



Machine Learning in Production Process and Technical Debt

Process...

Fundamentals of Engineering AI-Enabled Systems

Holistic system view: AI and non-AI components, pipelines, stakeholders, environment interactions, feedback loops

Requirements:

System and model goals
User requirements
Environment assumptions
Quality beyond accuracy
Measurement
Risk analysis
Planning for mistakes

Architecture + design:

Modeling tradeoffs
Deployment architecture
Data science pipelines
Telemetry, monitoring
Anticipating evolution
Big data processing
Human-AI design

Quality assurance:

Model testing
Data quality
QA automation
Testing in production
Infrastructure quality
Debugging

Operations:

Continuous deployment
Contin. experimentation
Configuration mgmt.
Monitoring
Versioning
Big data
DevOps, MLOps

Teams and process: Data science vs software eng. workflows, interdisciplinary teams, collaboration points, technical debt

Responsible AI Engineering

Provenance,
versioning,
reproducibility

Safety

Security and
privacy

Fairness

Interpretability
and explainability

Transparency
and trust

Ethics, governance, regulation, compliance, organizational culture

Readings

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
- Kruchten, Philippe, Robert L. Nord, and Ipek Ozkaya. "[Technical debt: From metaphor to theory and practice](#)." IEEE Software 29, no. 6 (2012): 18-21.

Learning Goals

- Overview of common data science workflows (e.g., CRISP-DM)
 - Importance of iteration and experimentation
 - Role of computational notebooks in supporting data science workflows
- Overview of software engineering processes and lifecycles: costs and benefits of process, common process models, role of iteration and experimentation
- Contrasting data science and software engineering processes, goals and conflicts
- Integrating data science and software engineering workflows in process model for engineering AI-enabled systems with ML and non-ML components; contrasting different kinds of AI-enabled systems with data science trajectories
- Overview of technical debt as metaphor for process management; common sources of technical debt in AI-enabled systems

Case Study: Real-Estate Website

The screenshot shows the Zillow homepage. At the top, there is a navigation bar with links for "Buy", "Rent", "Sell", "Home Loans", and "Agent finder". To the right of these links is the Zillow logo, which consists of a stylized blue 'Z' icon followed by the word "Zillow" in a bold, blue, sans-serif font. To the right of the logo are links for "Manage Rentals", "Advertise", "Help", and "Sign in". Below the navigation bar is a large, dark blue photograph of a house's exterior, showing the roofline and windows. Overlaid on this image is the text "Reimagine home" in a large, white, serif font, and below it, "We'll help you find a place you'll love." in a smaller, white, sans-serif font. At the bottom of the page is a white search bar with the placeholder text "Enter an address, neighborhood, city, or ZIP c..." followed by a blue magnifying glass icon. In the bottom left corner, there is a small blue icon consisting of three horizontal lines of varying lengths.

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.

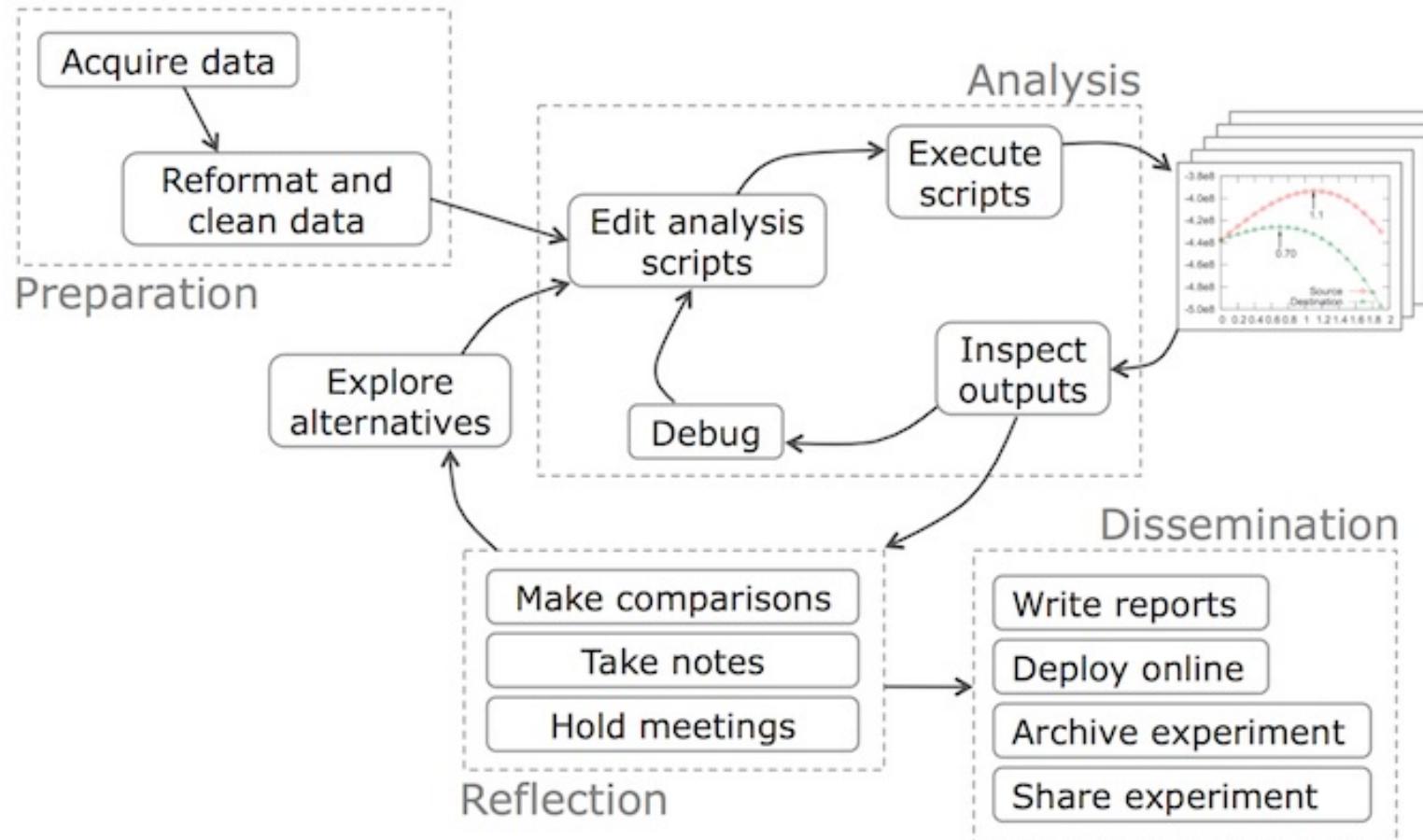
$$f(\text{size}, \text{rooms}, \text{tax}, \text{neighborhood}, \dots) \rightarrow \text{price}$$

Data Science: Iteration and Exploration

What's your process?

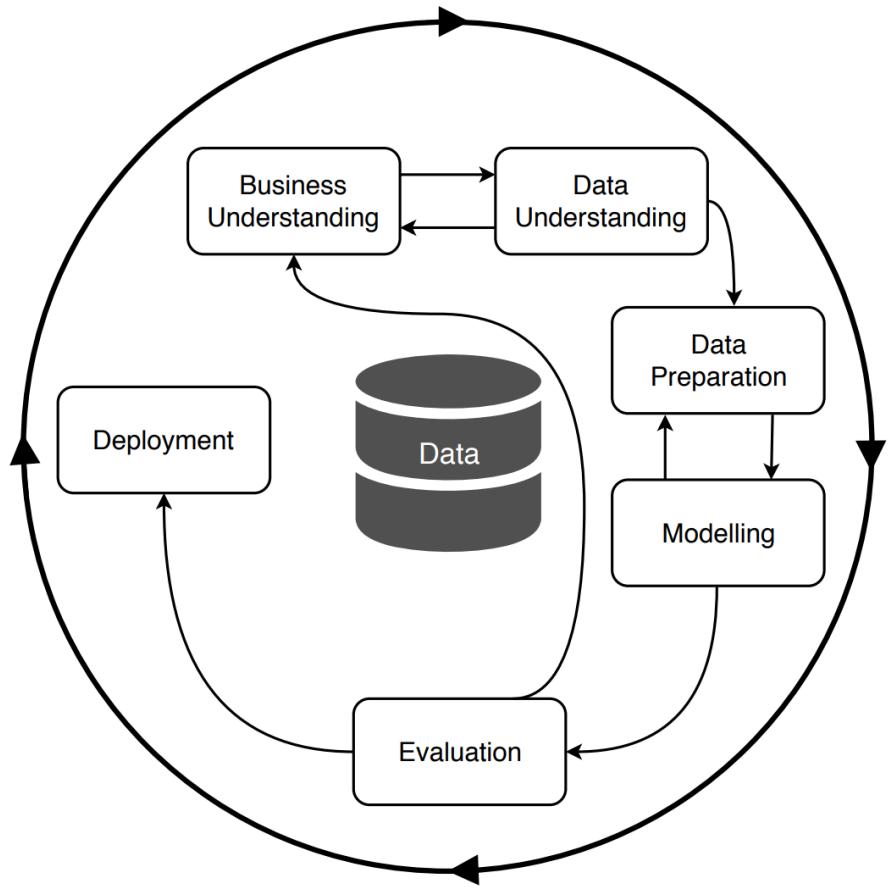


Data Science is Iterative and Exploratory



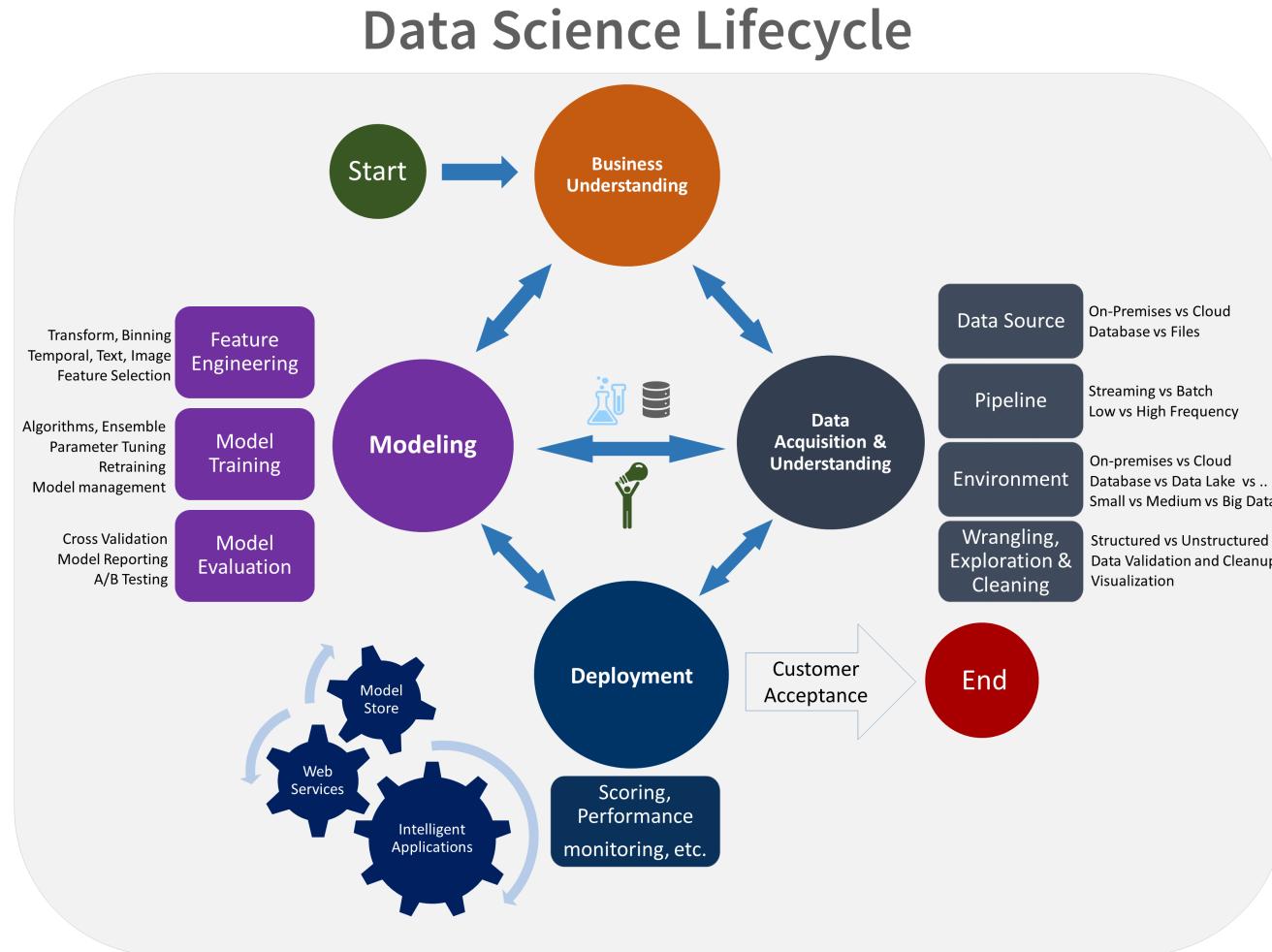
≡ Source: Guo. "Data Science Workflow: Overview and Challenges." Blog@CACM, Oct 2013

Data Science is Iterative and Exploratory



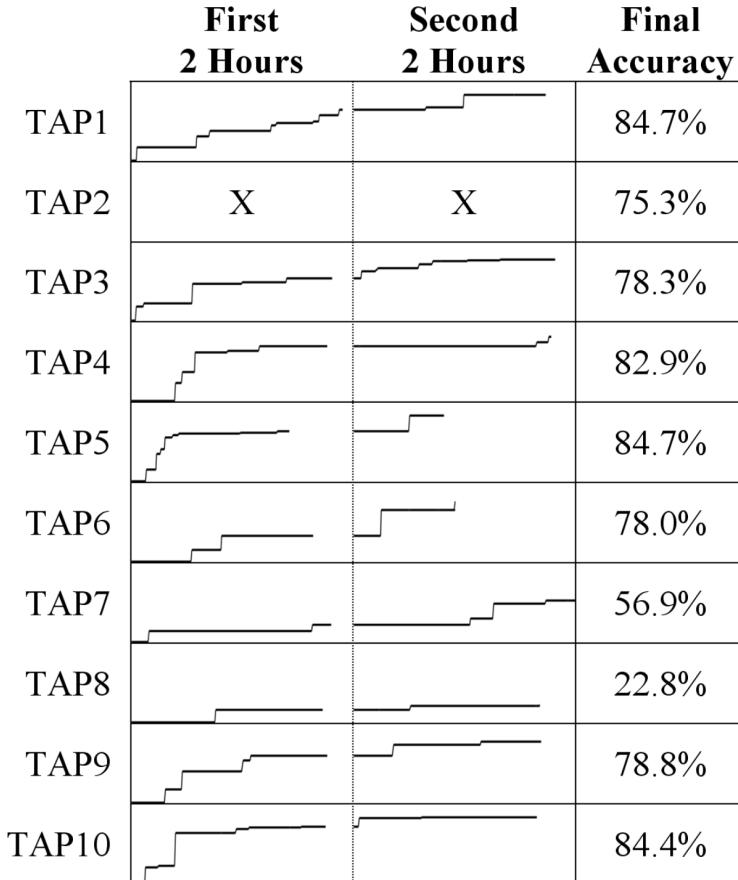
Martínez-Plumed et al. "CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories." IEEE Transactions on Knowledge and Data Engineering (2019).

Data Science is Iterative and Exploratory



Microsoft Azure Team, "What is the Team Data Science Process?" Microsoft Doc., 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

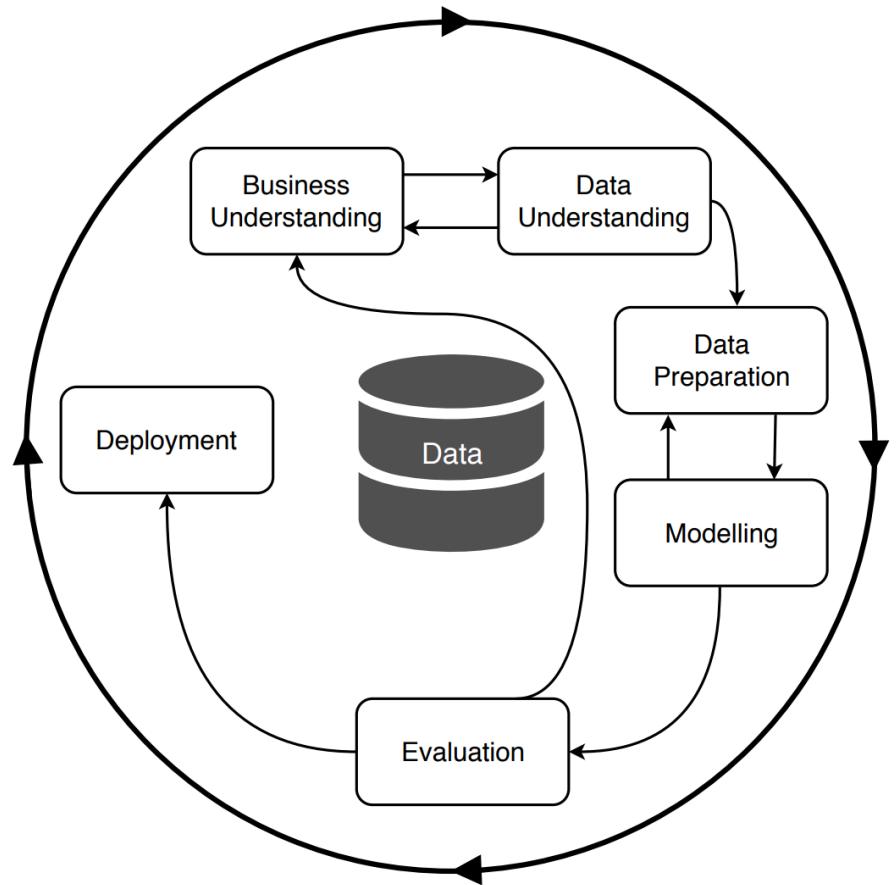
Try and error, refine iteratively, hypothesis testing

Go back to data collection and cleaning if needed, revise goals

Share Experience?

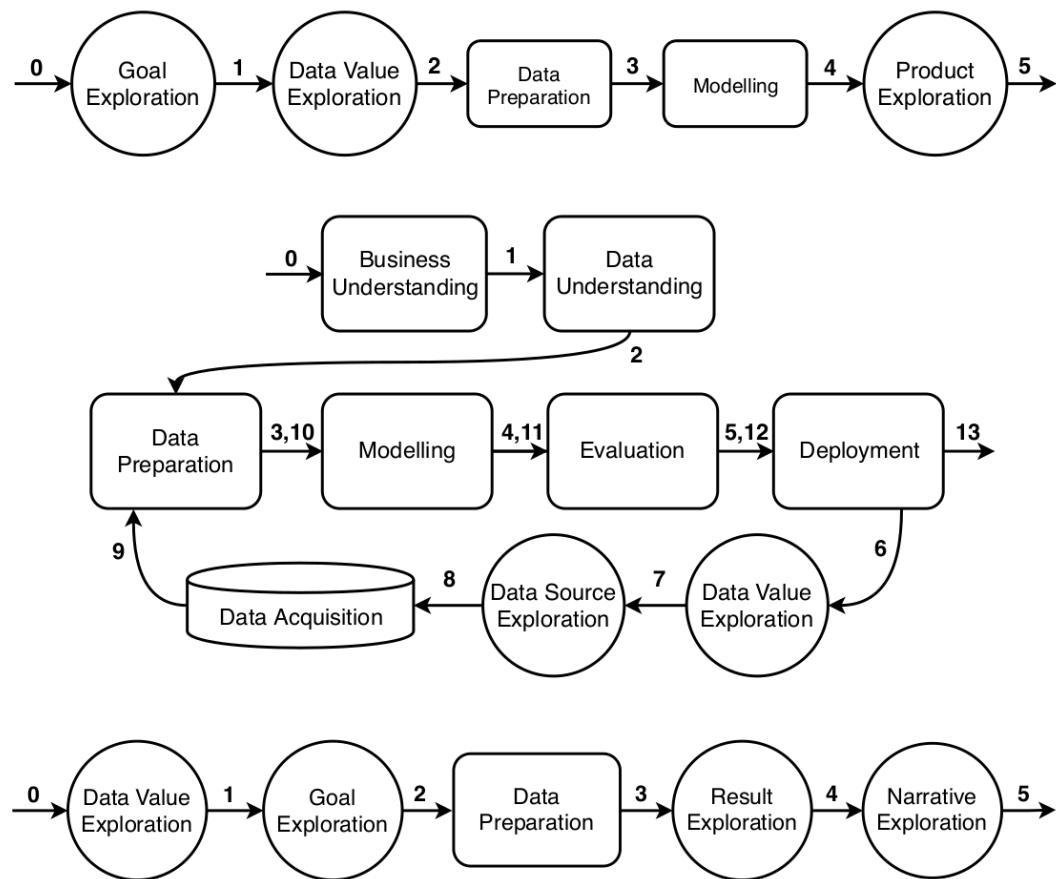


Different Trajectories



Martínez-Plumed et al. "CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories." IEEE Transactions on Knowledge and Data Engineering (2019).

Different Trajectories



From: Martínez-Plumed et al. "CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories." IEEE Transactions on Knowledge and Data Engineering (2019).

Speaker notes

- A product to recommend trips connecting tourist attractions in a town may be based on location tracking data collected by navigation and mapping apps. To build such a project, one might start with a concrete goal in mind and explore whether enough user location history data is available or can be acquired. One would then go through traditional data preparation and modeling stages before exploring how to best present the results to users.
- An insurance company tries to improve their model to score the risk of drivers based on their behavior and sensors in their cars. Here an existing product is to be refined and a better understanding of the business case is needed before diving into the data exploration and modeling. The team might spend significant time in exploring new data sources that may provide new insights and may debate the cost and benefits of this data or data gathering strategy (e.g., installing sensors in customer cars).
- A credit card company may want to sell data about what kind of products different people (nationalities) tend to buy at different times and days in different locations to other companies (retailers, restaurants). They may explore existing data without yet knowing what kind of data may be of interest to what kind of customers. They may actively search for interesting narratives in the data, posing questions such as “Ever wondered when the French buy their food?” or “Which places the Germans flock to on their holidays?” in promotional material.



Computational Notebooks

- Origins in "literate 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

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

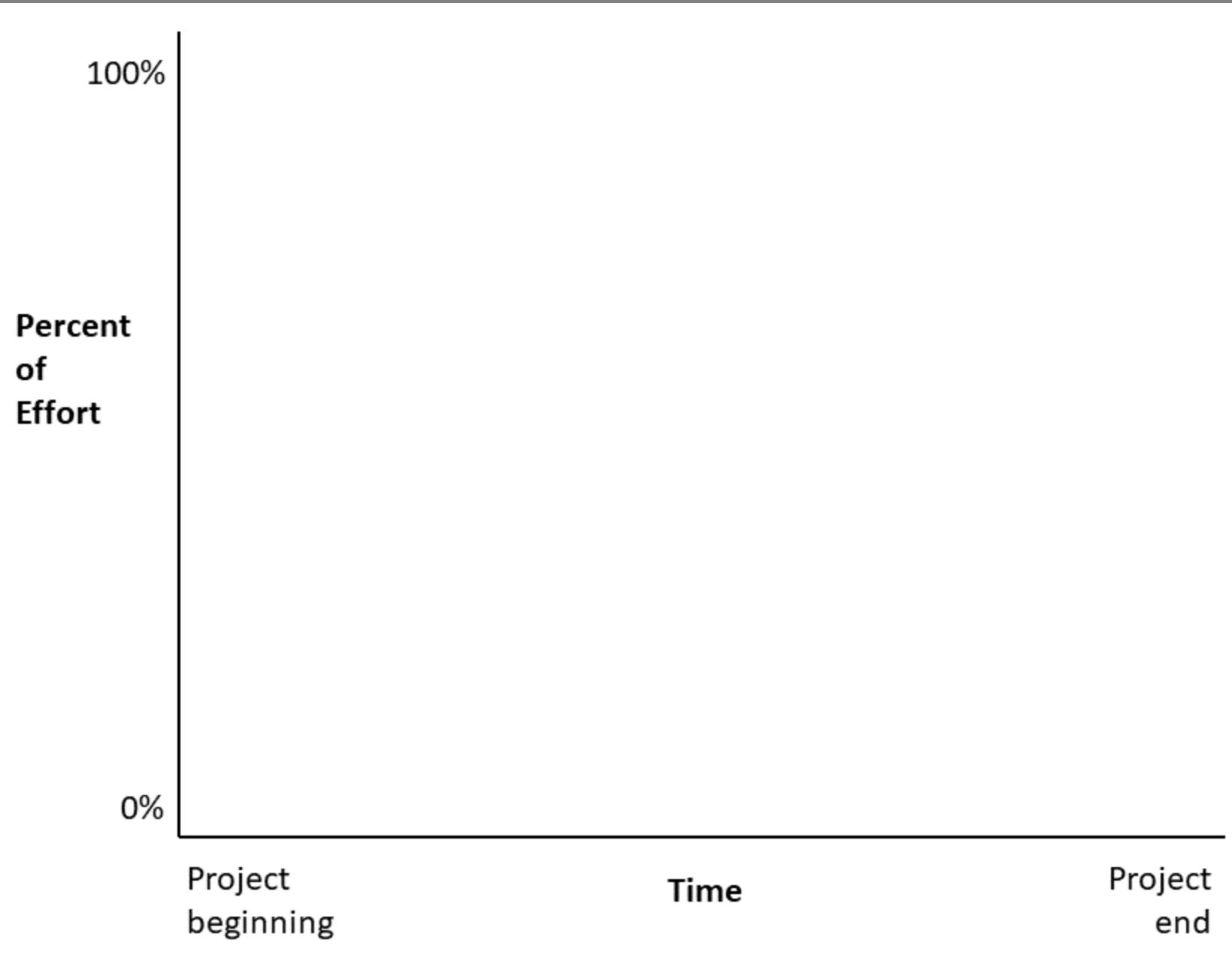
“The set of activities and associated results that produce a software product”

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

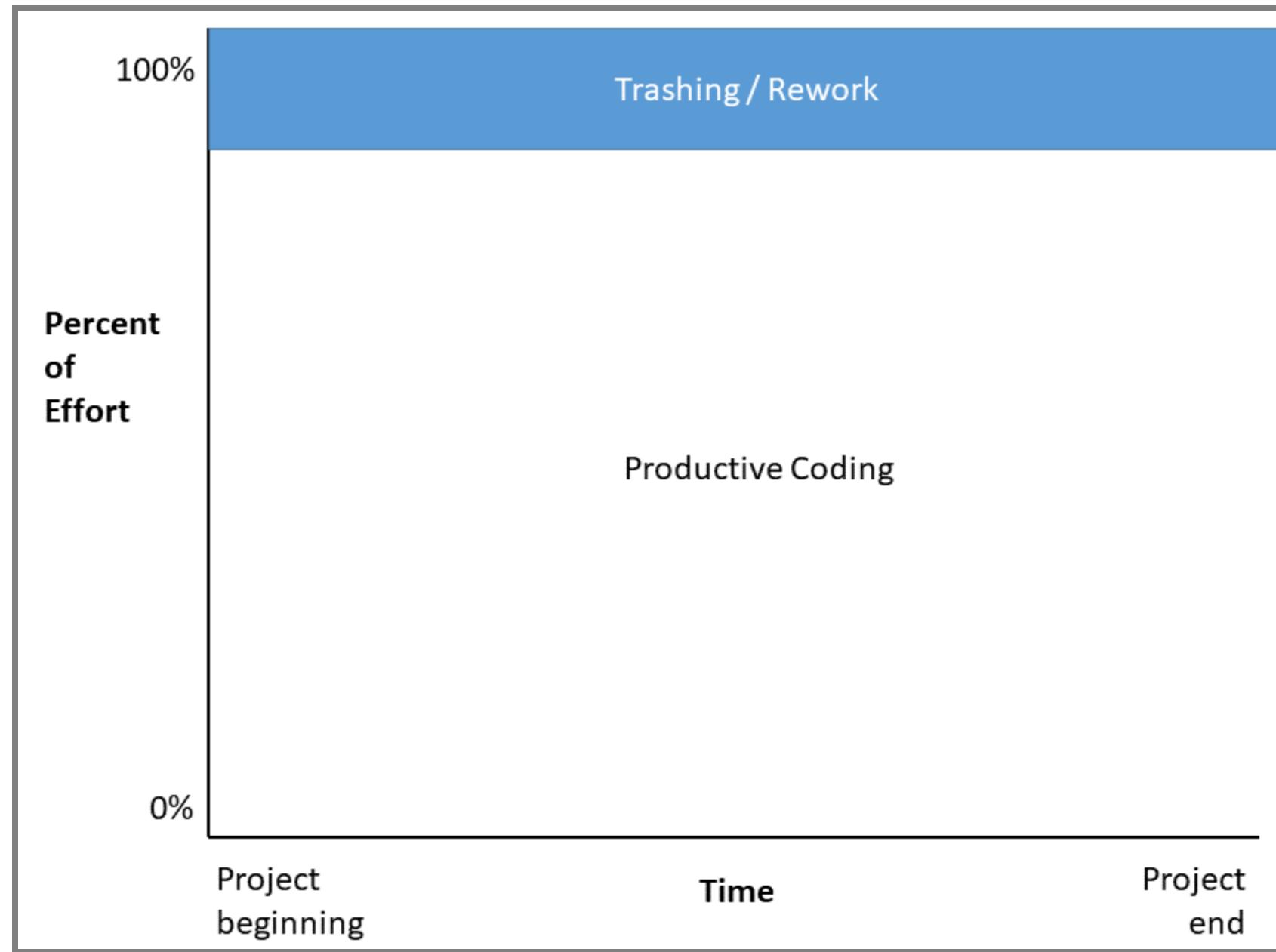




Speaker notes

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

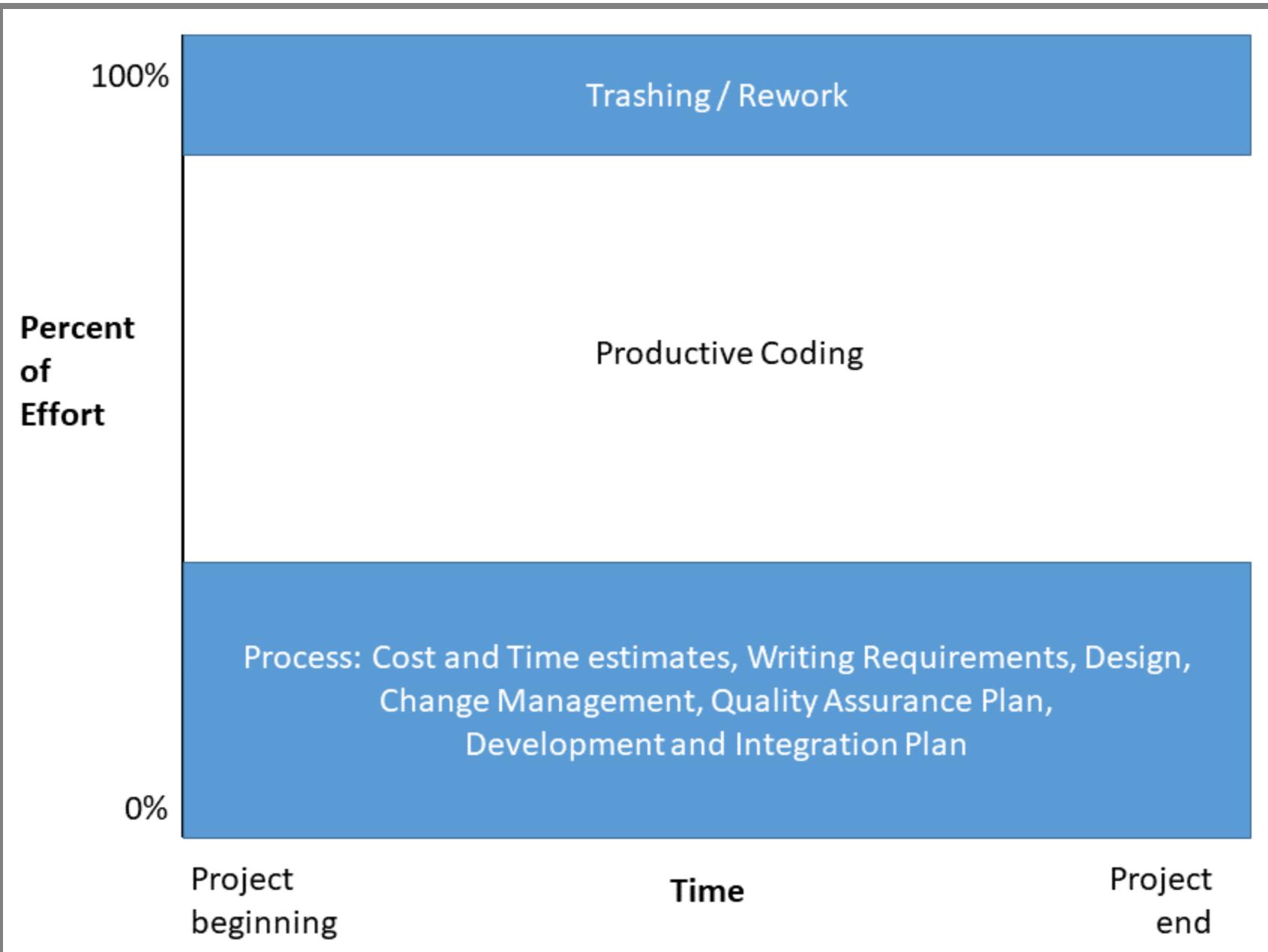




Speaker notes

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

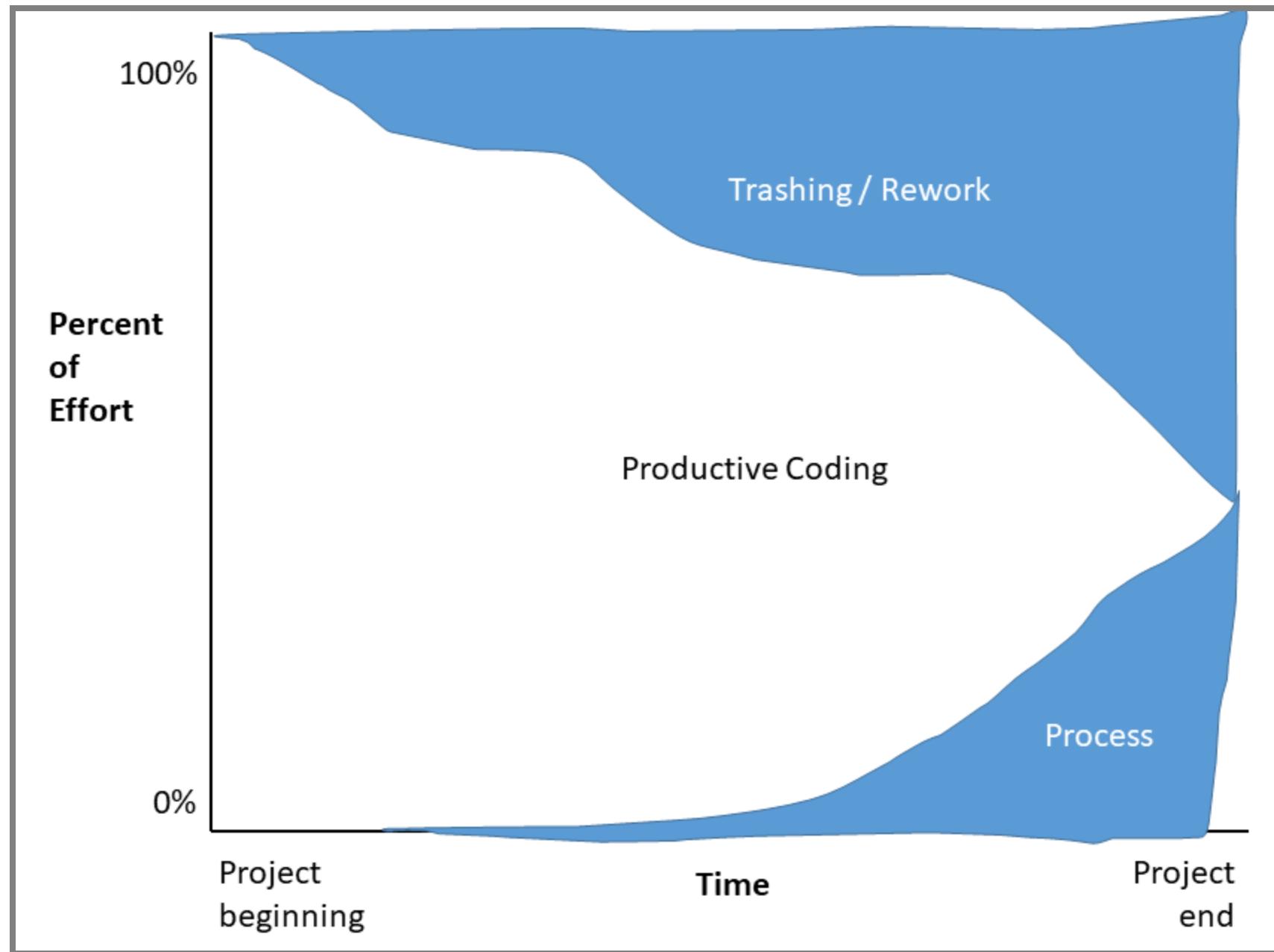




Speaker notes

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





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

Survival Mode

Missed deadlines -> "solo development mode" to meet own deadlines

Ignore integration work

Stop interacting with testers, technical writers, managers, ...



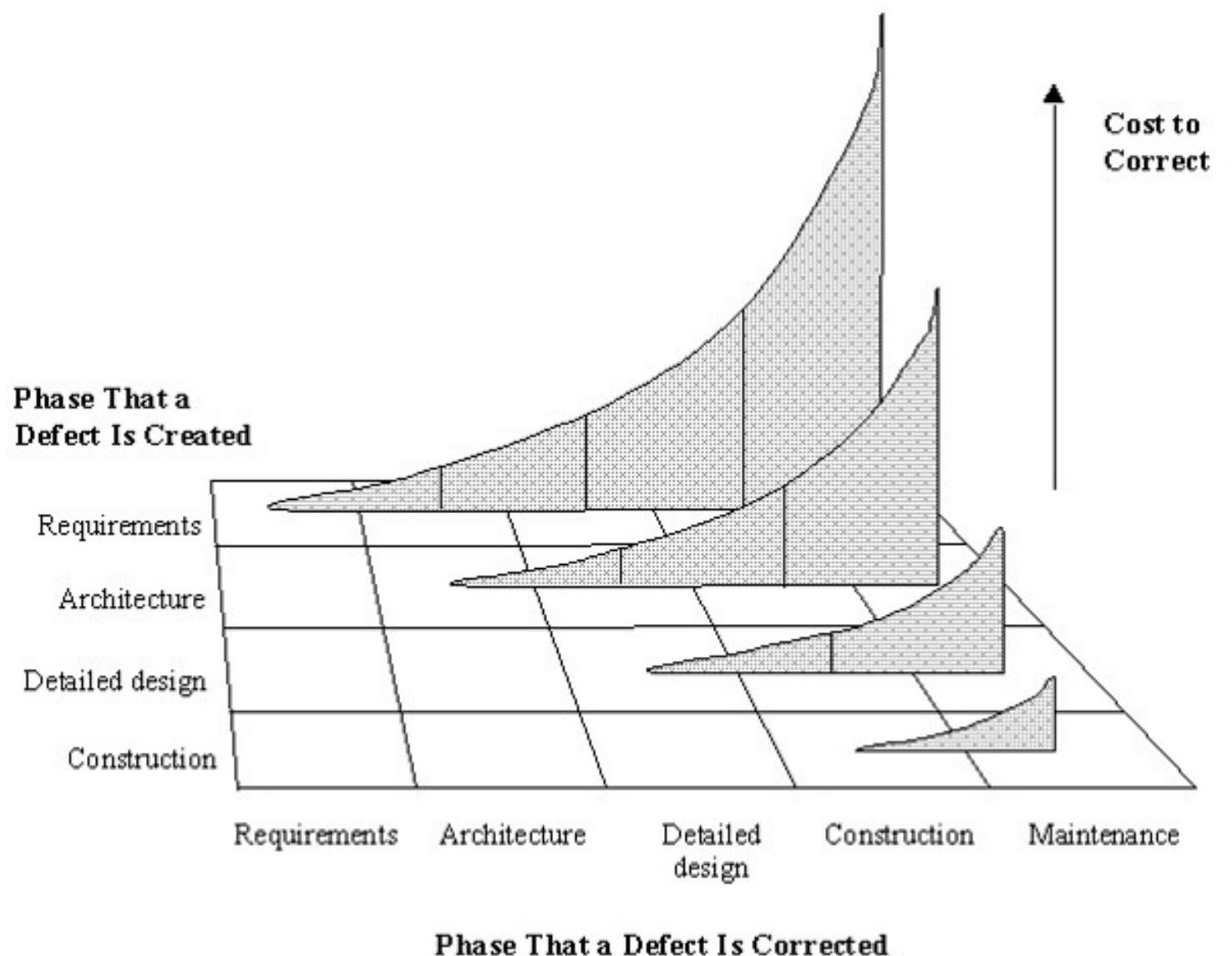
McConnell, Steve. Software project survival guide. Pearson Education, 1998.

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

Speaker notes

ideal setting of little process investment upfront



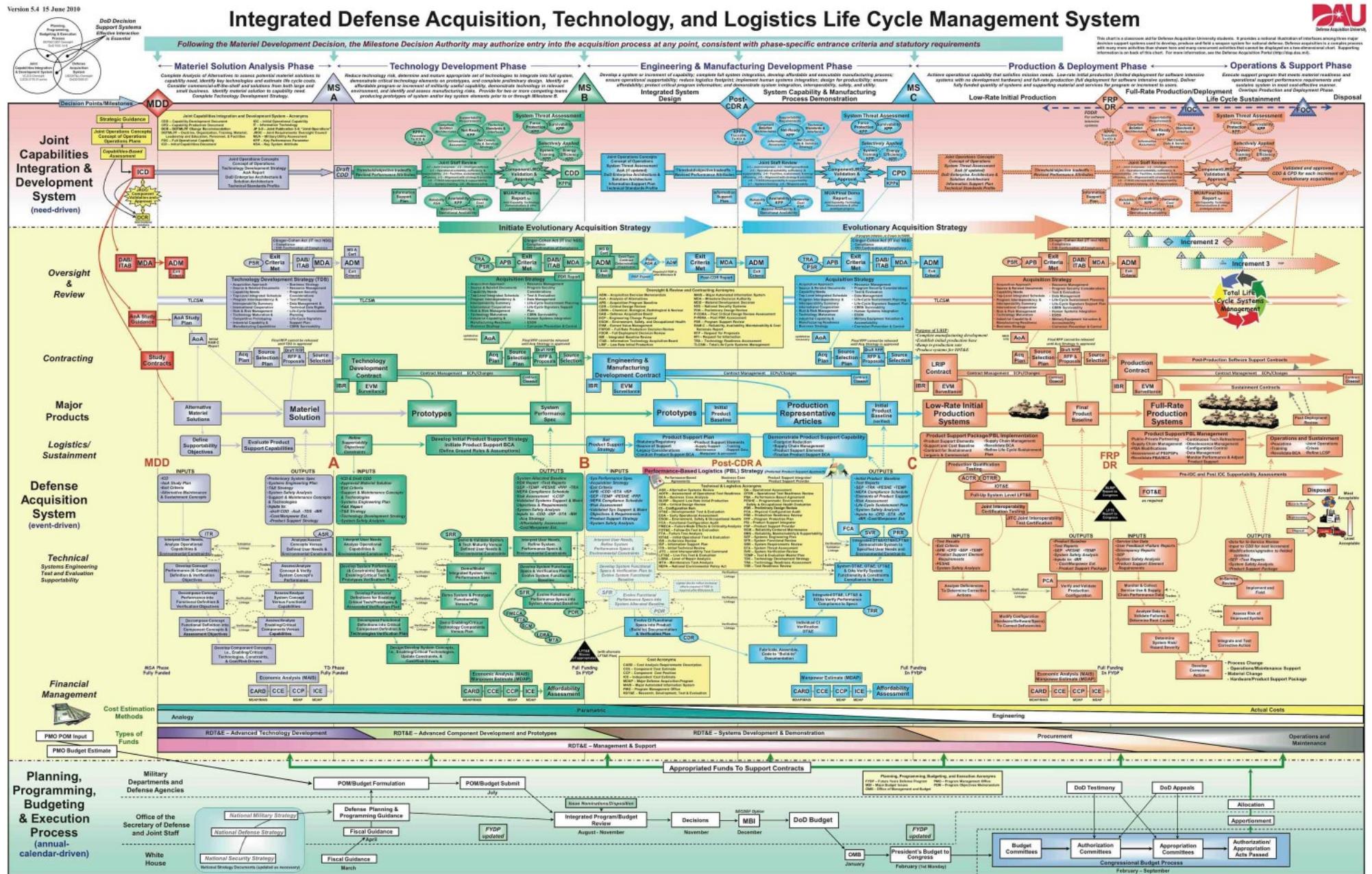


Copyright 1998 Steven C. McConnell. Reprinted with permission
from *Software Project Survival Guide* (Microsoft Press, 1998).

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.





Speaker notes

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

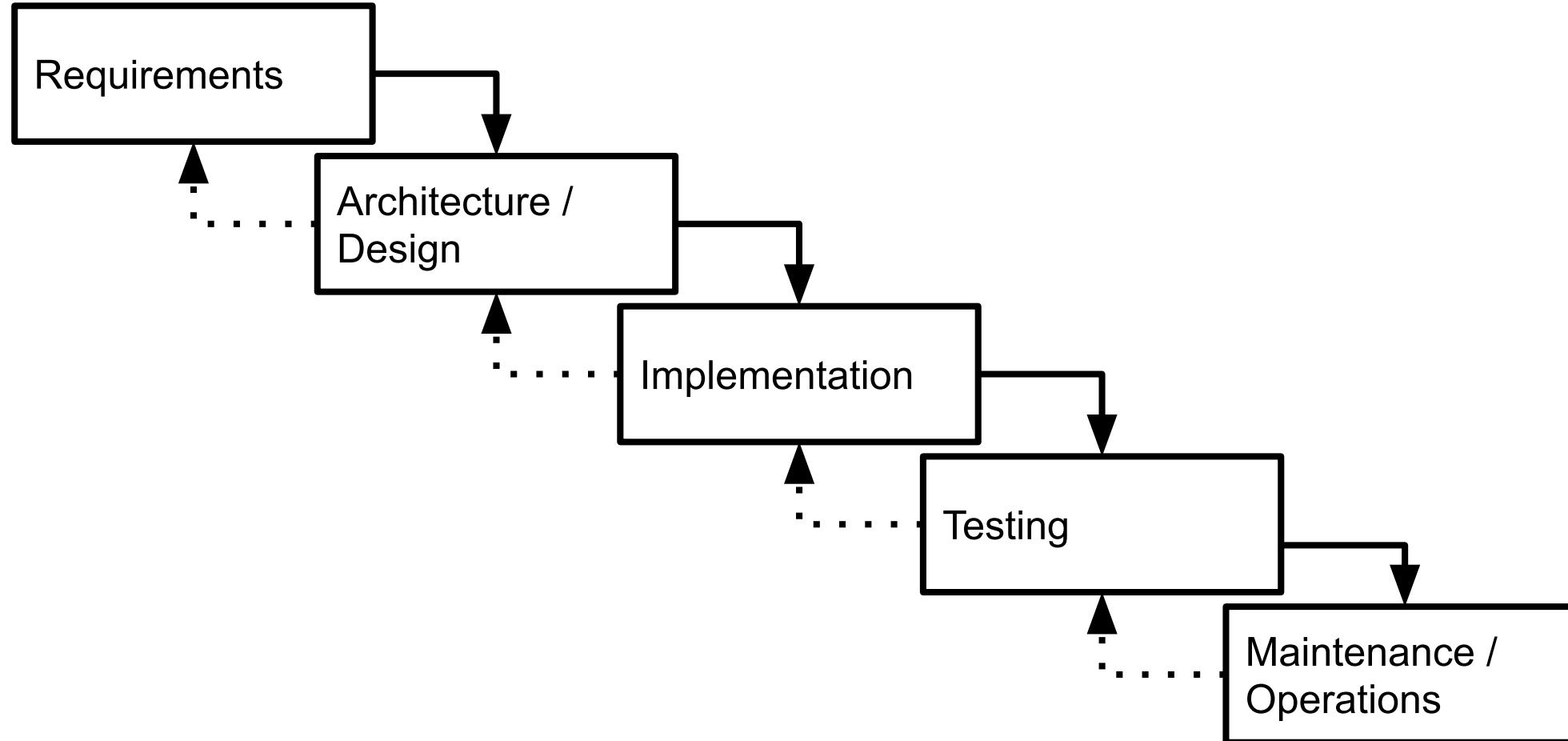


Software Process Models

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



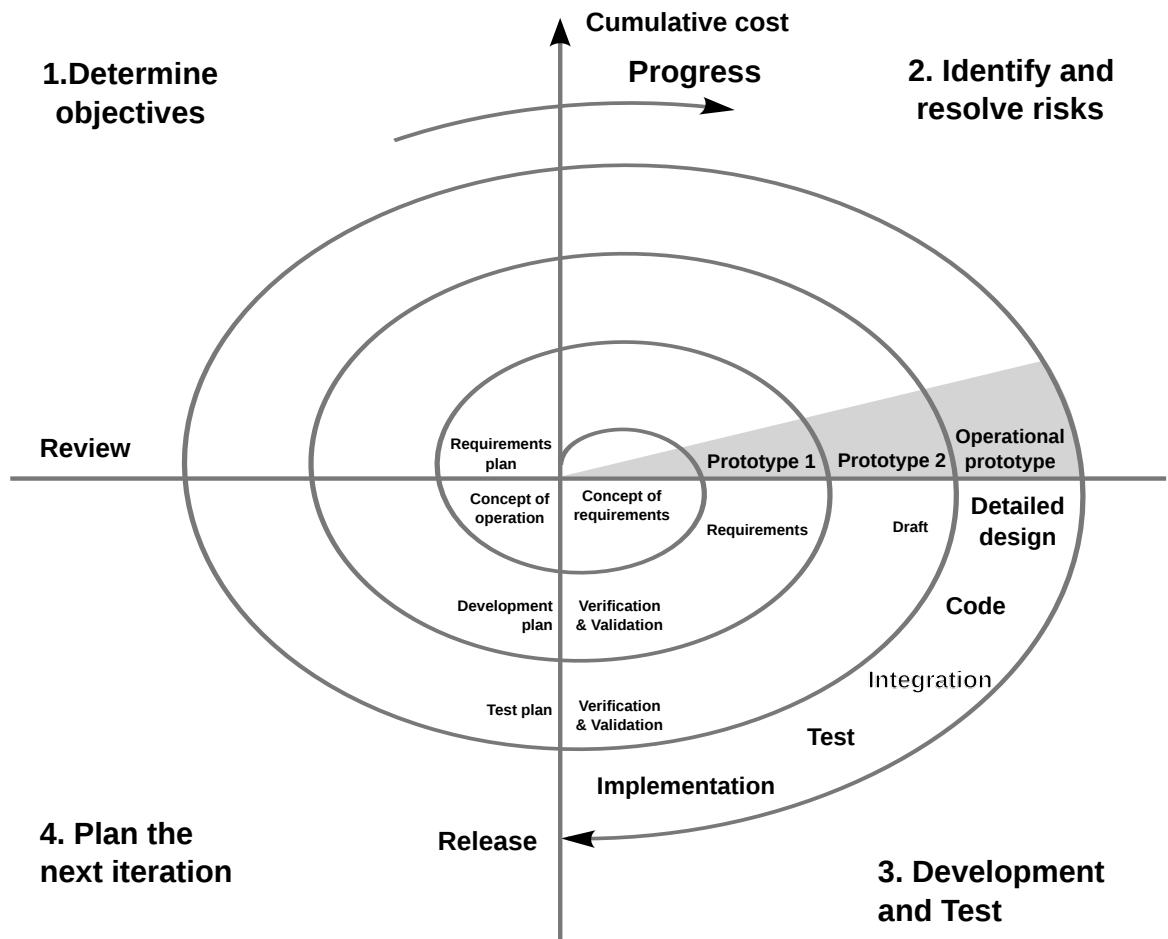
taming chaos, understand req., plan before coding, remember testing

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.

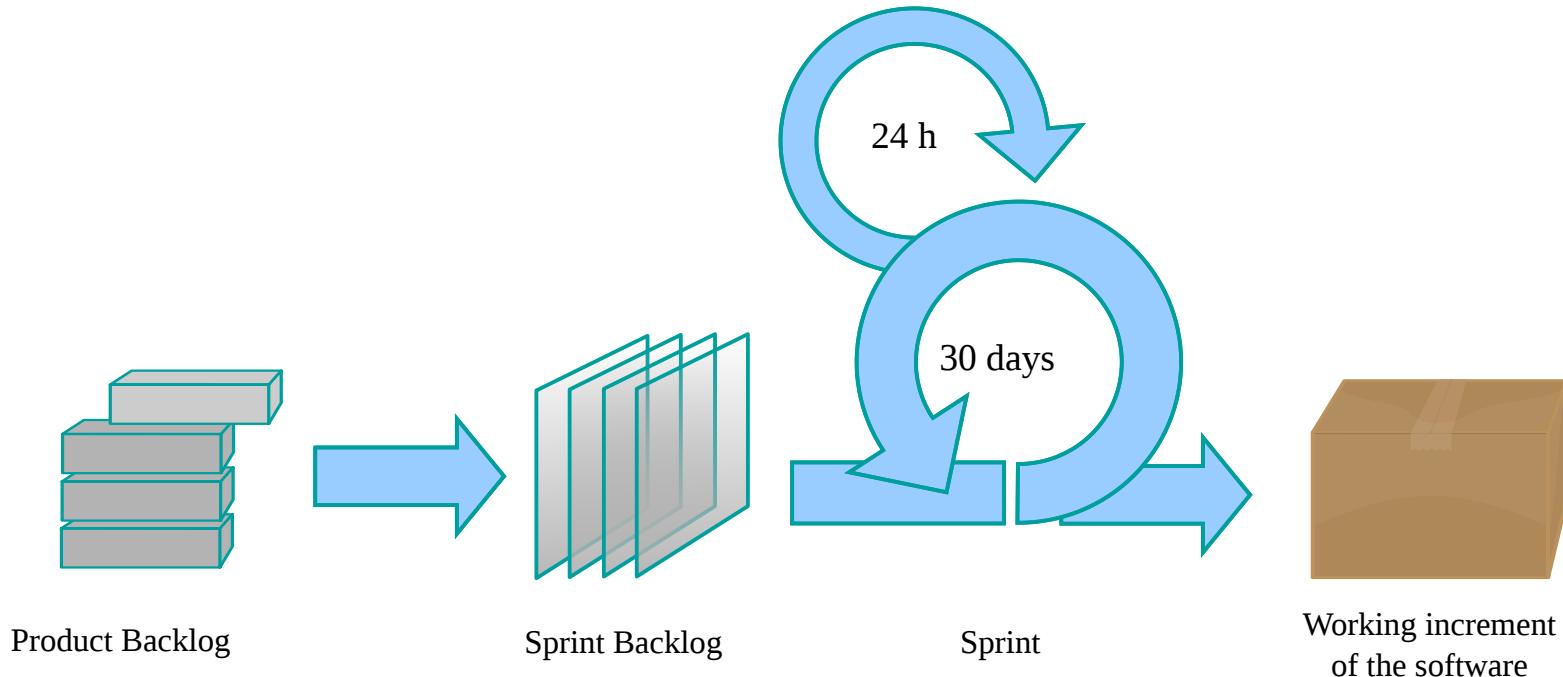


Risk First: Spiral Model



incremental prototypes, starting with most risky components

Constant iteration: Agile



working with customers, constant replanning



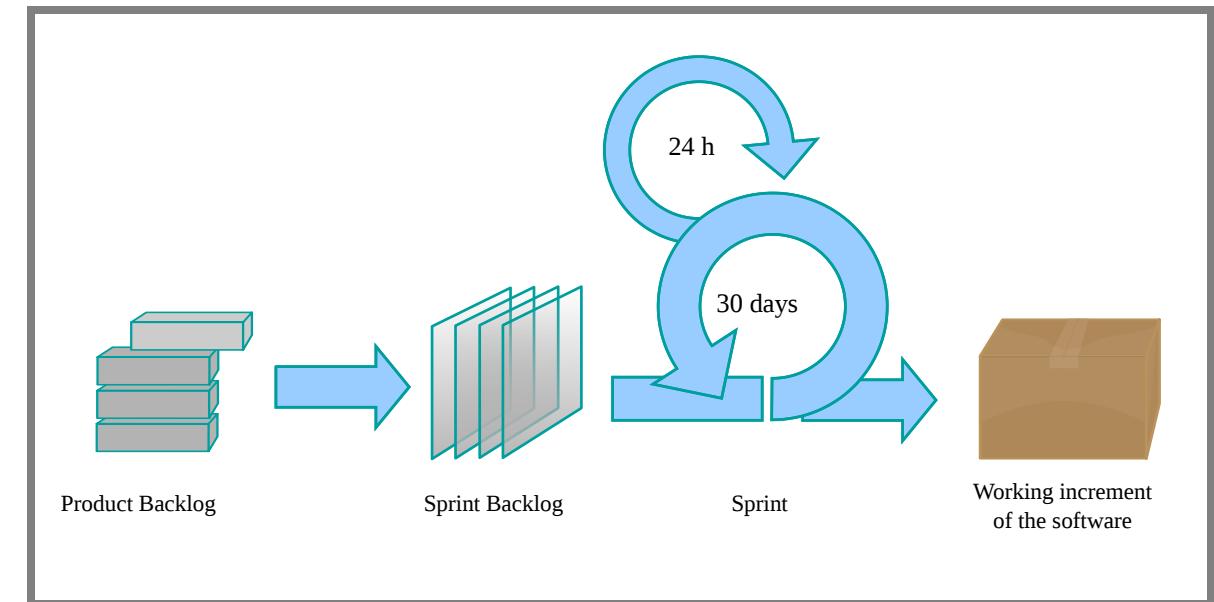
(Image CC BY-SA 4.0, Lakeworks)

Selecting Process Models

Individually, vote in slack: [1] Ad-hoc [2] Waterfall [3] Spiral [4] Agile
and write a short justification in #lecture

Data Science vs Software Engineering

Discussion: Iteration in Notebook vs Agile?



(CC BY-SA 4.0, Lakeworks)

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?

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

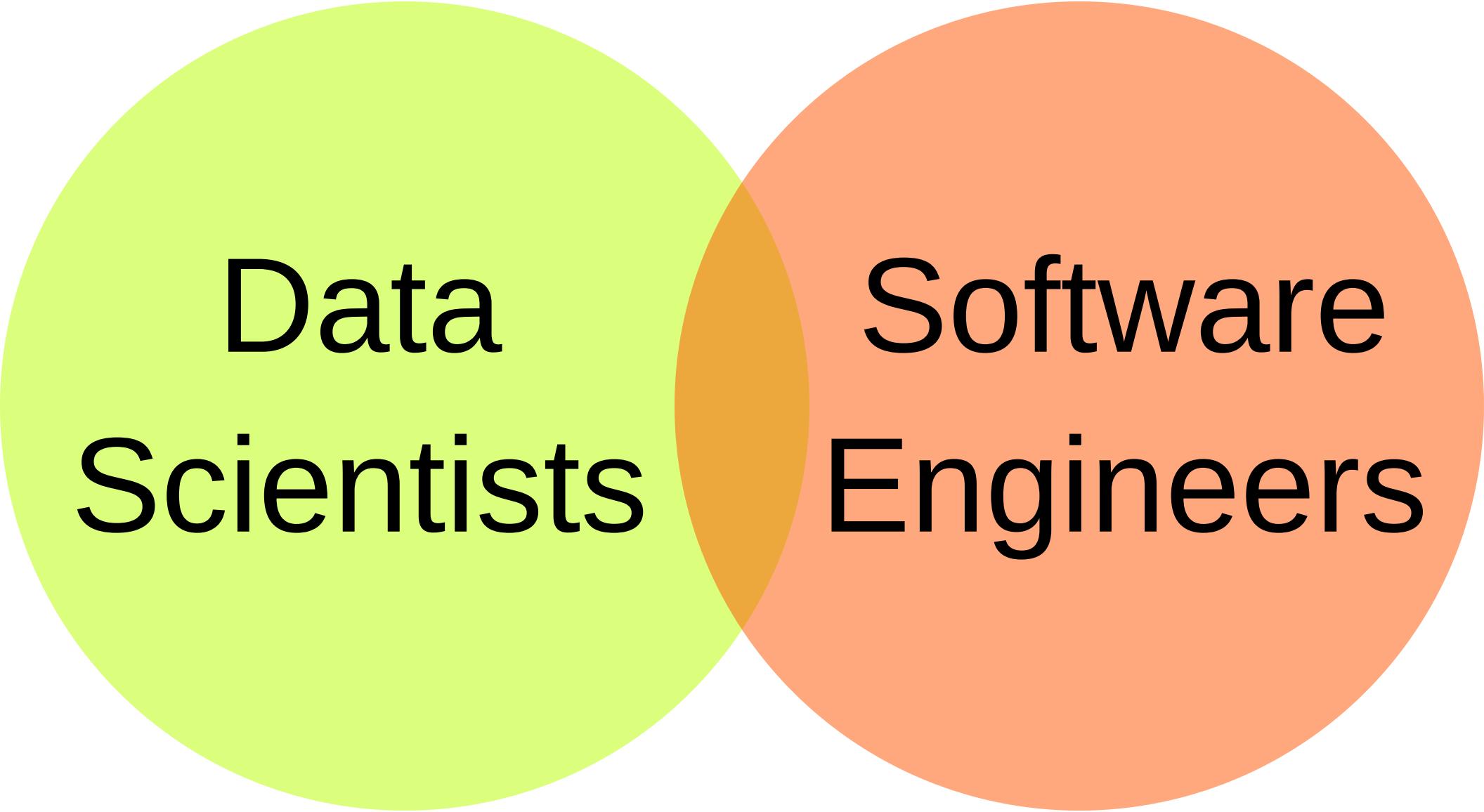
Understanding Data Scientist Workflows

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



A Venn diagram consisting of two overlapping circles. The left circle is light green and contains the text "Data Scientists". The right circle is light orange and contains the text "Software Engineers". The overlapping area between the two circles is shaded in a darker orange.

**Data
Scientists**

**Software
Engineers**

Data Science Practices by Software Eng.

- Many software engineers get involved in data science without explicit training
- Copying from public examples, little reading of documentation
- Lack of data visualization/exploration/understanding, no focus on data quality
- Strong preference for code editors, non-GUI tools
- Improve model by adding more data or changing models, rarely feature engineering or debugging
- Lack of awareness about overfitting/bias problems, single focus on accuracy, no monitoring
- More system thinking about the product and its needs

Yang, Qian, Jina Suh, Nan-Chen Chen, and Gonzalo Ramos. "[Grounding interactive machine learning tool design in how non-experts actually build models](#)." In *Proceedings of the 2018 Designing Interactive Systems Conference*, pp. 573-584. 2018.

Integrated Process for AI-Enabled Systems

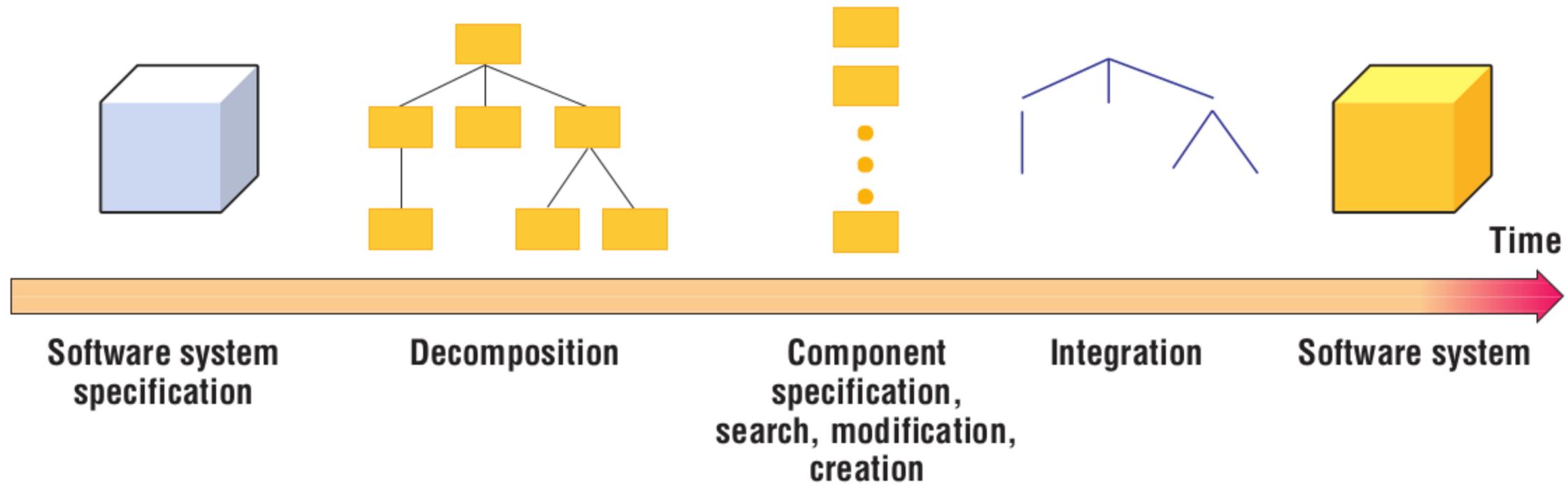
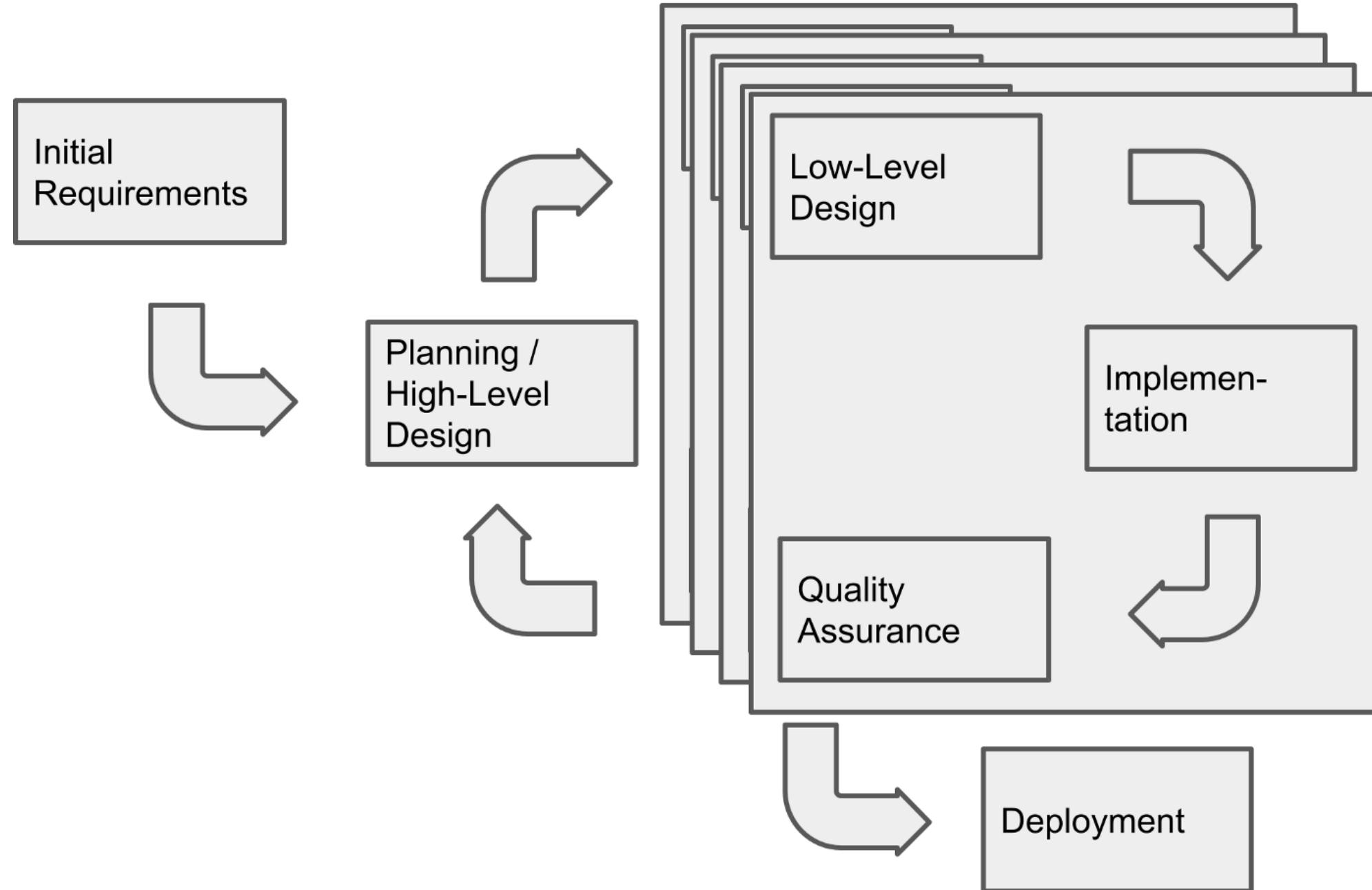
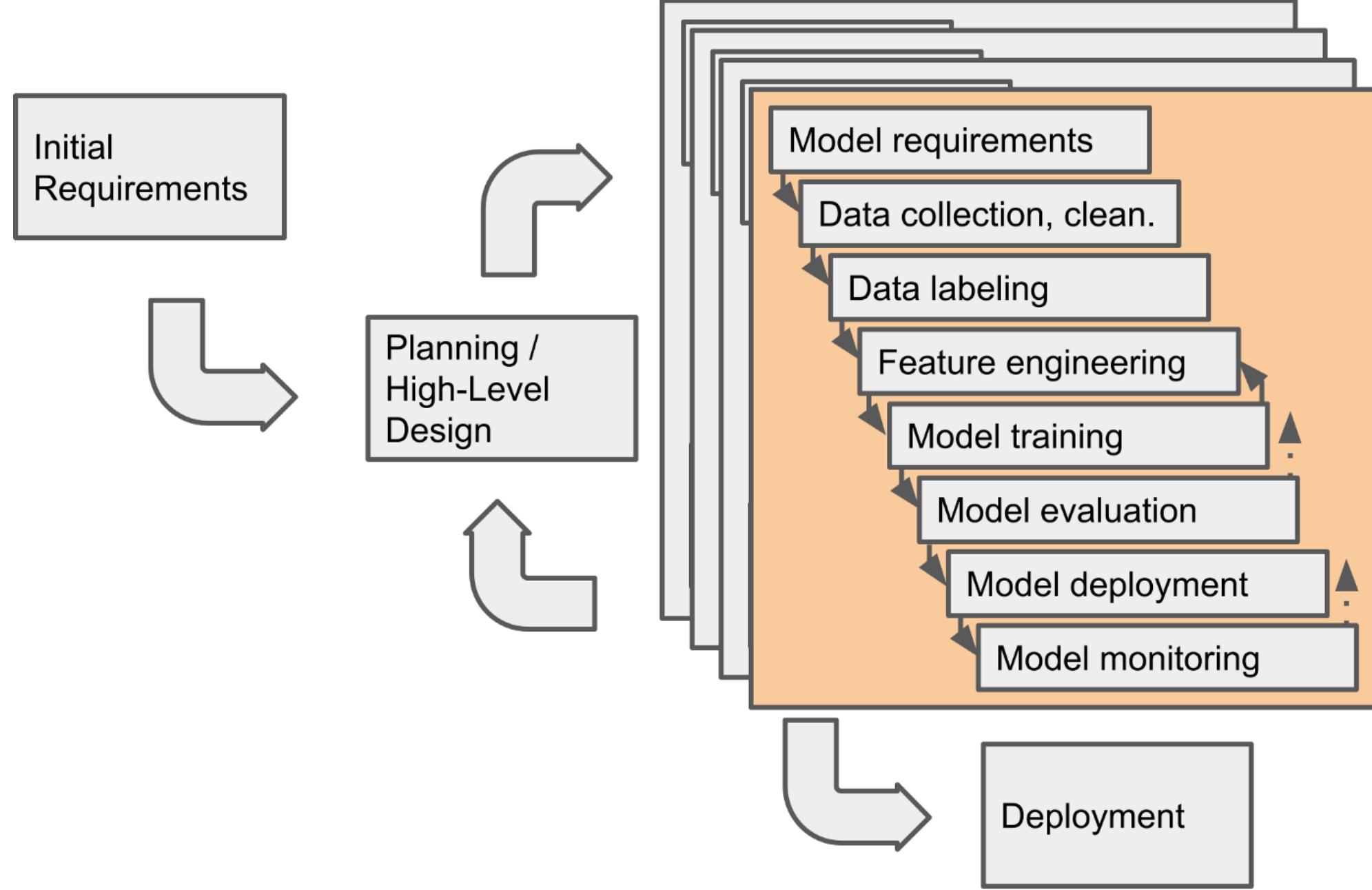
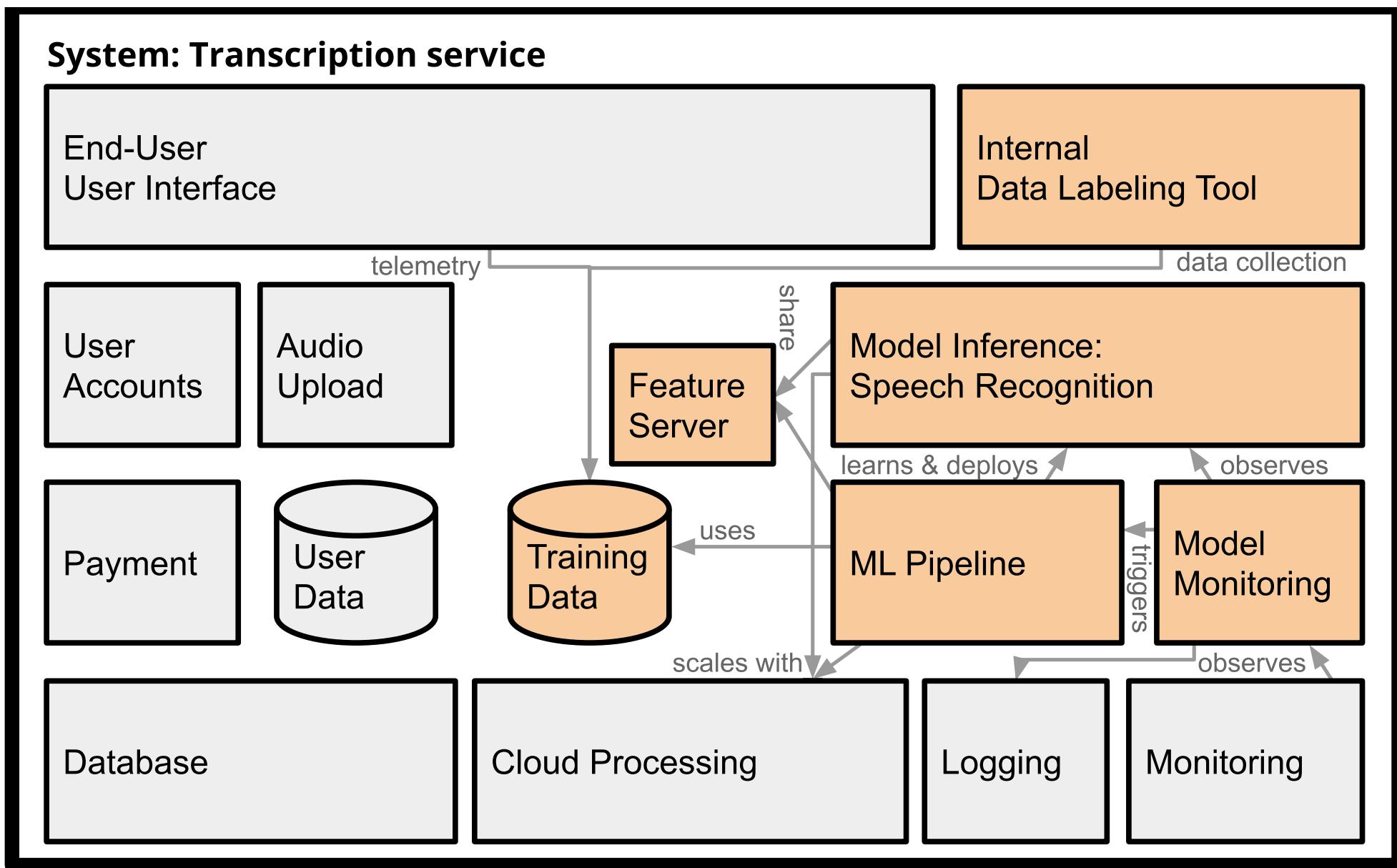


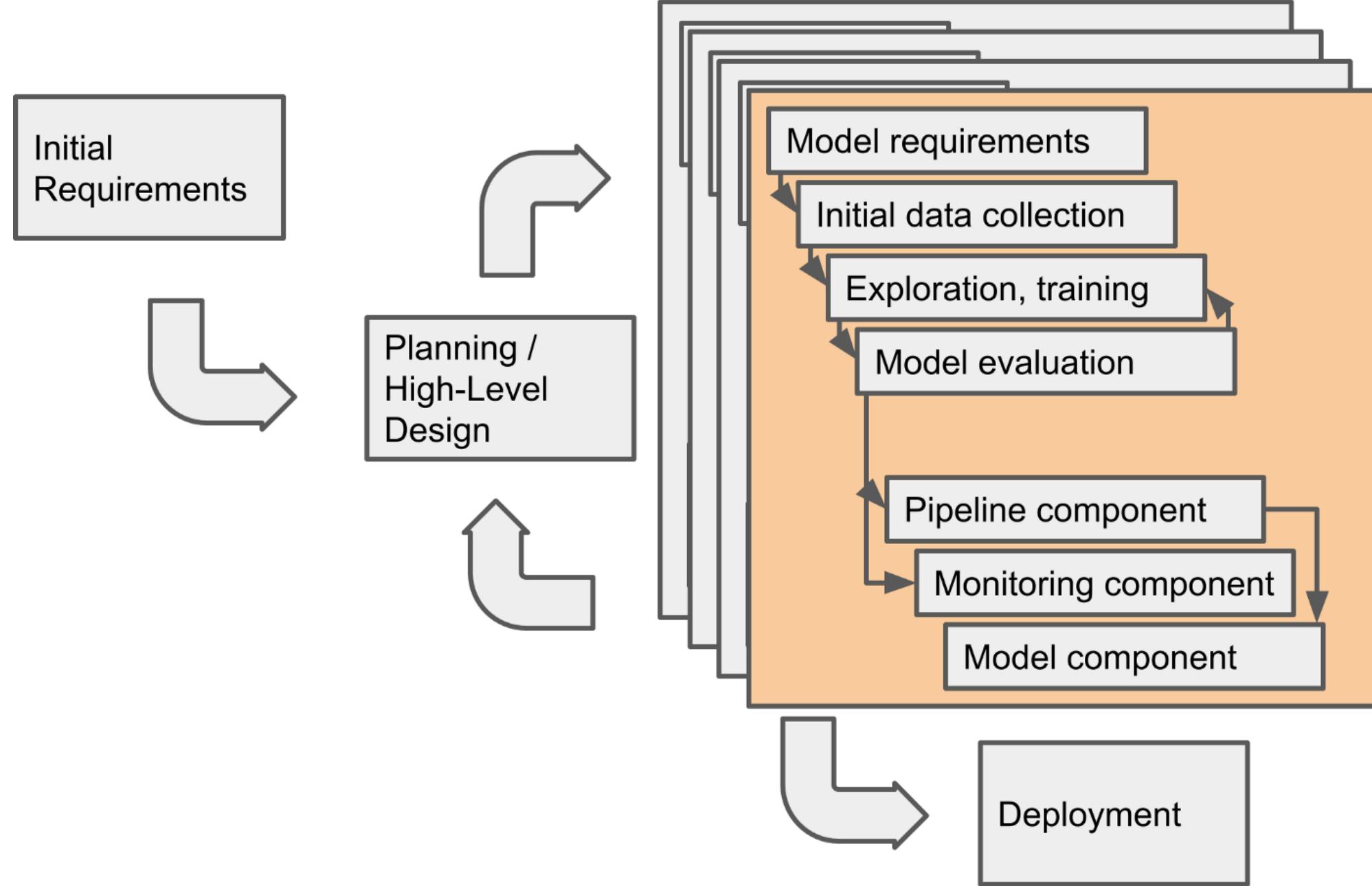
Figure from Dogru, Ali H., and Murat M. Tanik. “A process model for component-oriented software engineering.” IEEE Software 20, no. 2 (2003): 34–41.

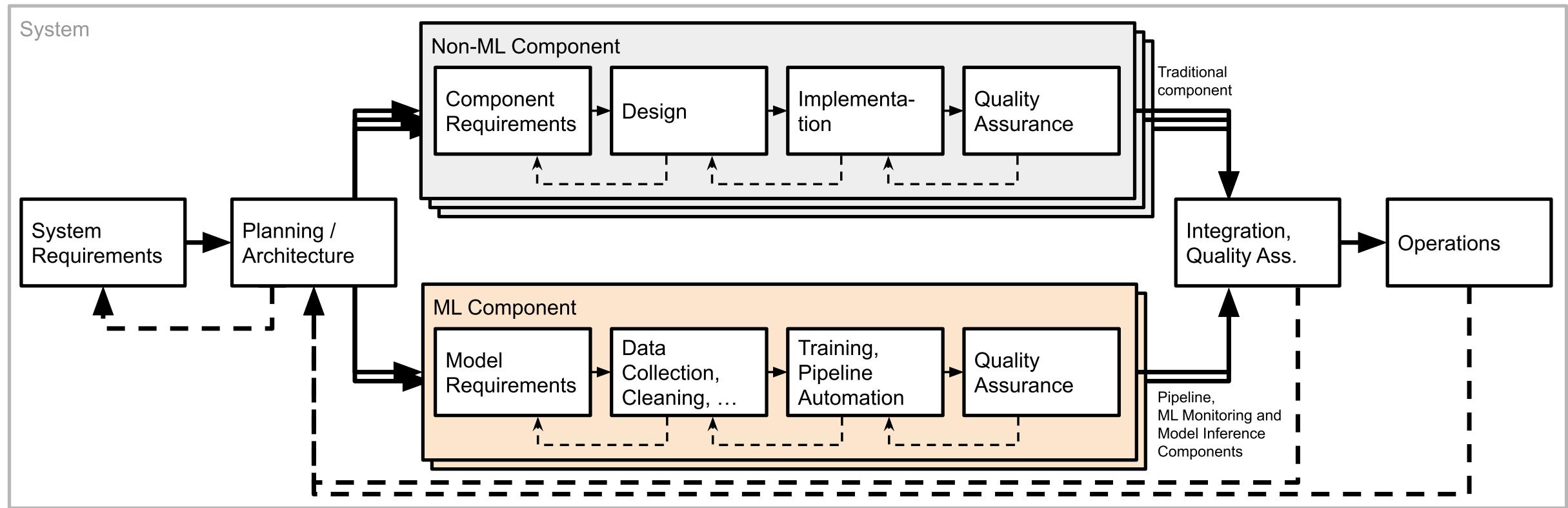






Legend: Non-ML component, ML component, system boundary





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

Trajectories

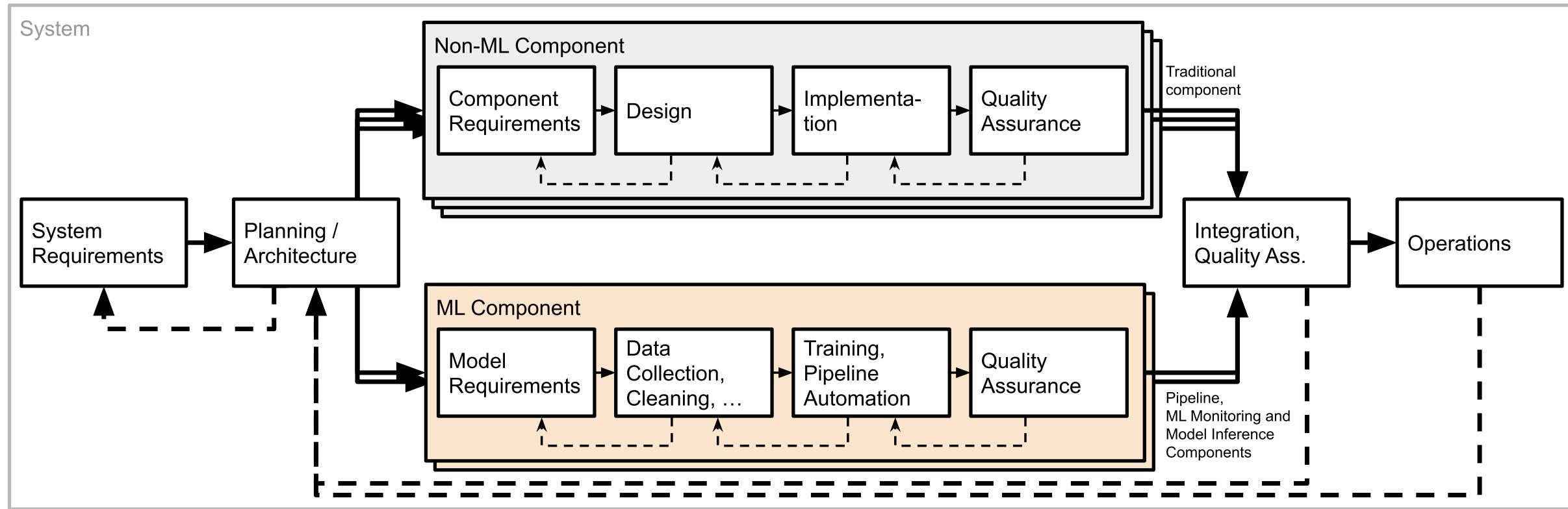
Not every project follows the same development process, e.g.

- Small ML addition: Product first, add ML feature later
- Research only: Explore feasibility before thinking about a product
- Data science first: Model as central component of potential product, build system around it

Different focus on system requirements, qualities, and upfront planning

Manage interdisciplinary teams and different expectations

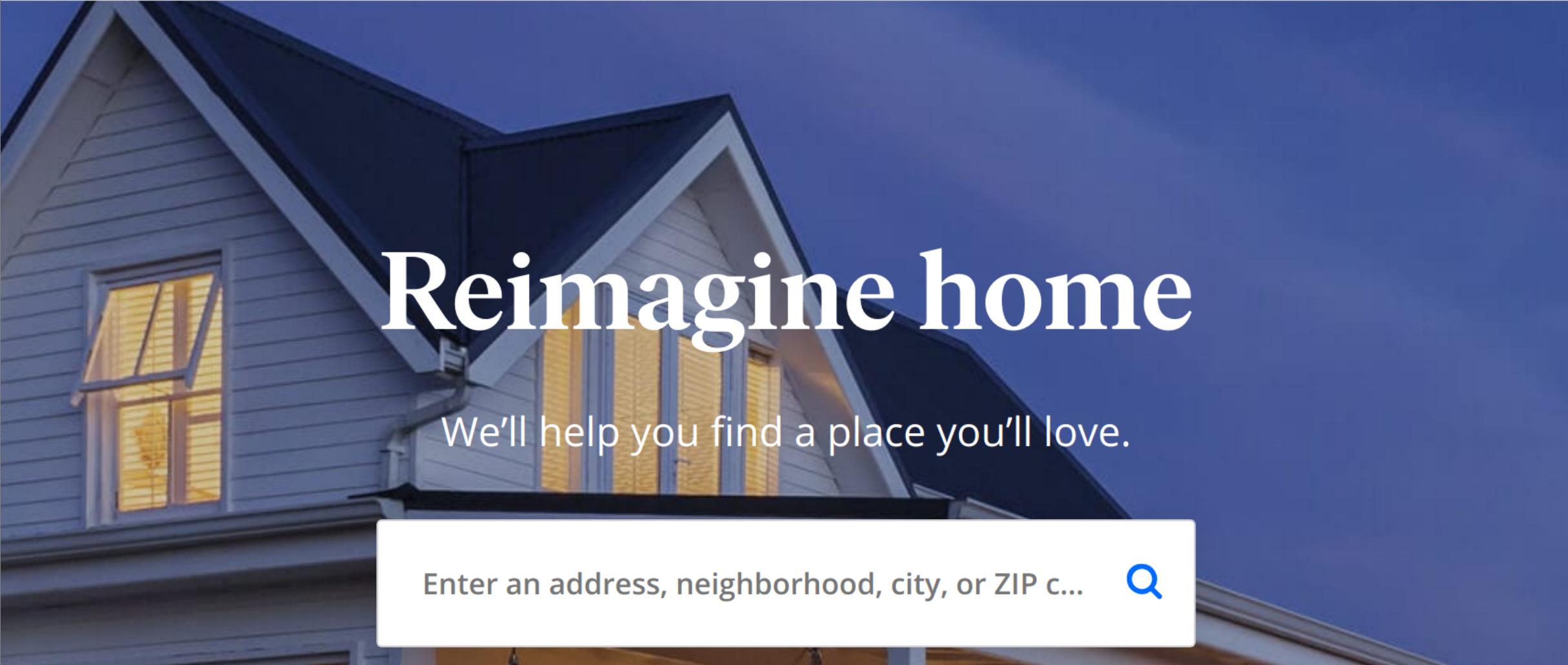
Model first vs Product first



Buy Rent Sell Home Loans Agent finder



Manage Rentals Advertise Help Sign in



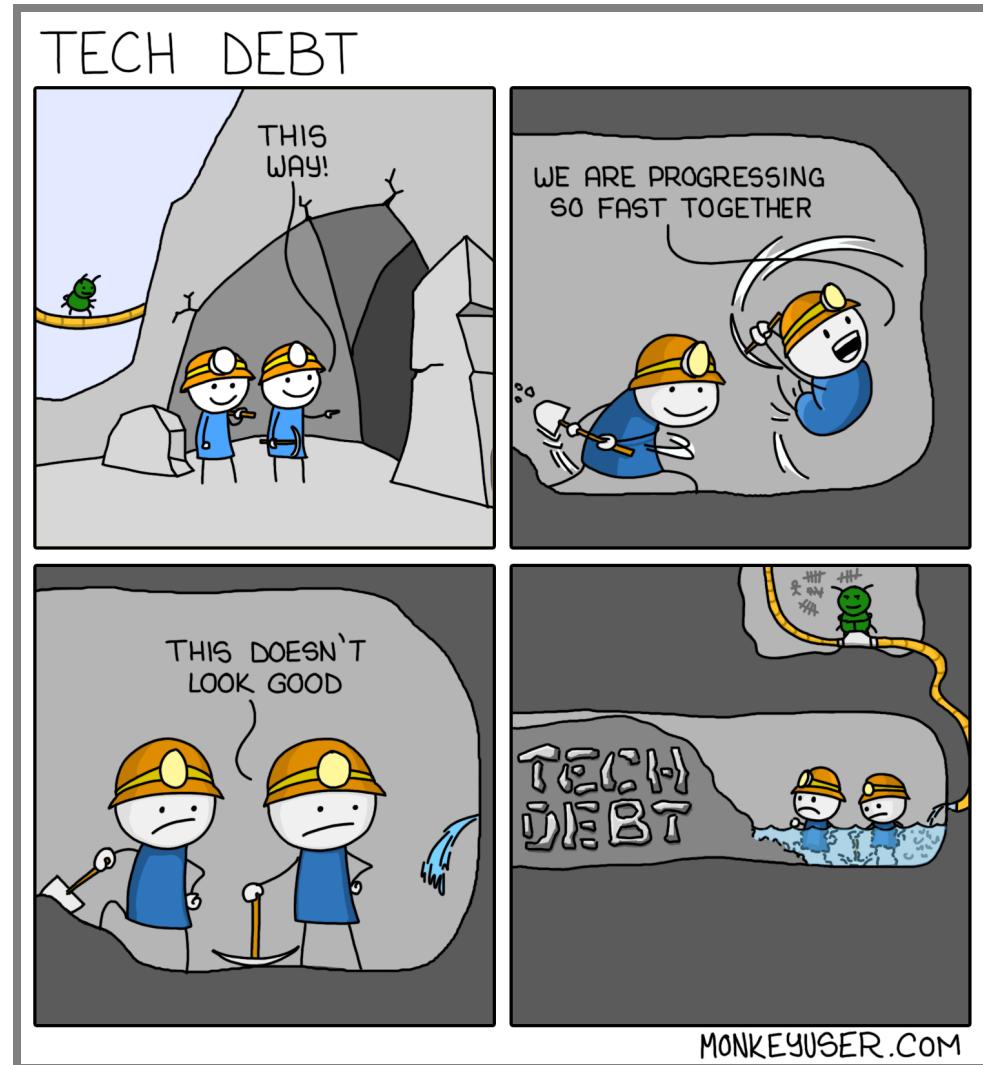
Reimagine home

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?

Reckless

*“We don’t have time
for design”*

Prudent

*“We must ship now
and deal with
consequences”*

Deliberate

Inadvertent

“What’s Layering?”

*“Now we know how we
should have done it”*

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

Breakout: Technical Debt from ML

As a group in #lecture, tagging members: Post two plausible examples technical debt in housing price prediction system:

1. Deliberate, prudent:
2. Reckless, inadvertent:

Technical Debt through Notebooks?

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

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: ML and Technical Debt

- 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
- ...

Buy Rent Sell Home Loans Agent finder



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...



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) technical debt with machine learning

Further 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.
- Studer, Stefan, Thanh Binh Bui, Christian Drescher, Alexander Hanuschkin, Ludwig Winkler, Steven Peters, and Klaus-Robert Mueller. "[Towards CRISP-ML \(Q\): A Machine Learning Process Model with Quality Assurance Methodology](#)." arXiv preprint arXiv:2003.05155 (2020).
- Martínez-Plumed, Fernando, et al. "[CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories](#)." IEEE Transactions on Knowledge and Data Engineering (2019).
-  Kaestner, Christian. [On the process for building software with ML components](#). Blog Post, 2020

Further Reading 2

- 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.
- Yang, Qian, Jina Suh, Nan-Chen Chen, and Gonzalo Ramos. "[Grounding interactive machine learning tool design in how non-experts actually build models.](#)" In *Proceedings of the 2018 Designing Interactive Systems Conference*, pp. 573-584. 2018.
-  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.