the subjects exhibited this `**zero-risk**' bias.”
  - Who? 60 CEOs of Oil and Chem Companies, 57 Economists, 94 Environmentalists, 29 Experts on Hazardous Waste, 89 Judges, 104 Legislators.
- Implications for SE: Your managers (and you) are likely to mistakenly favor risk-reduction strategies that reduce a risk to zero, even to the overall detriment of the company/product.

Process Topics

- Estimating Effort
- Risk and Uncertainty
- Planning and Scheduling



Estimating Time Costs

- How long would you estimate to develop a …
  - Java Monopoly game (you alone)
  - Bank smartphone app (you with a team of four developers, one with iPhone experience, one with a security background)

- Estimate in eight-hour workdays (20 in a month, 220 per year)
- Approach: break down the task into ~five smaller tasks and estimate them. Repeat.

Basic Plan: Learn from Experience

Constructive Cost Model

- A **constructive cost model** (**cocomo**) is a predictive model of time costs based on project history.
- This requires experience with similar projects.
- This rewards documentation of experience.

- Basically, it's an empirically-derived set of “effort multipliers”. You multiply the time cost by some numbers from a chart:

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Risk and Uncertainty

- **Risk management** is the identification, assessment, and prioritization of risks, followed by efforts to **minimize**, **monitor** and **control** unfortunate event **outcomes** and **probabilities**.

- Risk management is a **key project management** task. Examples:

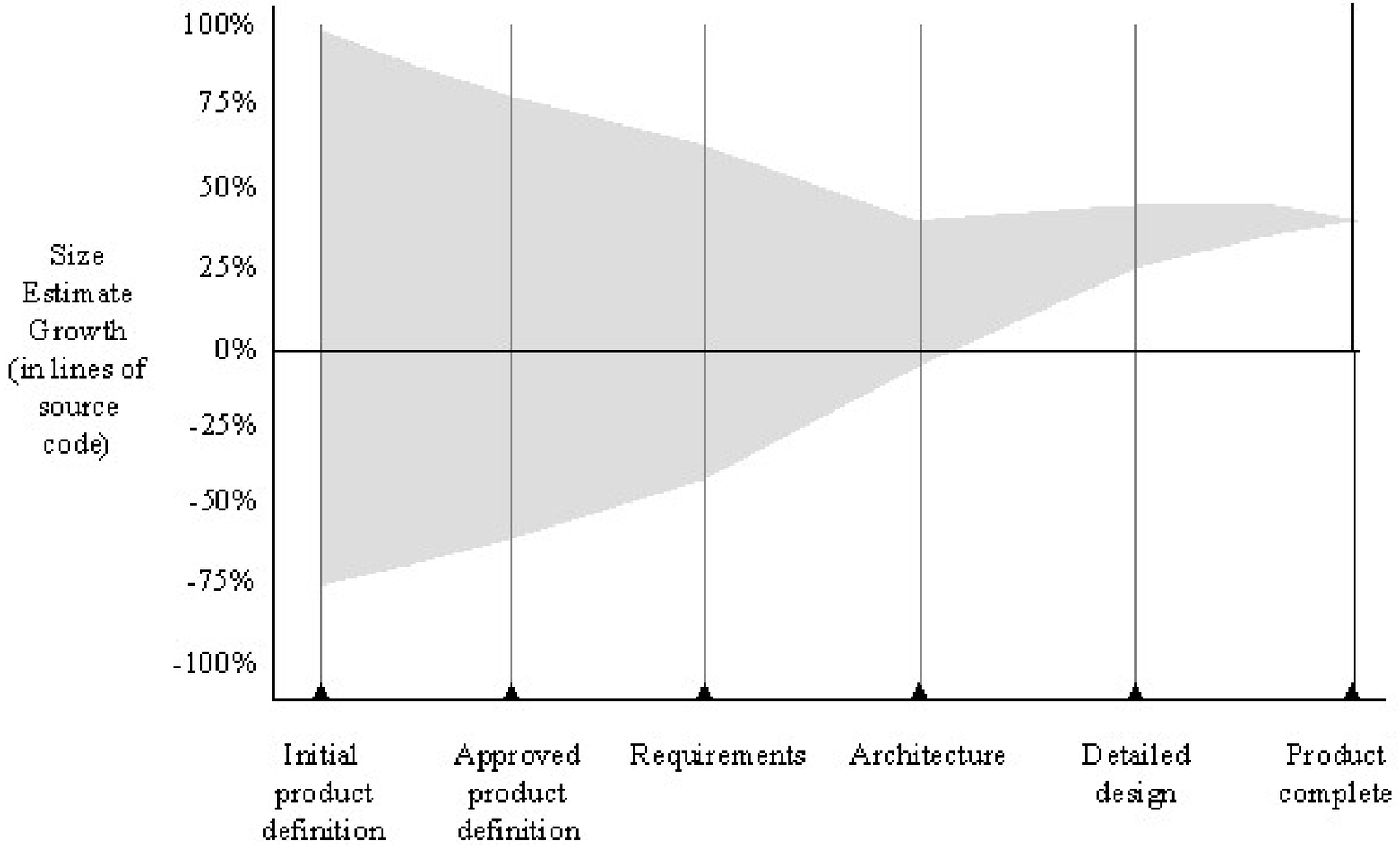
Risk and Uncertainty

- **Risk management** is the identification, assessment, and prioritization of risks, followed by efforts to **minimize**, **monitor** and **control** unfortunate event **outcomes** and **probabilities**.

- Risk management is a **key project management** task. Examples:
  - Staff illness or turnover, product is too slow, competitor introduces a similar product, etc.


THIS ROAD MAY
OR MAY NOT BE
CLOSED
ON OR ABOUT
MAY 22, 2012 FOR
APPROX 3 WEEKS

Uncertainty Reduction Over Time



Innovation and Risk

“Innovation”

- Most software projects are innovative
  - Google, Amazon, EBay, Netflix
  - Autonomous vehicles, robotics, biomed
  - Natural language processing, graphics

- Routine projects (now, not ten years ago)
  - E-Commerce website, adaptive control systems (e.g., thermostat), etc.

- As part of the innovation cycle, routine tasks are automated … leaving only innovative ones!

No Catch-All Solution

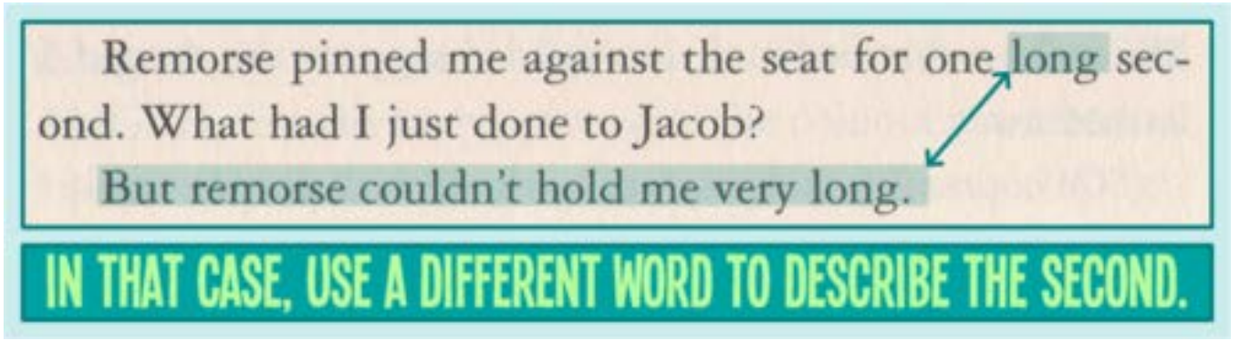
- Address risk **early**
- *Selectively innovate* to **increase value** while **minimizing risk** (i.e., focus risk where needed)
- Use **iteration** and **feedback** (e.g., prototypes)
- Estimate **likelihood** and **consequences**
  - Requires *experienced* project leads
  - Rough estimates (e.g., <10%, <25%) are OK
  - Focus on top ten risks
- Have contingency plans

Examples of Risk Management Strategies

Organizational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business and presenting reasons why cuts to the project budget would not be cost-effective.
Recruitment problems	Alert customer to potential difficulties and the possibility of delays; investigate buying-in components.
Staff illness	Reorganize team so that there is more overlap of work and people therefore understand each other's jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.
Requirements changes	Derive traceability information to assess requirements change impact; maximize information hiding in the design.
Organizational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying-in components; investigate use of a program or test generator.

Planning

- A project should plan time, cost and resources adequately to estimate the work needed and to effectively manage risk during project execution.
  - This includes scoping the work, estimating time costs, developing the schedule and budget, mitigating risks, developing quality assurance measures, etc.



Difficulties in Software Planning

- Typically a one-time endeavor
  (unique wrt. goals, constraints, organization, etc.)
- Typically involves an innovative technology
- **Intangible** results (intermediate or final) mean **progress** may be **hard to measure**

- Software projects tend to fail more often than other industrial projects

Measuring Progress?

“I’m almost done with the app. The frontend is almost fully implemented. The backend is fully finished except for the one stupid bug that keeps crashing the server. I only need to find the one stupid bug, but that can probably be done in an afternoon. We should be ready to release next week.”

```
static int IsNegative(float arg)
{
   char*p = (char*) malloc(20);
   sprintf(p, "%f", arg);
   return p[0]=='-';
}
```

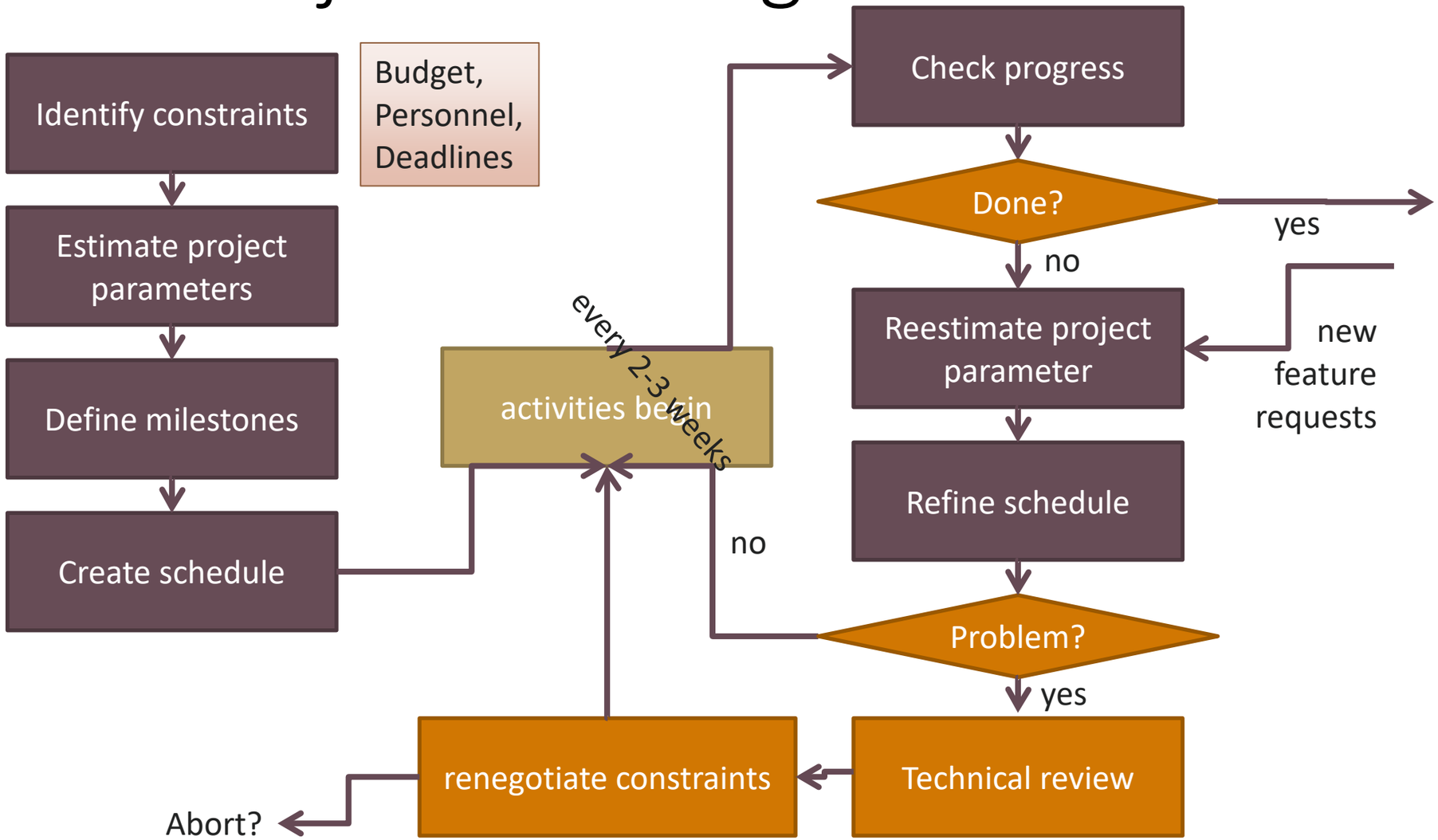
Milestones and Deliverables

- **Milestones** and **deliverables** make **intermediate progress** observable, especially for software

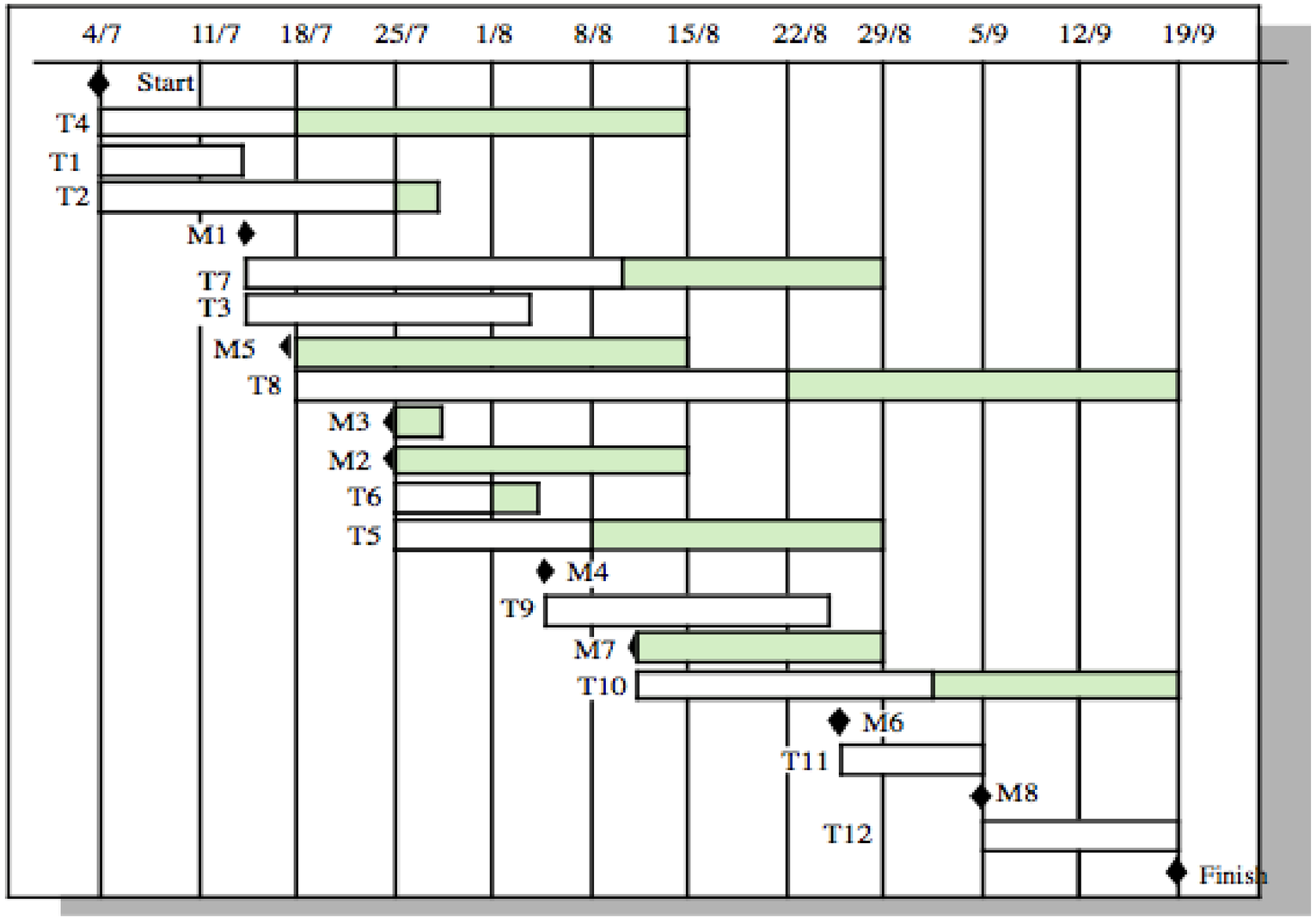
- A **milestone** is a clean end point of a (sub)task
  - Used by the project manager
  - Reports, prototypes, completed subprojects, etc.
  - “80% done” is not a suitable milestone

- **Deliverables** are results for the customer
  - Used by the customer, outward facing

Idealized Project Planning



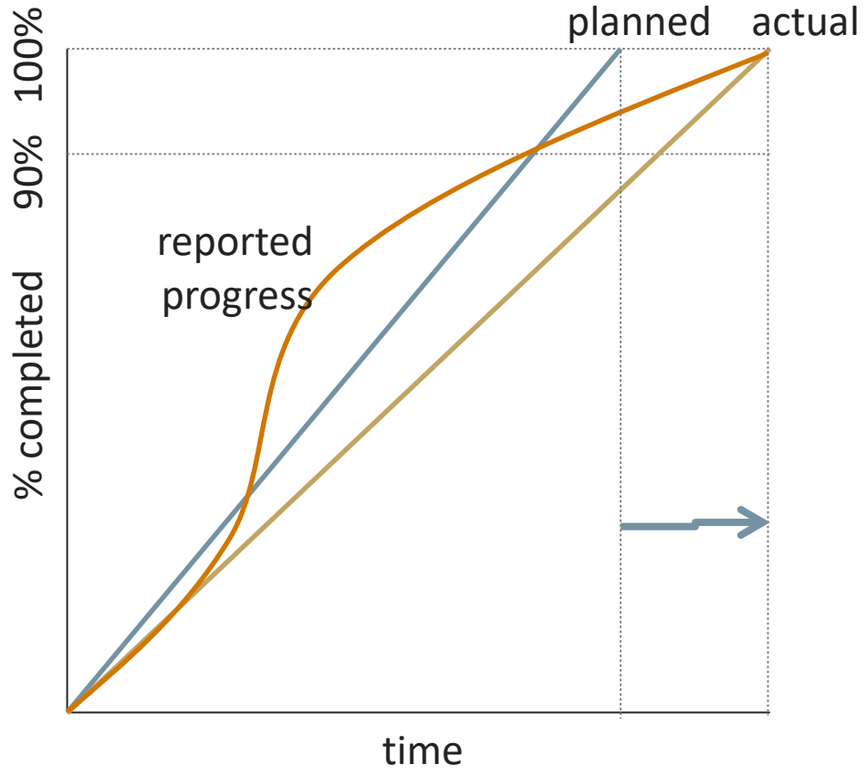
Gantt Diagram



Scheduling

- Inaccurate predictions are normal → update

- The “almost done” problem: the last 10% of work takes 40% of the time
  - Avoid depending entirely on developer estimates



Story So Far

- Software processes can help, but to use them we need project planning, which needs effort estimation, which is complicated by uncertainty, which stems from risk and a lack of data.

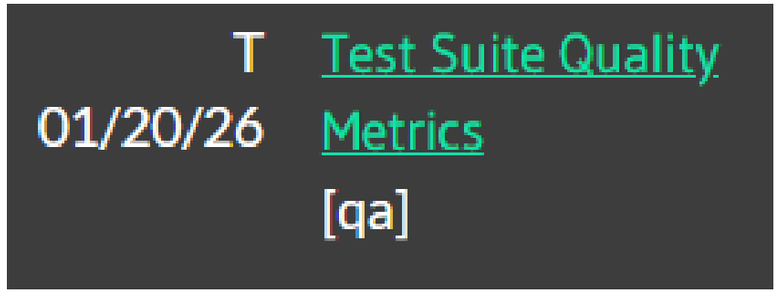
- So … we don't know anything?

- Stay tuned for next time for **measurement**, a potential solution to our problems.

- HW0 due this Sunday
- HW6 – start early: GitHub, research topics

HW1

- This HW is about test coverage
  - We'll talk more about this later



- HW1a due 1/18 HW1bcd due 1/25

- 5 submission attempts per day

HW1

https://huang.isis.vanderbilt.edu/cs4278-sp26/hw1.php