PROCESS AND TECHNICAL DEBT

Christian Kaestner

Required Reading:

- Sculley, David, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. "Hidden technical debt in machine learning systems." In Advances in neural information processing systems, pp. 2503-2511. 2015.

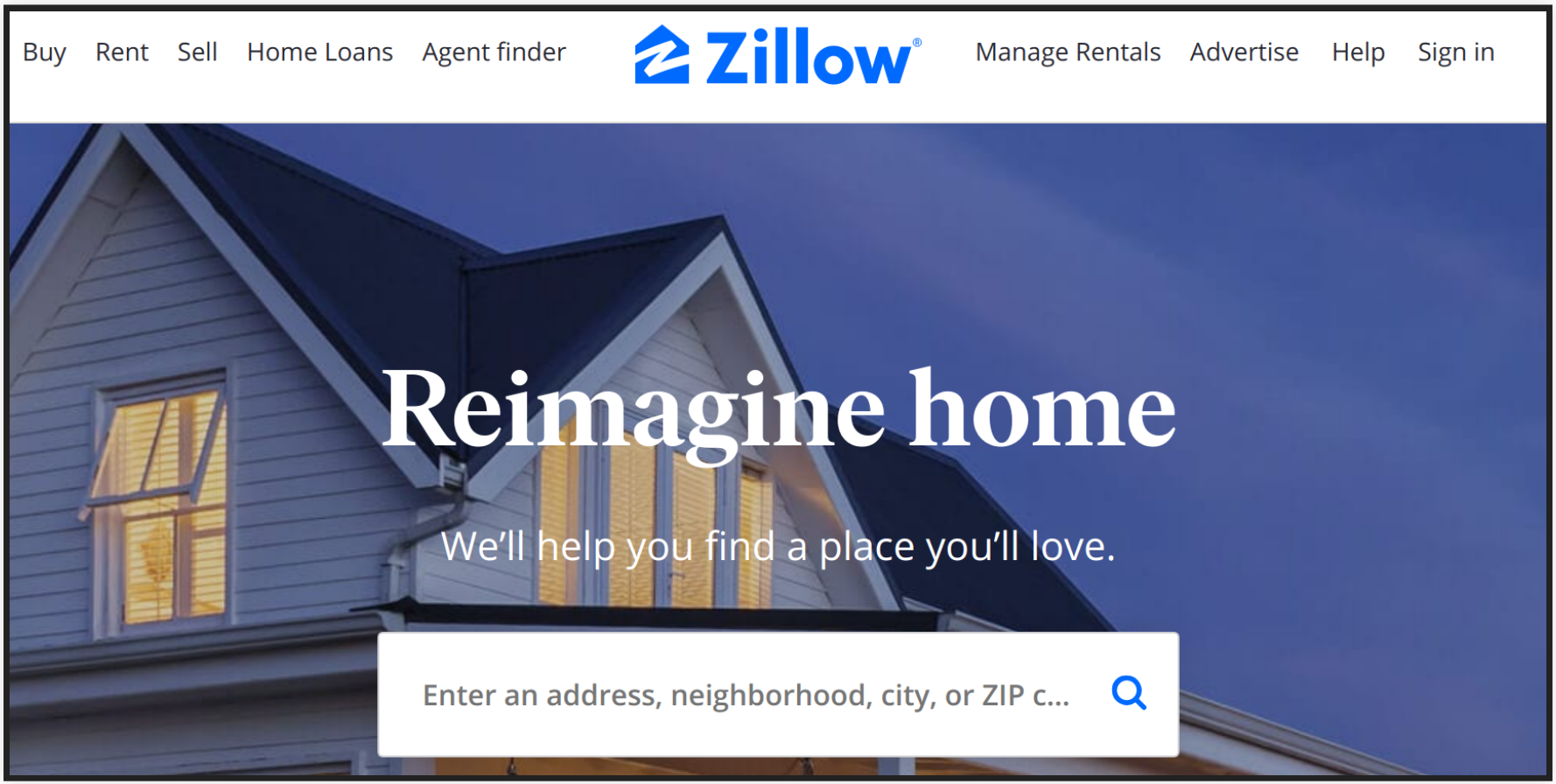
Suggested Readings:

- Fowler and Highsmith. The Agile Manifesto
- Steve McConnell. Software project survival guide. Chapter 3
- Pfleeger and Atlee. Software Engineering: Theory and Practice. Chapter 2
- Kruchten, Philippe, Robert L. Nord, and Ipek Ozkaya. "Technical debt: From metaphor to theory and practice." IEEE Software 29, no. 6 (2012): 18-21.
- Patel, Kayur, James Fogarty, James A. Landay, and Beverly Harrison. "Investigating statistical machine learning as a tool for software development." In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 667-676. 2008.

LEARNING GOALS

- Contrast development processes of software engineers and data scientists
- Outline process conflicts between different roles and suggest ways to mitigate them
- Recognize the importance of process
- Describe common agile practices and their goals
- Understand and correctly use the metaphor of technical debt
- Describe how ML can incur reckless and inadvertent technical debt, outline common sources of technical debt

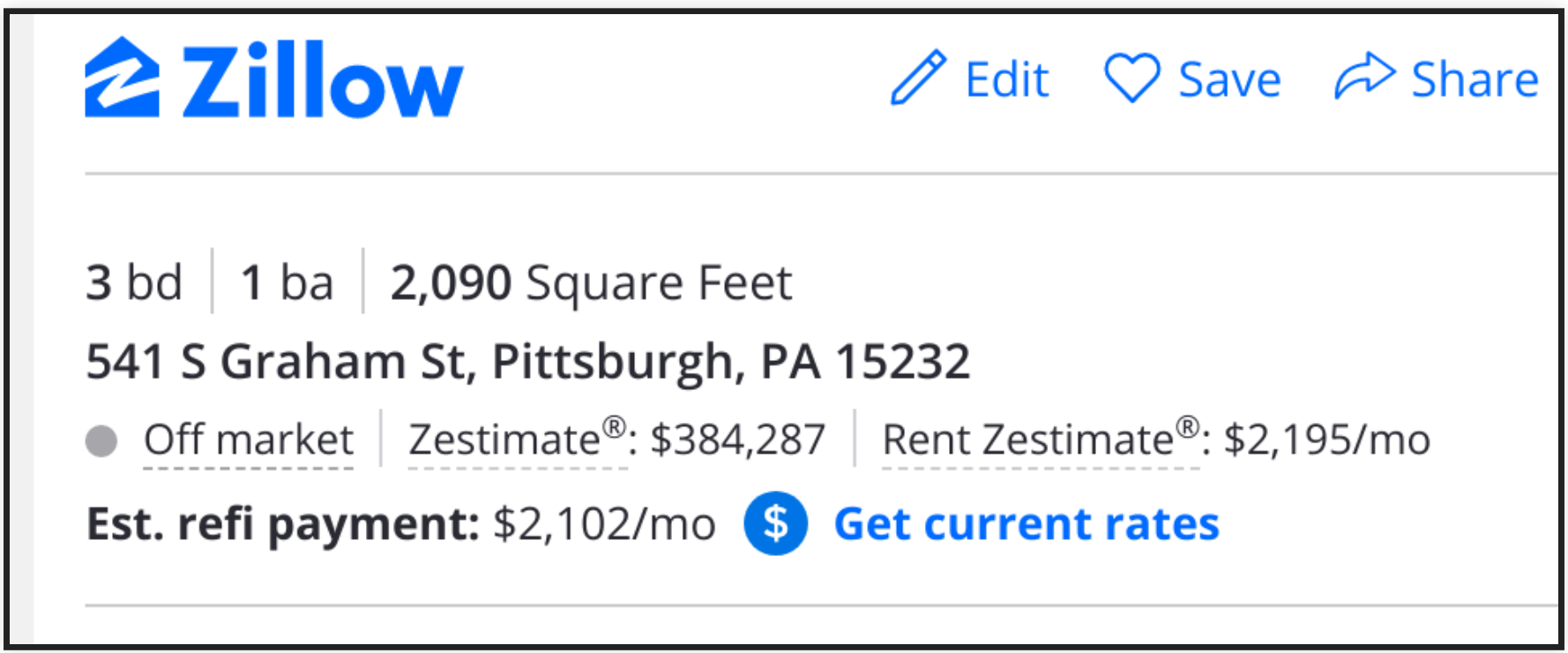
CASE STUDY: REAL-ESTATE WEBSITE



ML COMPONENT: PREDICTING REAL ESTATE VALUE

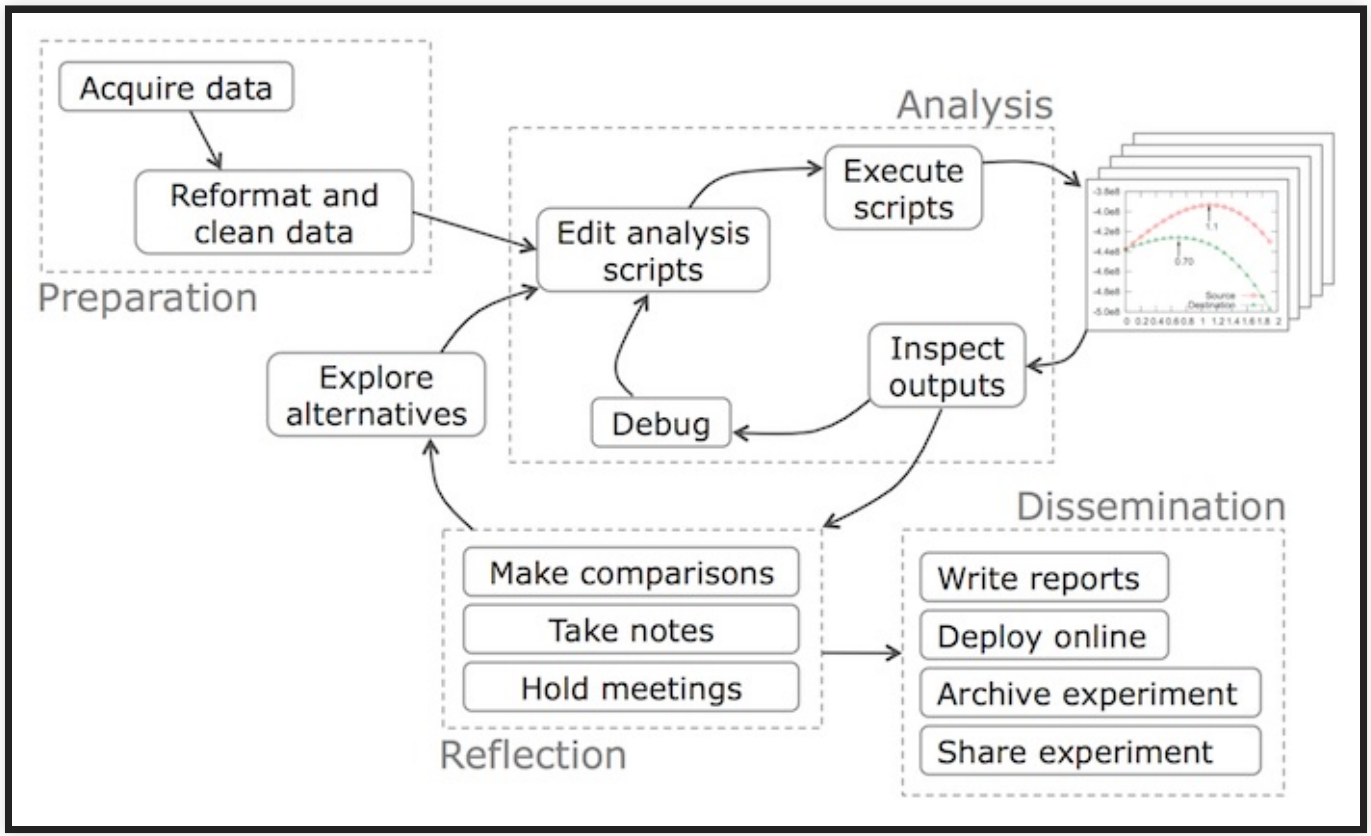
Given a large database of house sales and statistical/demographic data from public records, predict the sales price of a house.

$$f(size, rooms, tax, neighborhood, \ldots) \rightarrow price$$



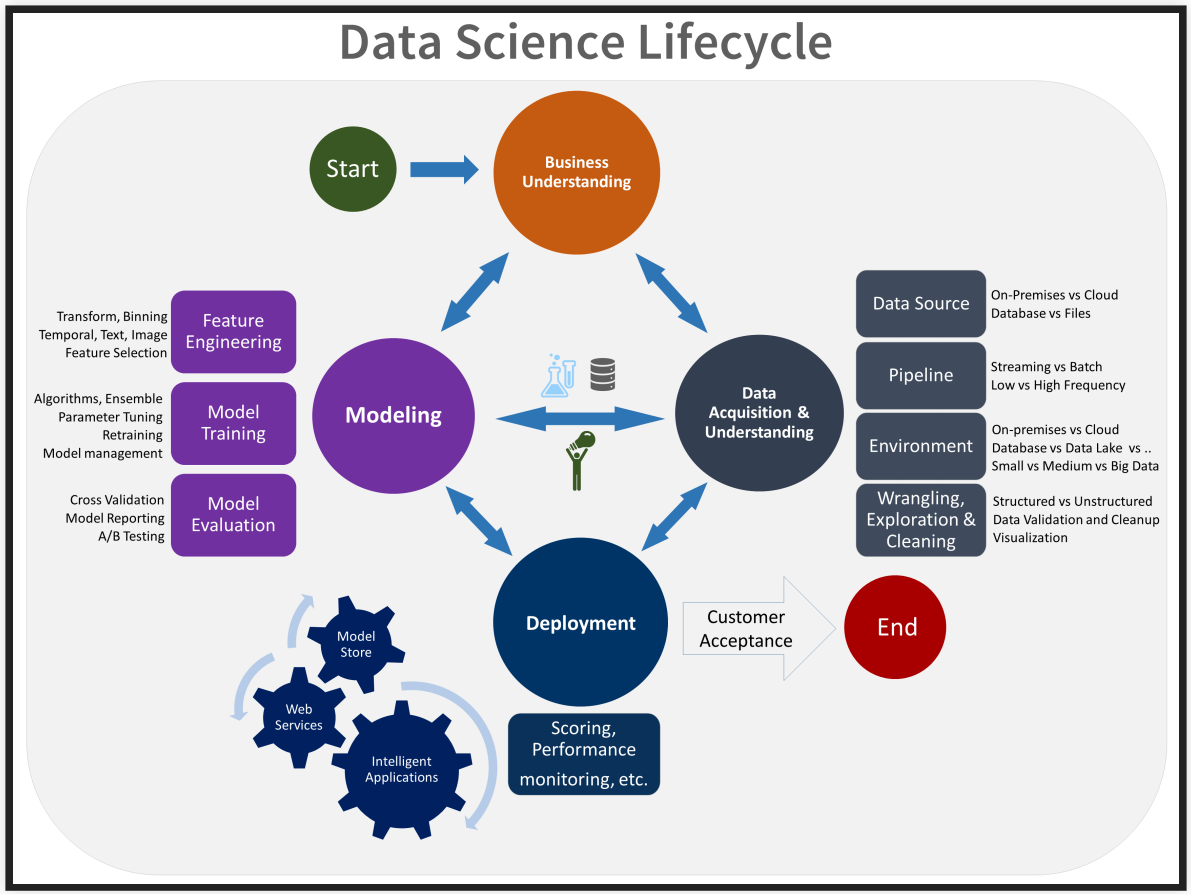
DATA SCIENCE: ITERATION AND EXPLORATION

DATA SCIENCE IS ITERATIVE AND EXPLORATORY



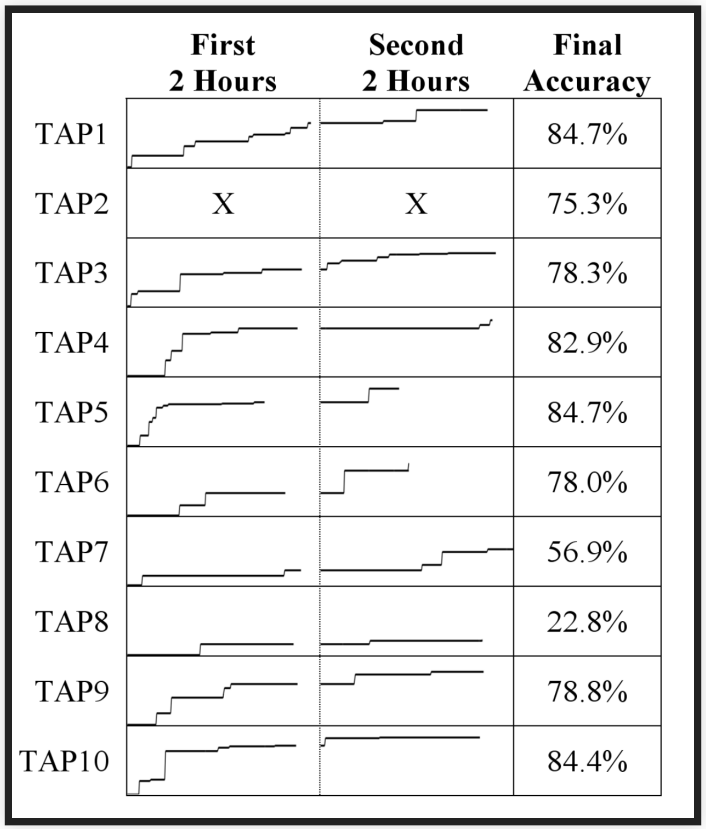
(Source: Guo. "Data Science Workflow: Overview and Challenges." Blog@CACM, Oct 2013)

DATA SCIENCE IS ITERATIVE AND EXPLORATORY



(Microsoft Azure Team, "What is the Team Data Science Process?" Microsoft Documentation, Jan 2020)

DATA SCIENCE IS ITERATIVE AND EXPLORATORY



Source: Patel, Kayur, James Fogarty, James A. Landay, and Beverly Harrison. "Investigating statistical machine learning as a tool for software development." In Proc. CHI, 2008.

Speaker notes

This figure shows the result from a controlled experiment in which participants had 2 sessions of 2h each to build a model. Whenever the participants evaluated a model in the process, the accuracy is recorded. These plots show the accuracy improvements over time, showing how data scientists make incremental improvements through frequent iteration.

DATA SCIENCE IS ITERATIVE AND EXPLORATORY

- Science mindset: start with rough goal, no clear specification, unclear whether possible
- Heuristics and experience to guide the process
- Try and error, refine iteratively, hypothesis testing
- Go back to data collection and cleaning if needed, revise goals

SHARE EXPERIENCE?

COMPUTATIONAL NOTEBOOKS

- Origins in "literal programming", interleaving text and code, treating programs as literature (Knuth'84)
- First notebook in Wolfram Mathematica 1.0 in 1988
- Document with text and code cells, showing execution results under cells
- Code of cells is executed, per cell, in a kernel
- Many notebook implementations and supported languages, Python + Jupyter currently most popular

```
# load data collected from team1
import pandas as pd


url = 'http://128.2.25.78:8080/private/log1.clean'
df = pd.read_csv(url)
df.head()
```

	dayIdx	user	userAvgTime	location	dow	isWeekend	time
0	0	Pittsburgh66Correy	7.045001	Pittsburgh	6	True	0.000000
1	1	Pittsburgh66Correy	7.045001	Pittsburgh	7	True	6.883333
2	2	Pittsburgh66Correy	7.045001	Pittsburgh	1	False	6.816667
3	3	Pittsburgh66Correy	7.045001	Pittsburgh	2	False	7.383333
4	4	Pittsburgh66Correy	7.045001	Pittsburgh	3	False	0.000000

Data was preprocessed externally, identifying the time at a given day when the light was first turned on
12pm). Weather and sunrise information is not included here, though that'd be important. If the light w
this morning (quite common), 0 is recorded.

```
[ ] # just data encoding and splitting X and Y

    X = df.drop(['time'], axis=1)
    YnonZero = df['time'] > 0
    Y = df['time']

    from sklearn import preprocessing
    # leDate = preprocessing.LabelEncoder()
    # leDate.fit(X['date'])
    # leDate.transform(X['date'])

    X=X.apply(preprocessing.LabelEncoder().fit_transform)
    X
```

Speaker notes

- See also https://en.wikipedia.org/wiki/Literate_programming
- Demo with public notebook, e.g., https://colab.research.google.com/notebooks/mlcc/intro_to_pandas.ipynb

NOTEBOOKS SUPPORT ITERATION AND EXPLORATION

- Quick feedback, similar to REPL
- Visual feedback including figures and tables
- Incremental computation: reexecuting individual cells
- Quick and easy: copy paste, no abstraction needed
- Easy to share: document includes text, code, and results

BRIEF DISCUSSION: NOTEBOOK LIMITATIONS AND DRAWBACKS?

SOFTWARE ENGINEERING PROCESS

INNOVATIVE VS ROUTINE PROJECTS

- Like data science tasks, most software projects are innovative
    - Google, Amazon, Ebay, Netflix
    - Vehicles and robotics
    - Language processing, Graphics, AI
- Routine (now, not 20 years ago)
    - E-commerce websites?
    - Product recommendation? Voice recognition?
    - Routine gets automated -> innovation cycle

A SIMPLE 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

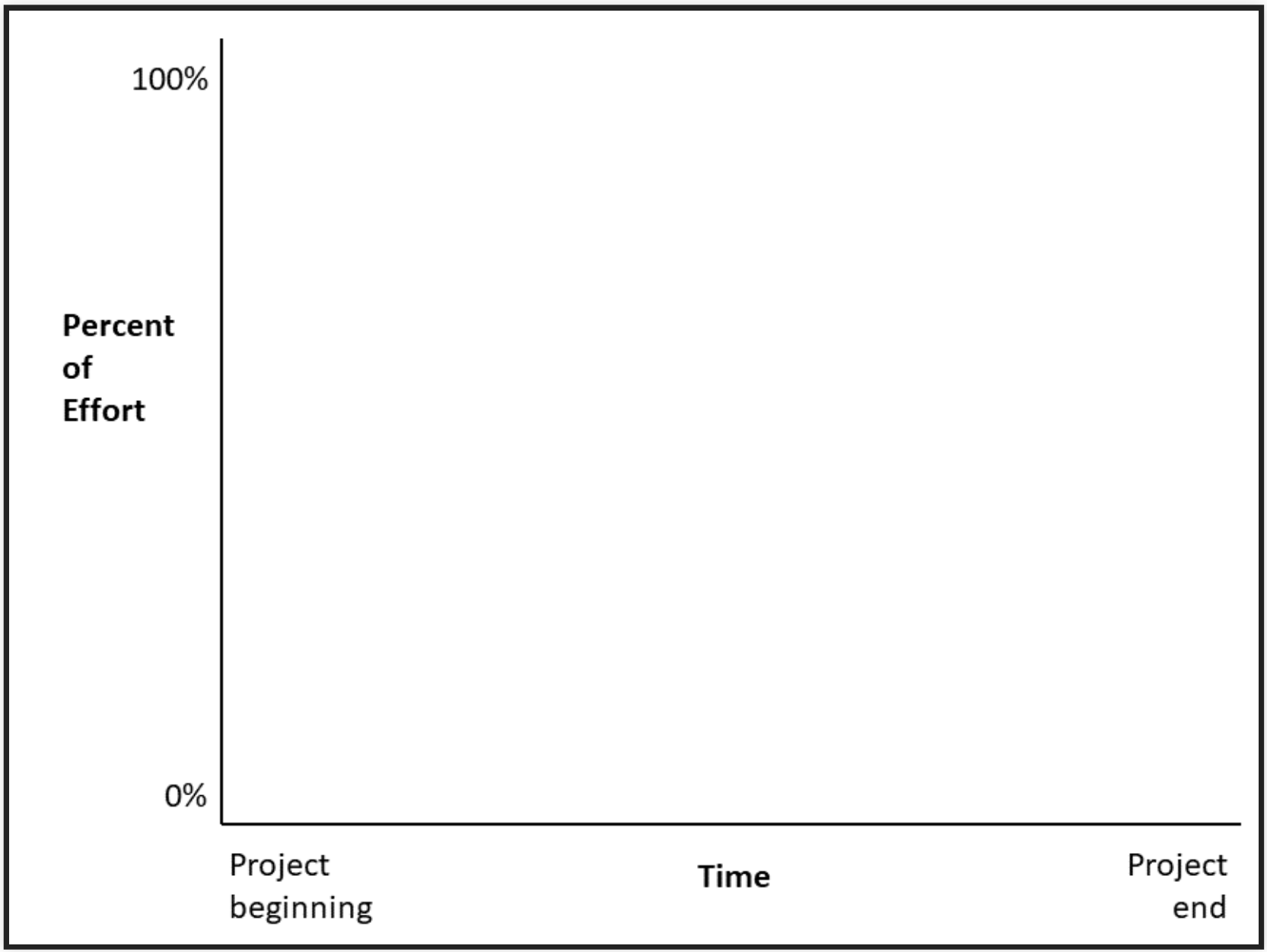
SOFTWARE PROCESS

“The set of activities and associated results that produce a software product”

Examples?

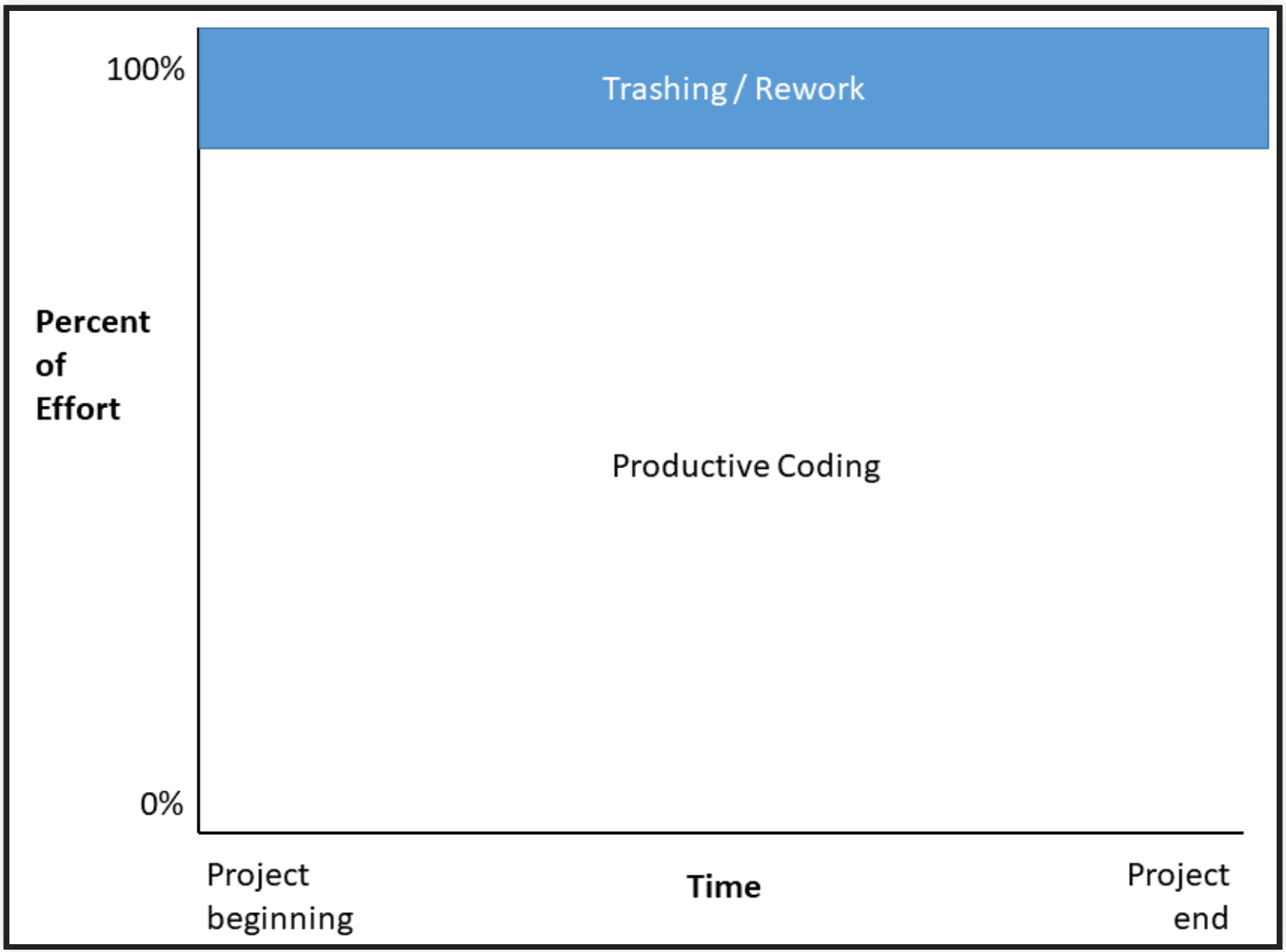
Speaker notes

Writing down all requirements Require approval for all changes to requirements Use version control for all changes Track all reported bugs Review requirements and code Break down development into smaller tasks and schedule and monitor them Planning and conducting quality assurance Have daily status meetings Use Docker containers to push code between developers and operation


100%
Percent of Effort
0%
Project beginning
Time
Project end

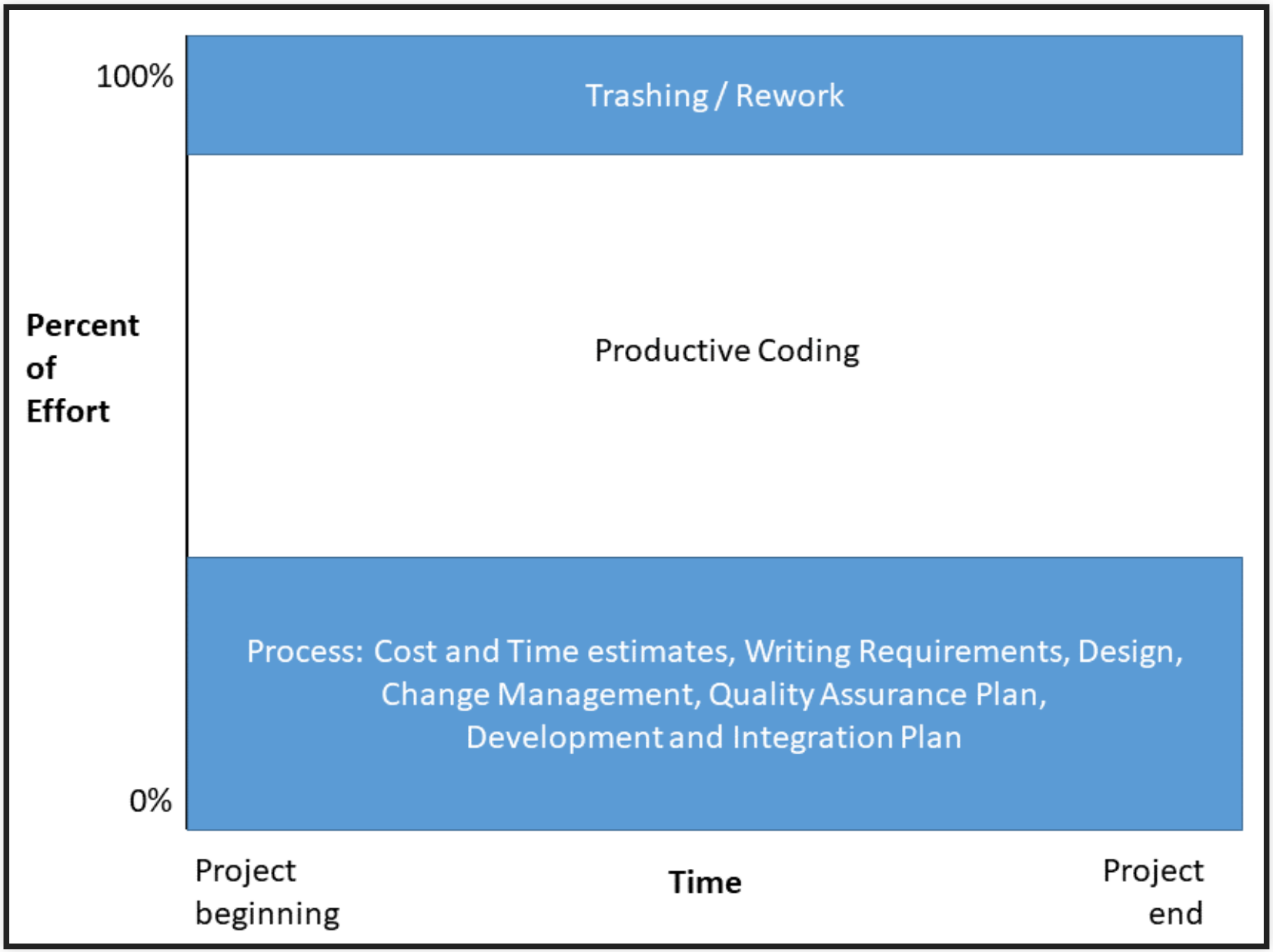
Speaker notes

Visualization following McConnell, Steve. Software project survival guide. Pearson Education, 1998.


100%
Trashing / Rework
Percent of Effort
Productive Coding
0%
Project beginning
Time
Project end

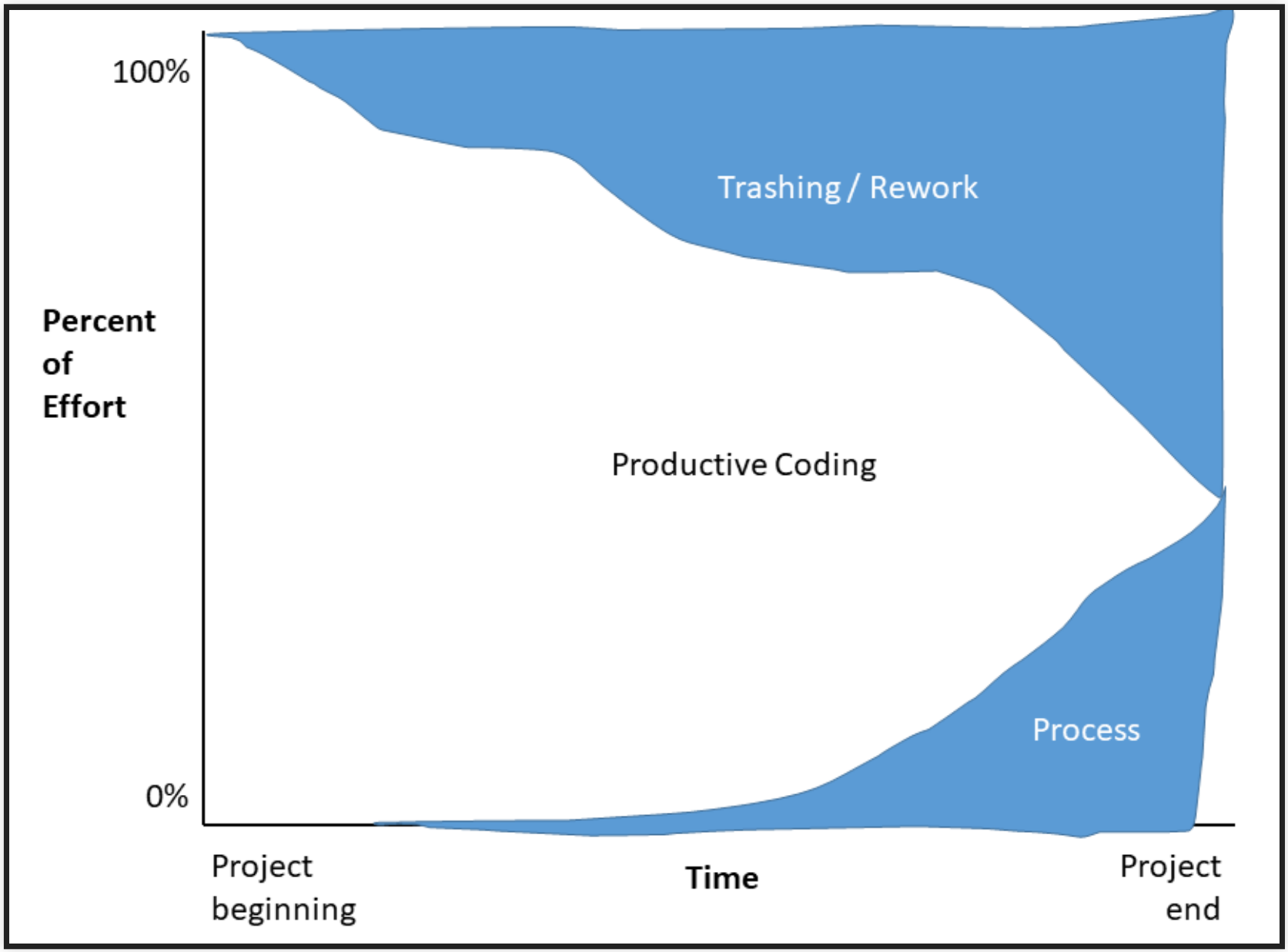
Speaker notes

Idea: spent most of the time on coding, accept a little rework


100%
Trashing / Rework
Percent of Effort
Productive Coding
Process: Cost and Time estimates, Writing Requirements, Design, Change Management, Quality Assurance Plan, Development and Integration Plan
0%
Project beginning
Time
Project end

Speaker notes

negative view of process. pure overhead, reduces productive work, limits creativity


100%
Trashing / Rework
Percent of Effort
Productive Coding
Process
0%
Project beginning
Time
Project end

Speaker notes

Real experience if little attention is payed to process: increasingly complicated, increasing rework; attempts to rescue by introducing process

EXAMPLE OF PROCESS PROBLEMS?

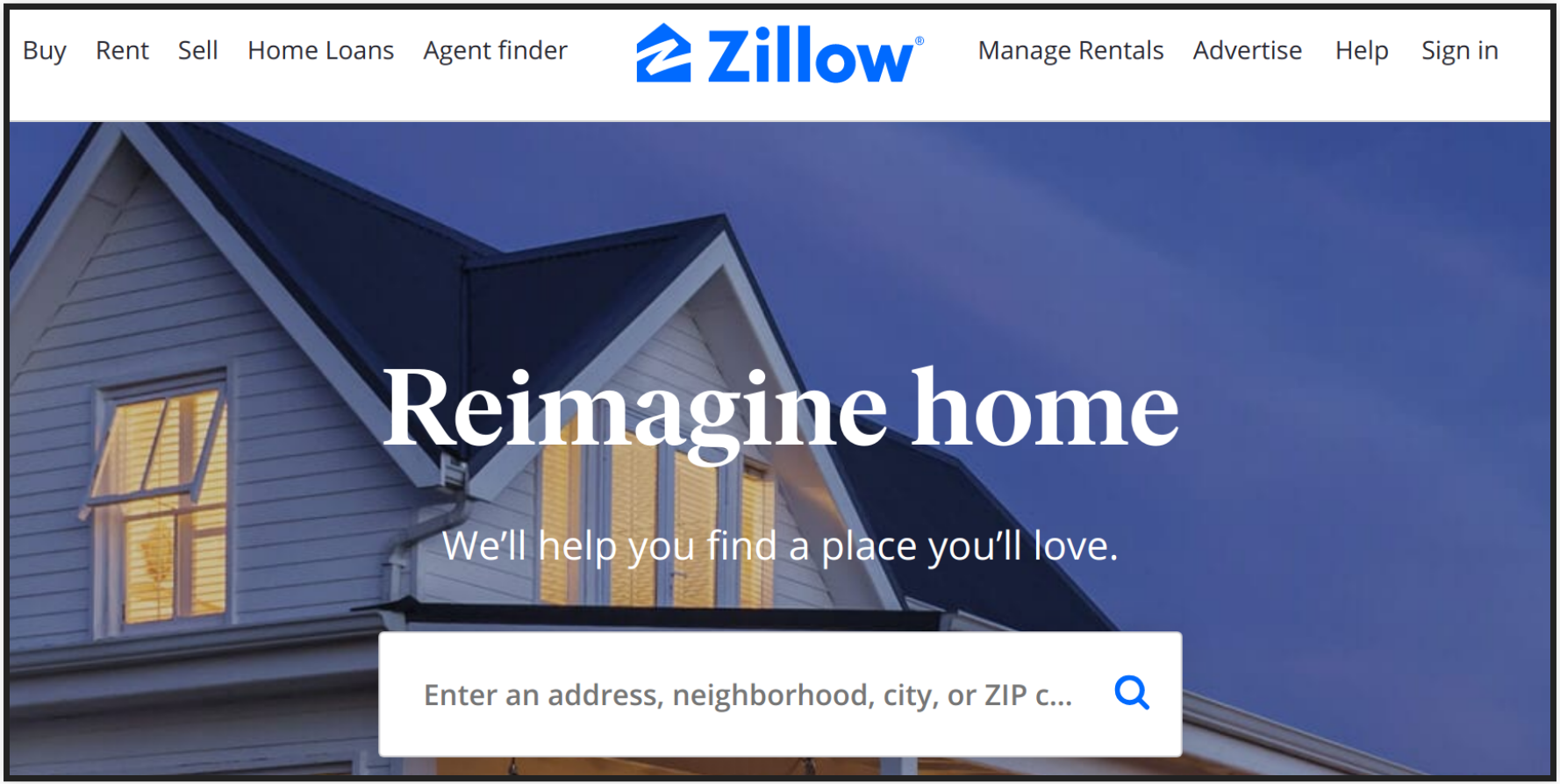
Speaker notes

Collect examples of what could go wrong:

Change Control: 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, forgotten System Integration: Integration of independently developed components at the very 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.

TYPICAL PROCESS STEPS (NOT NECESSARILY IN THIS ORDER)

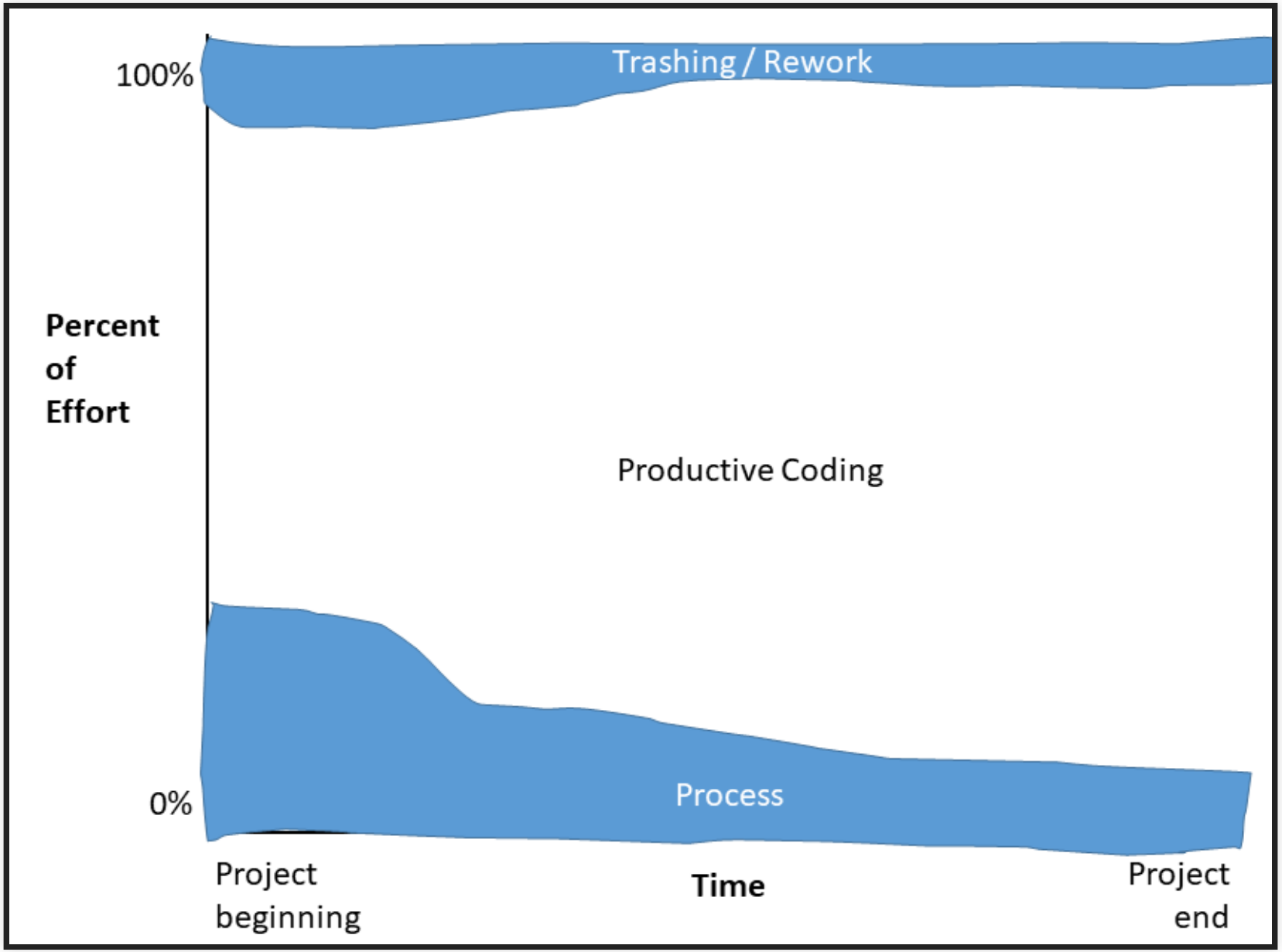
- Understand customers, identify what to build, by when, budget
- Identify relevant qualities, plan/design system accordingly
- Test, deploy, maintain, evolve
- Plan, staff, workaround


Buy Rent Sell Home Loans Agent finder
Zillow
Manage Rentals Advertise Help Sign in
Reimagine home
We'll help you find a place you'll love.
Enter an address, neighborhood, city, or ZIP c...

SURVIVAL MODE

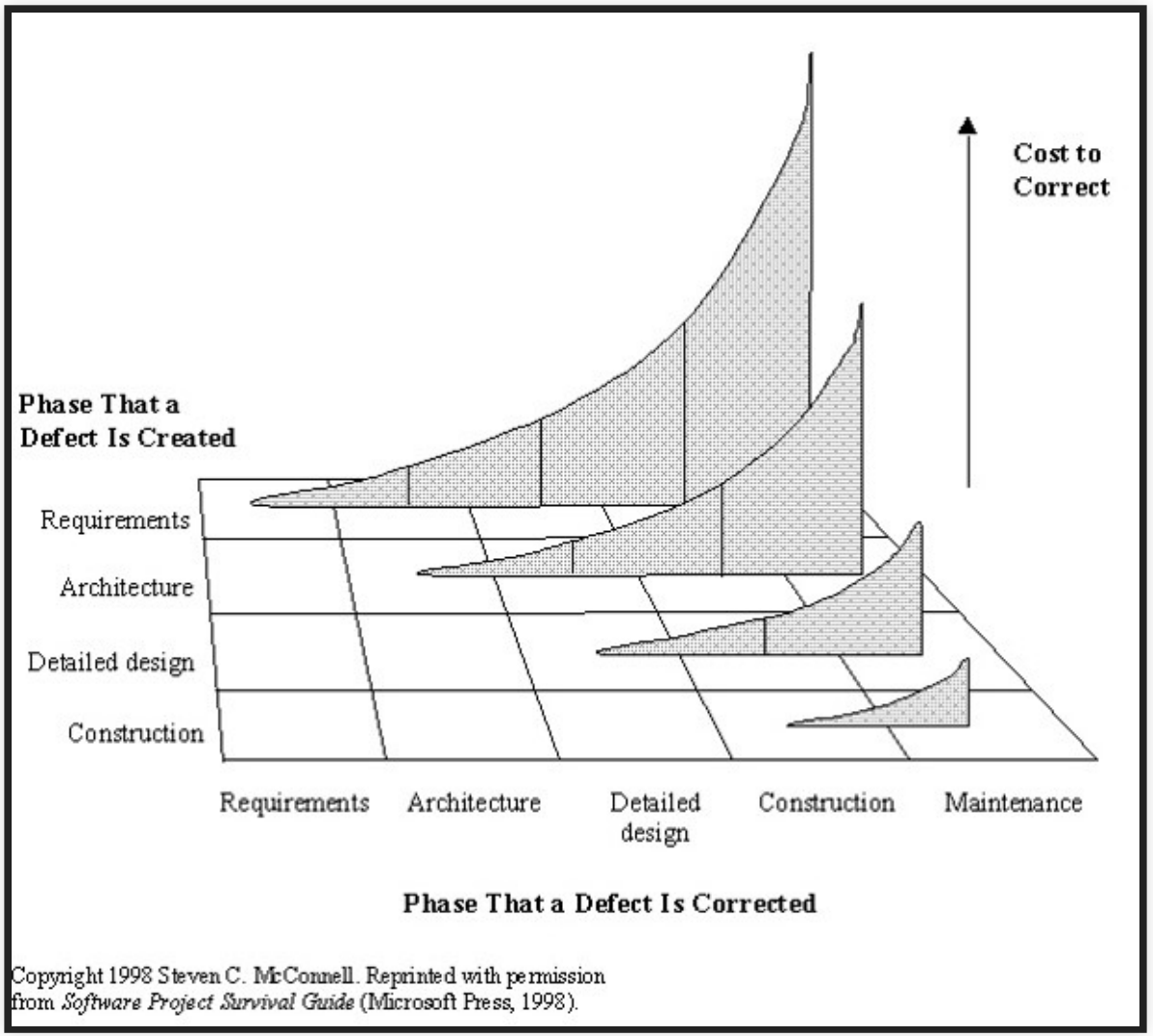
- Missed deadlines -> "solo development mode" to meet own deadlines
- Ignore integration work
- Stop interacting with testers, technical writers, managers, ...

Hypothesis: Process increases flexibility and efficiency + Upfront investment for later greater returns


100%
Trashing / Rework
Percent of Effort
Productive Coding
0%
Process
Project beginning
Time
Project end

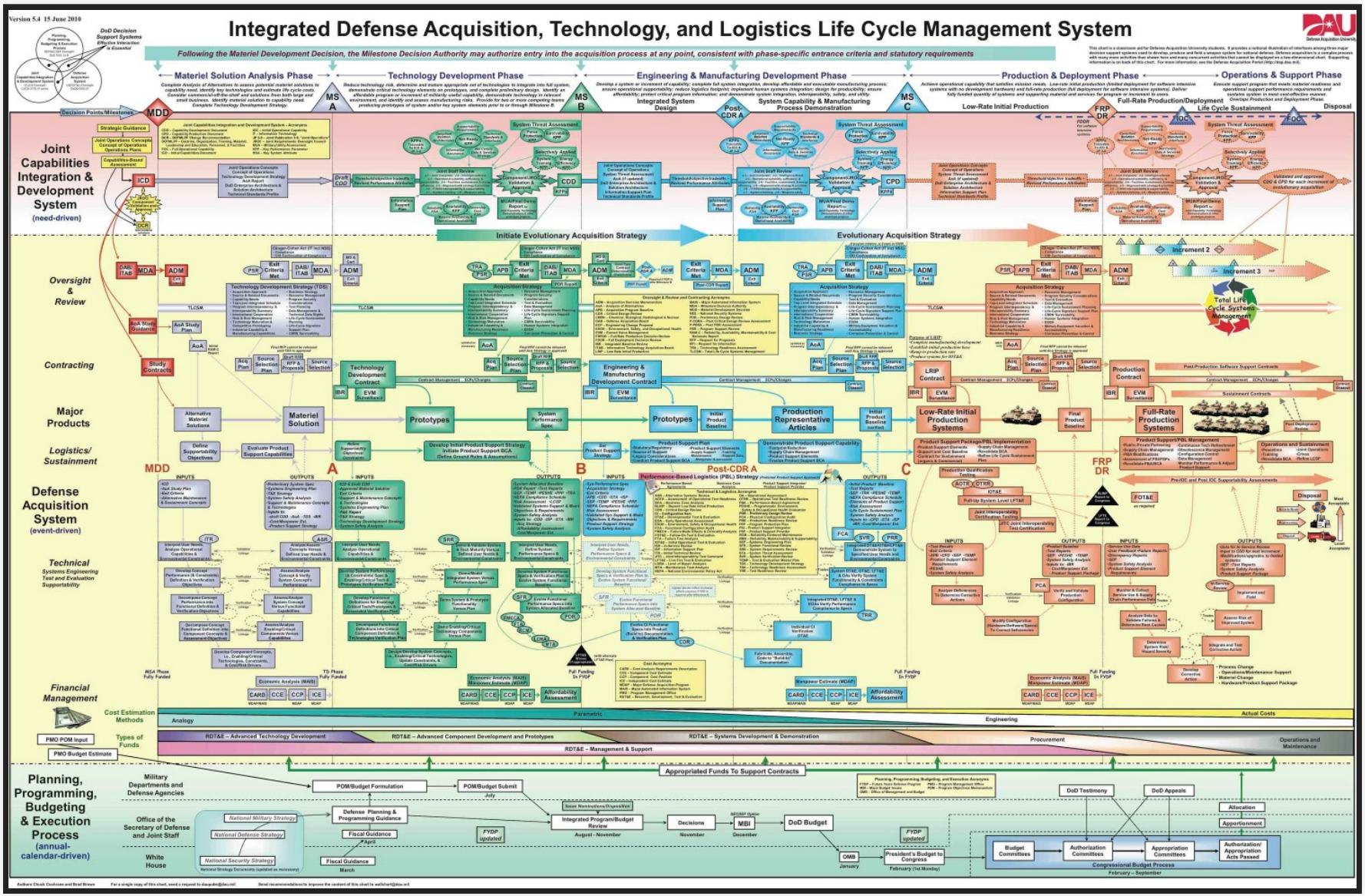
Speaker notes

ideal setting of little process investment upfront



Speaker notes

Empirically well established rule: Bugs are increasingly expensive to fix the larger the distance between the phase where they are created vs where they are corrected.


Version 5.4 15 June 2010
Integrated Defense Acquisition, Technology, and Logistics Life Cycle Management System
DoD Decision Support Systems Effective Interaction is Essential
Following the Materiel Development Decision, the Milestone Decision Authority may authorize entry into the acquisition process at any point, consistent with phase-specific entrance criteria and statutory requirements
Materiel Solution Analysis Phase
Technology Development Phase
Engineering & Manufacturing Development Phase
Production & Deployment Phase
Operations & Support Phase
MS A
MS B
MS C
Post-CDR A
FRP DR
IOC
FOC
Decision Points/Milestones
MDD
Low-Rate Initial Production
Full-Rate Production/Deployment
Life Cycle Sustainment
Disposal
Joint Capabilities Integration & Development System (need-driven)
Oversight & Review
Contracting
Major Products
Logistics/ Sustainment
Defense Acquisition System (event-driven)
Technical Systems Engineering Test and Evaluation Supportability
Financial Management
Initiate Evolutionary Acquisition Strategy
Evolutionary Acquisition Strategy
Increment 2
Increment 3
Total Life Cycle Systems Management
Post-CDR A
Pre-IOC and Post IOC Supportability Assessments
Cost Estimation Methods
Analogy
Parametric
Engineering
Actual Costs
Types of Funds
RDT&E – Advanced Technology Development
RDT&E – Advanced Component Development and Prototypes
RDT&E – Systems Development & Demonstration
RDT&E – Management & Support
Procurement
Operations and Maintenance
PMO POM Input
PMO Budget Estimate
Appropriated Funds To Support Contracts
Planning, Programming, Budgeting & Execution Process (annual-calendar-driven)
Military Departments and Defense Agencies
Office of the Secretary of Defense and Joint Staff
White House
POM/Budget Formulation
POM/Budget Submit
July
Defense Planning & Programming Guidance
Fiscal Guidance
April
Fiscal Guidance
March
Issue Nominations/Disposition
Integrated Program/Budget Review
August - November
Decisions
November
MBI
December
DoD Budget
January
OMB
President's Budget to Congress
February (1st Monday)
FYDP updated
Budget Committees
Authorization Committees
Appropriation Committees
Authorization/ Appropriation Acts Passed
Congressional Budget Process
February - September
DoD Testimony
DoD Appeals
Allocation
Apportionment
Authors Chuck Cochrane and Brad Brown
For a single copy of this chart, send a request to dacpubs@dau.mil
Send recommendations to improve the content of this chart to walchart@dau.mil

Speaker notes

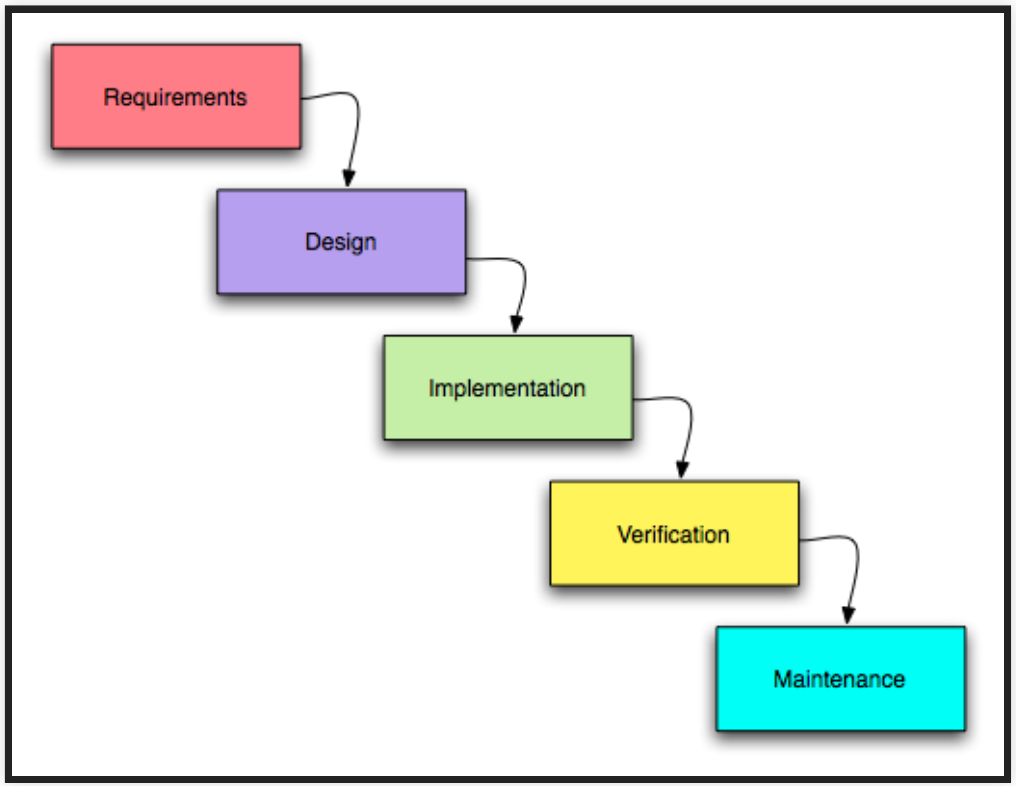
Complicated processes like these are often what people associate with "process". Software process is needed, but does not need to be complicated.

SOFTWARE PROCESS MODELS

AD-HOC PROCESSES

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

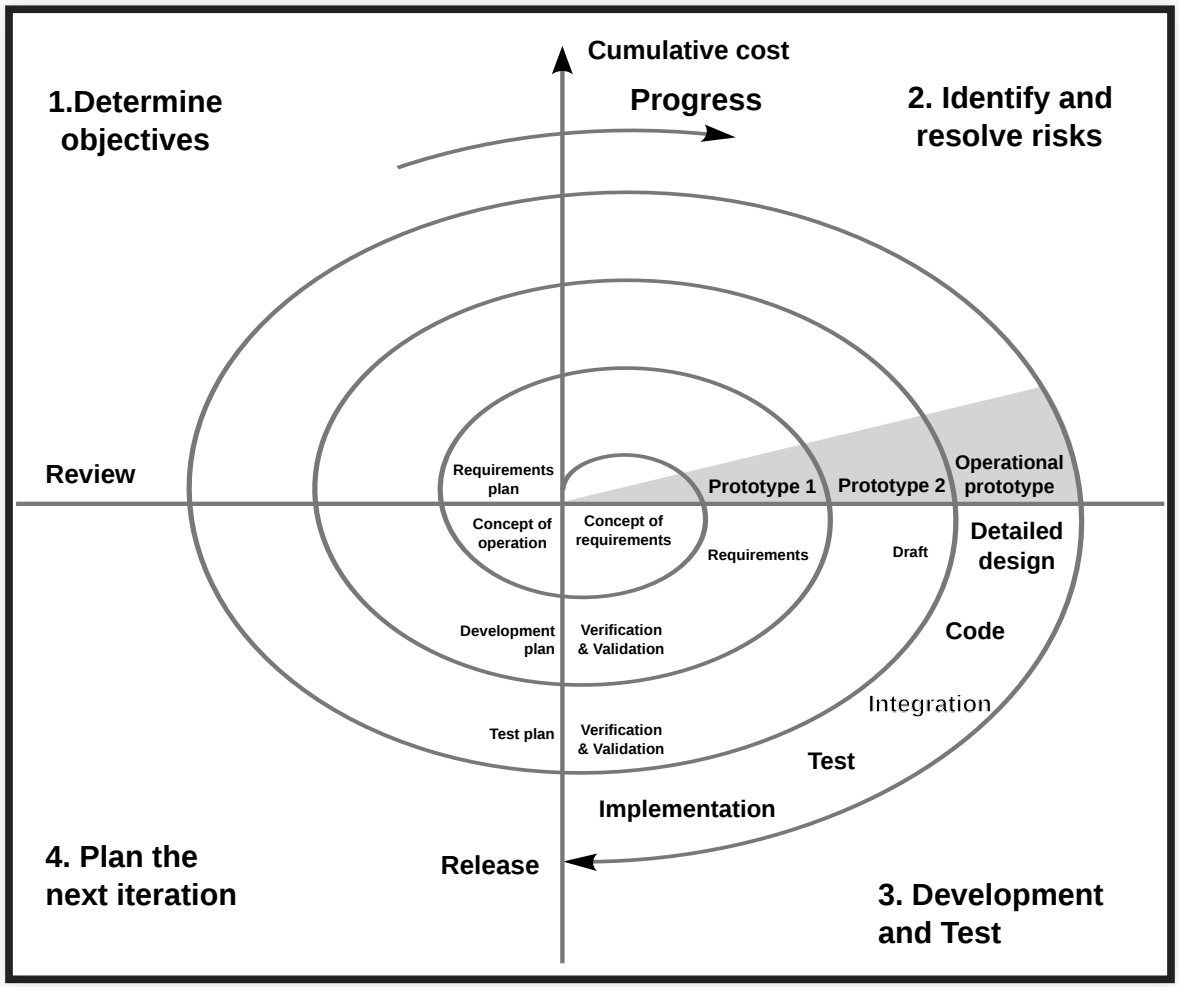


taming the chaos, understand requirements, plan before coding, remember testing

Speaker notes

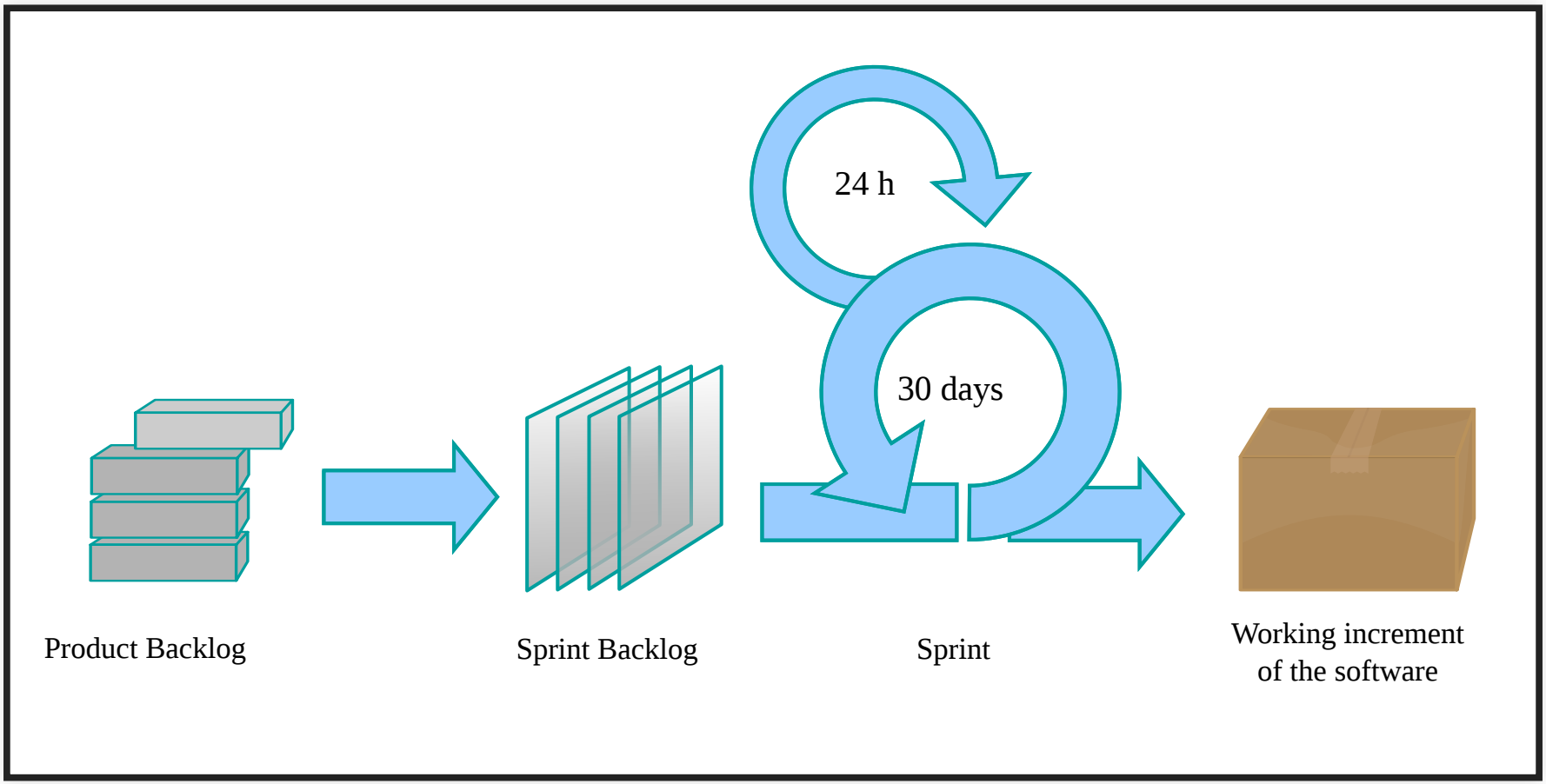
Although dated, the key idea is still essential -- think and plan before implementing. Not all requirements and design can be made upfront, but planning is usually helpful.

RISK FIRST: SPIRAL MODEL



incremental prototypes, starting with most risky components

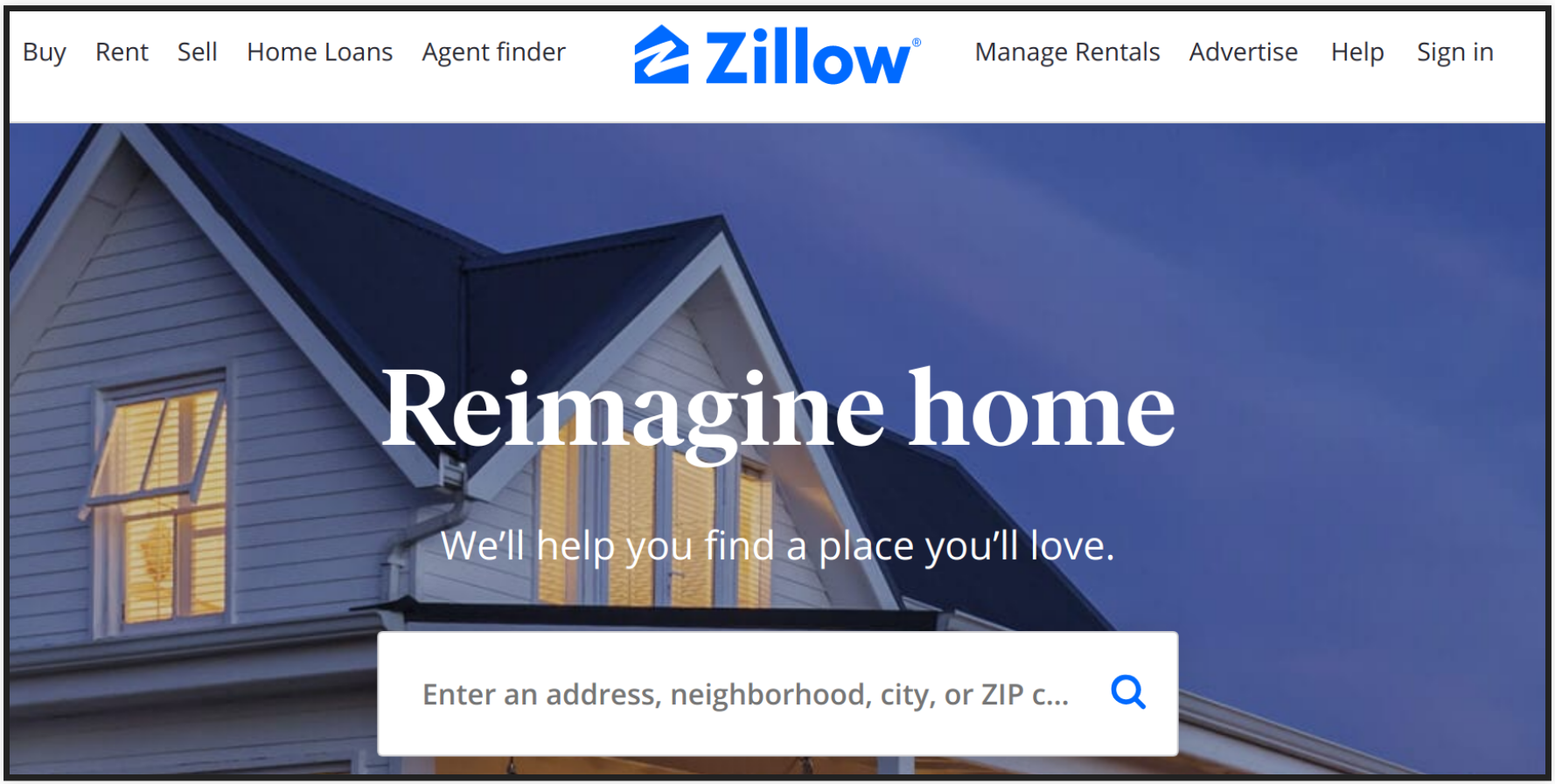
CONSTANT ITERATION: AGILE



working with customers, constant replanning

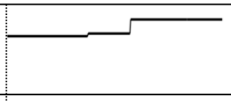
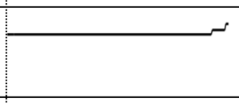
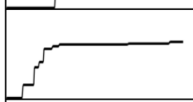
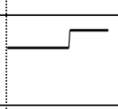
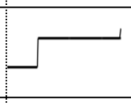
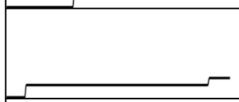
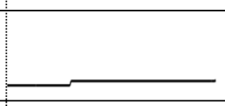
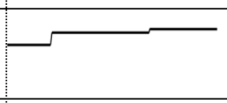
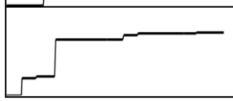
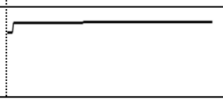
CONTRASTING PROCESS MODELS

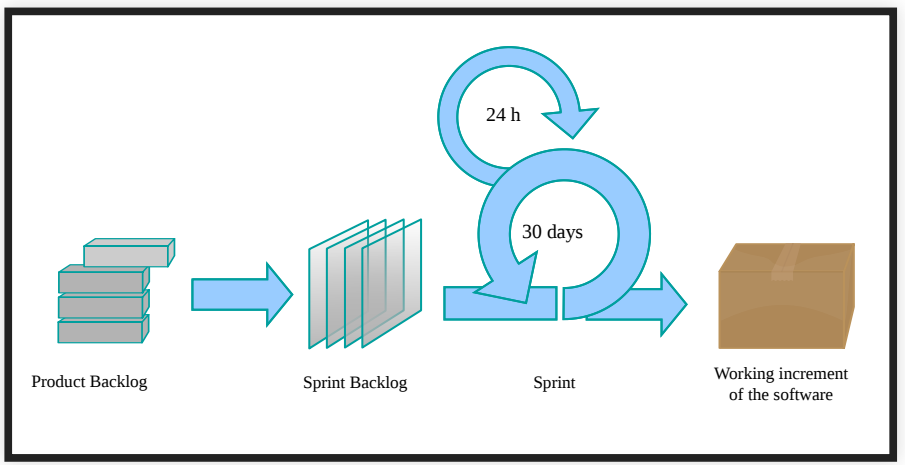
Ad-hoc -- Waterfall -- Spiral -- Agile



DATA SCIENCE VS SOFTWARE ENGINEERING

DISCUSSION: ITERATION IN NOTEBOOK VS AGILE?

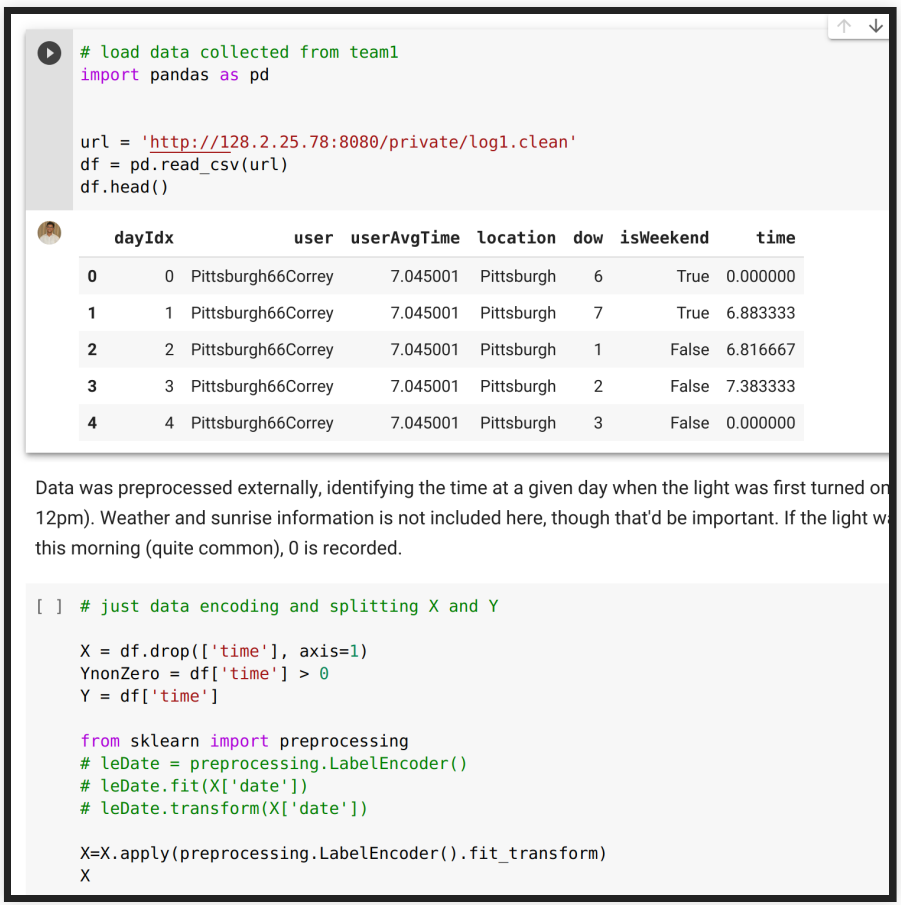
	First 2 Hours	Second 2 Hours	Final Accuracy
TAP1			84.7%
TAP2	X	X	75.3%
TAP3			78.3%
TAP4			82.9%
TAP5			84.7%
TAP6			78.0%
TAP7			56.9%
TAP8			22.8%
TAP9			78.8%
TAP10			84.4%



Speaker notes

There is similarity in that there is an iterative process, but the idea is different and the process model seems mostly orthogonal to iteration in data science. The spiral model prioritizes risk, especially when it is not clear whether a model is feasible. One can do similar things in model development, seeing whether it is feasible with data at hand at all and build an early prototype, but it is not clear that an initial okay model can be improved incrementally into a great one later. Agile can work with vague and changing requirements, but that again seems to be a rather orthogonal concern. Requirements on the product are not so much unclear or changing (the goal is often clear), but it's not clear whether and how a model can solve it.

POOR SOFTWARE ENGINEERING PRACTICES IN NOTEBOOKS?



```
# load data collected from team1
import pandas as pd


url = 'http://128.2.25.78:8080/private/log1.clean'
df = pd.read_csv(url)
df.head()
```

	dayIdx	user	userAvgTime	location	dow	isWeekend	time
0	0	Pittsburgh66Correy	7.045001	Pittsburgh	6	True	0.000000
1	1	Pittsburgh66Correy	7.045001	Pittsburgh	7	True	6.883333
2	2	Pittsburgh66Correy	7.045001	Pittsburgh	1	False	6.816667
3	3	Pittsburgh66Correy	7.045001	Pittsburgh	2	False	7.383333
4	4	Pittsburgh66Correy	7.045001	Pittsburgh	3	False	0.000000

Data was preprocessed externally, identifying the time at a given day when the light was first turned on
12pm). Weather and sunrise information is not included here, though that'd be important. If the light w
this morning (quite common), 0 is recorded.

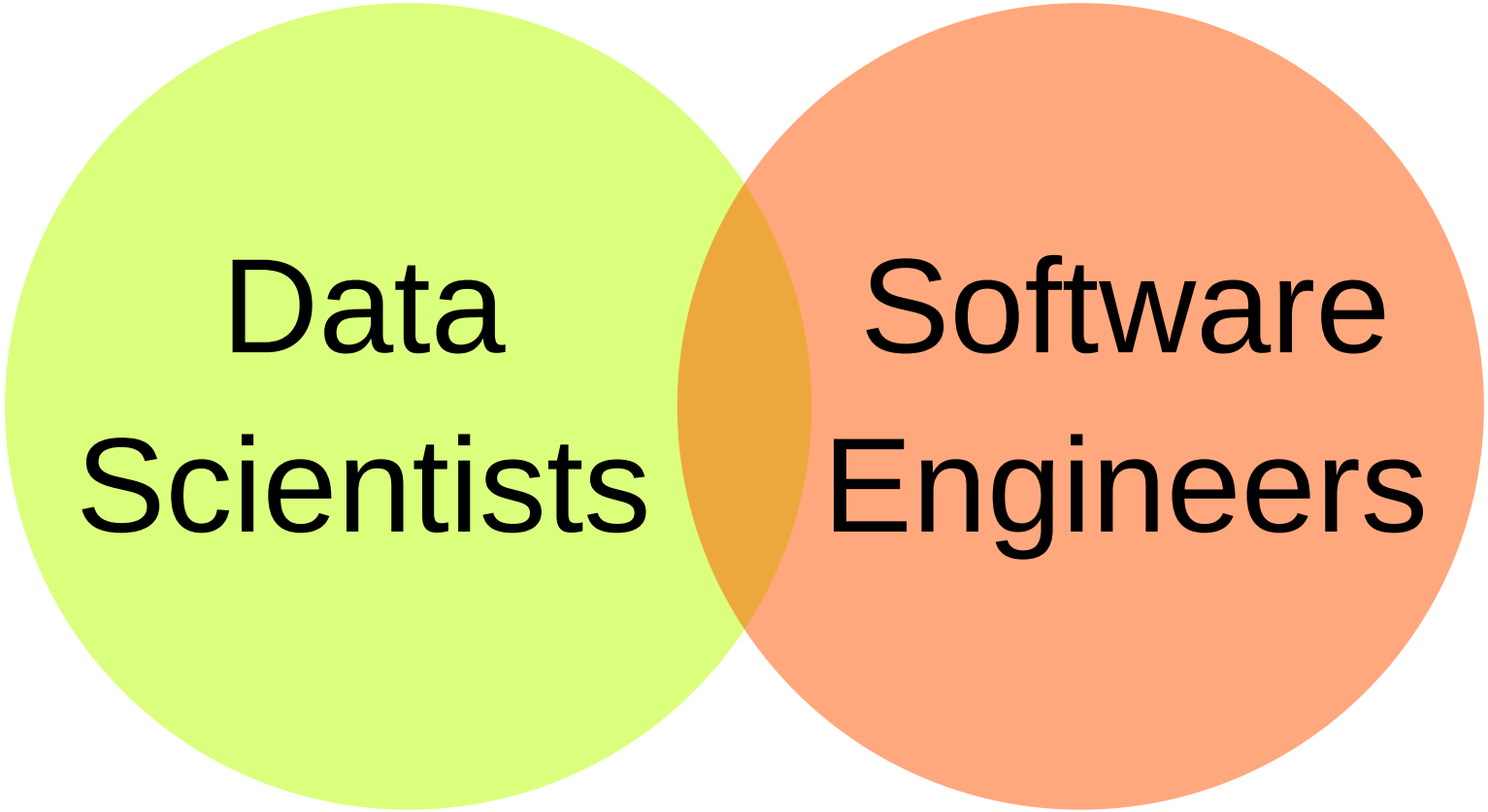
```
[ ] # just data encoding and splitting X and Y

    X = df.drop(['time'], axis=1)
    YnonZero = df['time'] > 0
    Y = df['time']

    from sklearn import preprocessing
    # leDate = preprocessing.LabelEncoder()
    # leDate.fit(X['date'])
    # leDate.transform(X['date'])

    X=X.apply(preprocessing.LabelEncoder().fit_transform)
    X
```

*

- Little abstraction
- Global state
- No testing
- Heavy copy and paste
- Little documentation
- Poor version control
- Out of order execution
- Poor development features (vs IDE)

UNDERSTANDING DATA SCIENTIST WORKFLOWS

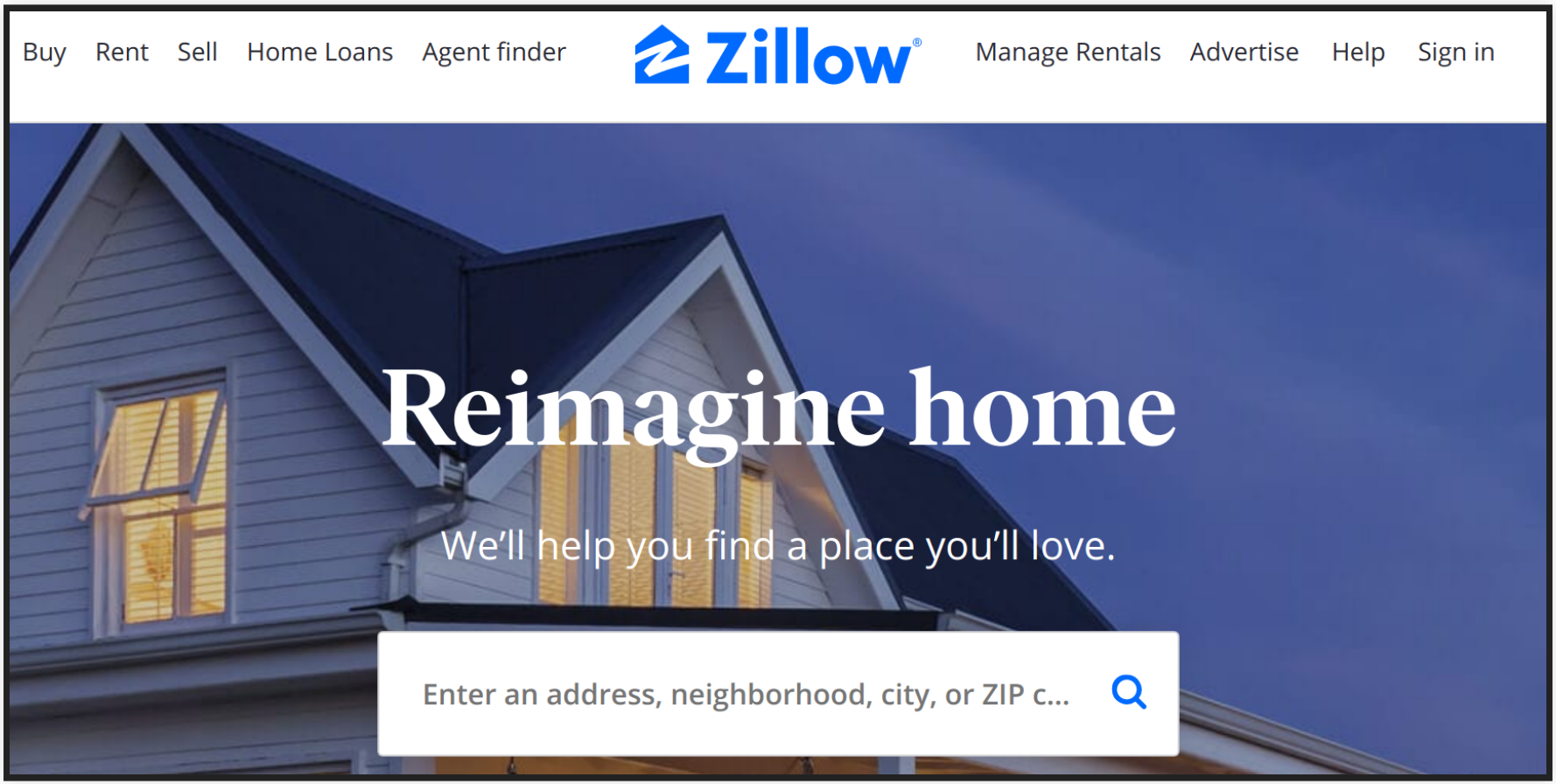
- Instead of blindly recommended "SE Best Practices" understand context
- Documentation and testing not a priority in exploratory phase
- Help with transitioning into practice
    - From notebooks to pipelines
    - Support maintenance and iteration once deployed
    - Provide infrastructure and tools


Data
Scientists
Software
Engineers

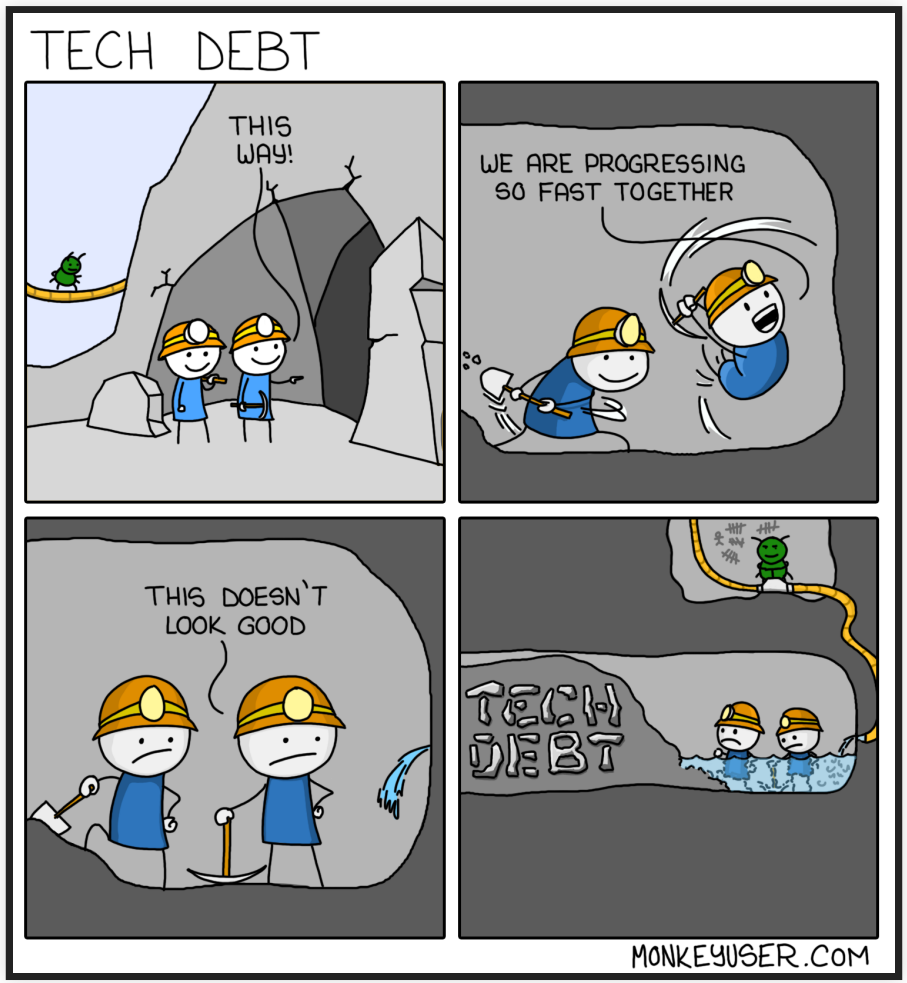
PROCESS FOR AI-ENABLED SYSTEMS

- Integrate Software Engineering and Data Science processes
- Establish system-level requirements (e.g., user needs, safety, fairness)
- Inform data science modeling with system requirements (e.g., privacy, fairness)
- Try risky parts first (most likely include ML components; ~spiral)
- Incrementally develop prototypes, incorporate user feedback (~agile)
- Provide flexibility to iterate and improve
- Design system with characteristics of AI component (e.g., UI design, safeguards)
- Plan for testing throughout the process and in production
- Manage project understanding both software engineering and data science workflows

- No existing "best practices" or workflow models


Buy Rent Sell Home Loans Agent finder
Zillow
Manage Rentals Advertise Help Sign in
Reimagine home
We'll help you find a place you'll love.
Enter an address, neighborhood, city, or ZIP c...

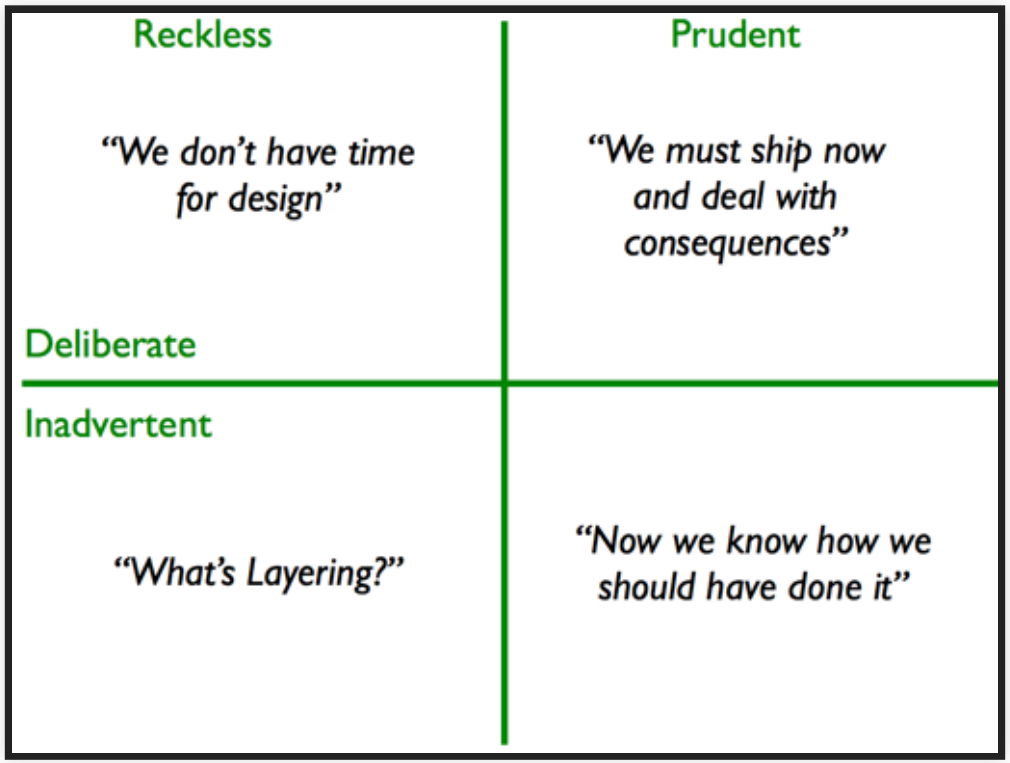
TECHNICAL DEBT



TECHNICAL DEBT METAPHOR

- Analogy to financial debt
  - Have a benefit now (e.g., progress quickly, release now)
  - accepting later cost (loss of productivity, e.g., higher maintenance/operating cost, rework)
  - debt accumulates and can suffocate project
- Ideally a deliberate decision (short term tactical or long term strategic)
- Ideally track debt and plan for paying it down

Examples?



Source: Martin Fowler 2009,
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

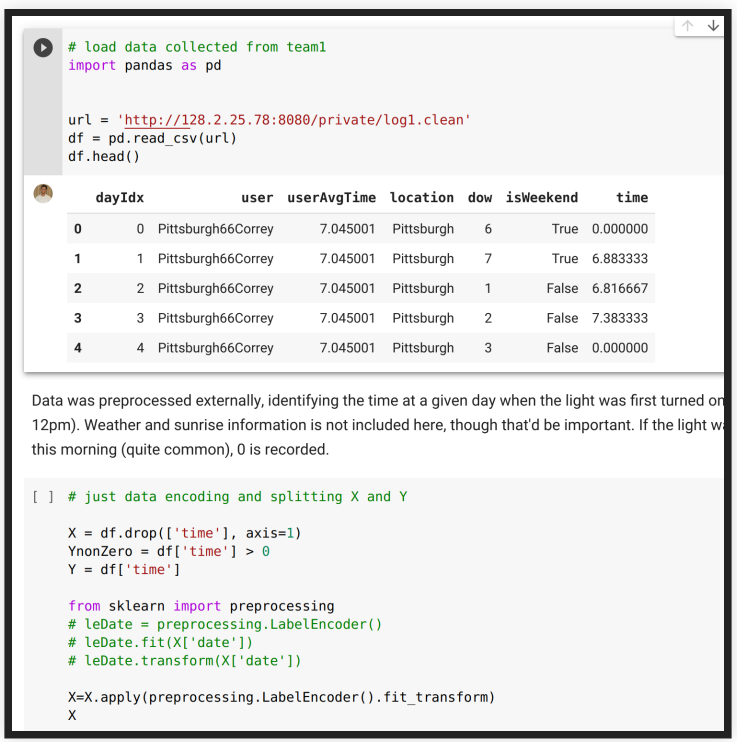
TECHNICAL DEBT FROM ML COMPONENTS?

(see reading)

Sculley, David, et al. Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems. 2015.

THE NOTEBOOK

Jupyter Notebooks are a gift from God to those who work with data. They allow us to do quick experiments with Julia, Python, R, and more -- John Paul Ada



```
# load data collected from team1
import pandas as pd


url = 'http://128.2.25.78:8080/private/log1.clean'
df = pd.read_csv(url)
df.head()
```

	dayIdx	user	userAvgTime	location	dow	isWeekend	time
0	0	Pittsburgh66Correy	7.045001	Pittsburgh	6	True	0.000000
1	1	Pittsburgh66Correy	7.045001	Pittsburgh	7	True	6.883333
2	2	Pittsburgh66Correy	7.045001	Pittsburgh	1	False	6.816667
3	3	Pittsburgh66Correy	7.045001	Pittsburgh	2	False	7.383333
4	4	Pittsburgh66Correy	7.045001	Pittsburgh	3	False	0.000000

Data was preprocessed externally, identifying the time at a given day when the light was first turned on
12pm). Weather and sunrise information is not included here, though that'd be important. If the light w
this morning (quite common), 0 is recorded.

```
[ ] # just data encoding and splitting X and Y

    X = df.drop(['time'], axis=1)
    YnonZero = df['time'] > 0
    Y = df['time']

    from sklearn import preprocessing
    # leDate = preprocessing.LabelEncoder()
    # leDate.fit(X['date'])
    # leDate.transform(X['date'])

    X=X.apply(preprocessing.LabelEncoder().fit_transform)
    X
```

Speaker notes

Discuss benefits and drawbacks of Jupyter style notebooks

ML AND TECHNICAL DEBT

- **Often reckless and inadvertent in inexperienced teams**
- ML can seem like an easy addition, but it may cause long-term costs
- Needs to be maintained, evolved, and debugged
- Goals may change, environment may change, some changes are subtle

- Example problems
  - Systems and models are tangled and changing one has cascading effects on the other
  - Untested, brittle infrastructure; manual deployment
  - Unstable data dependencies, replication crisis
  - Data drift and feedback loops
  - Magic constants and dead experimental code paths

Further reading: Sculley, David, et al. Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems. 2015.

CONTROLLING TECHNICAL DEBT FROM ML COMPONENTS

CONTROLLING TECHNICAL DEBT FROM ML COMPONENTS

- Avoid AI when not needed
- Understand and document requirements, design for mistakes
- Build reliable and maintainable pipelines, infrastructure, good engineering practices
- Test infrastructure, system testing, testing and monitoring in production
- Test and monitor data quality
- Understand and model data dependencies, feedback loops, ...
- Document design intent and system architecture
- Strong interdisciplinary teams with joint responsibilities
- Document and track technical debt
- ...

SUMMARY

- Data scientists and software engineers follow different processes
- ML projects need to consider process needs of both
- Iteration and upfront planning are both important, process models codify good practices
- Deliberate technical debt can be good, too much debt can suffocate a project
- Easy to amount (reckless) debt with machine learning