Principles & Practices of Software Development

Daniel Spoonhower
Carnegie Mellon University
Pittsburgh, PA

Daniel Huttenlocher
Cornell University
Ithaca, NY

Intelligent Markets, Inc.
San Francisco, CA
New York, NY

In This Talk

- Purpose
  - Relate some of our experience
  - Introduce way of talking about software development
    - Language for dialogue
- Audience
- ***Not*** in this talk
  - Breadth of experiences
  - Scientific study

Outline

- Background
- Experience: Requirements & Estimation
- Terminology: Principles, Problems and Practices
  - Some examples & comparisons
  - Things to look out for (e.g. competing Principles)
- Relating Principles to experience

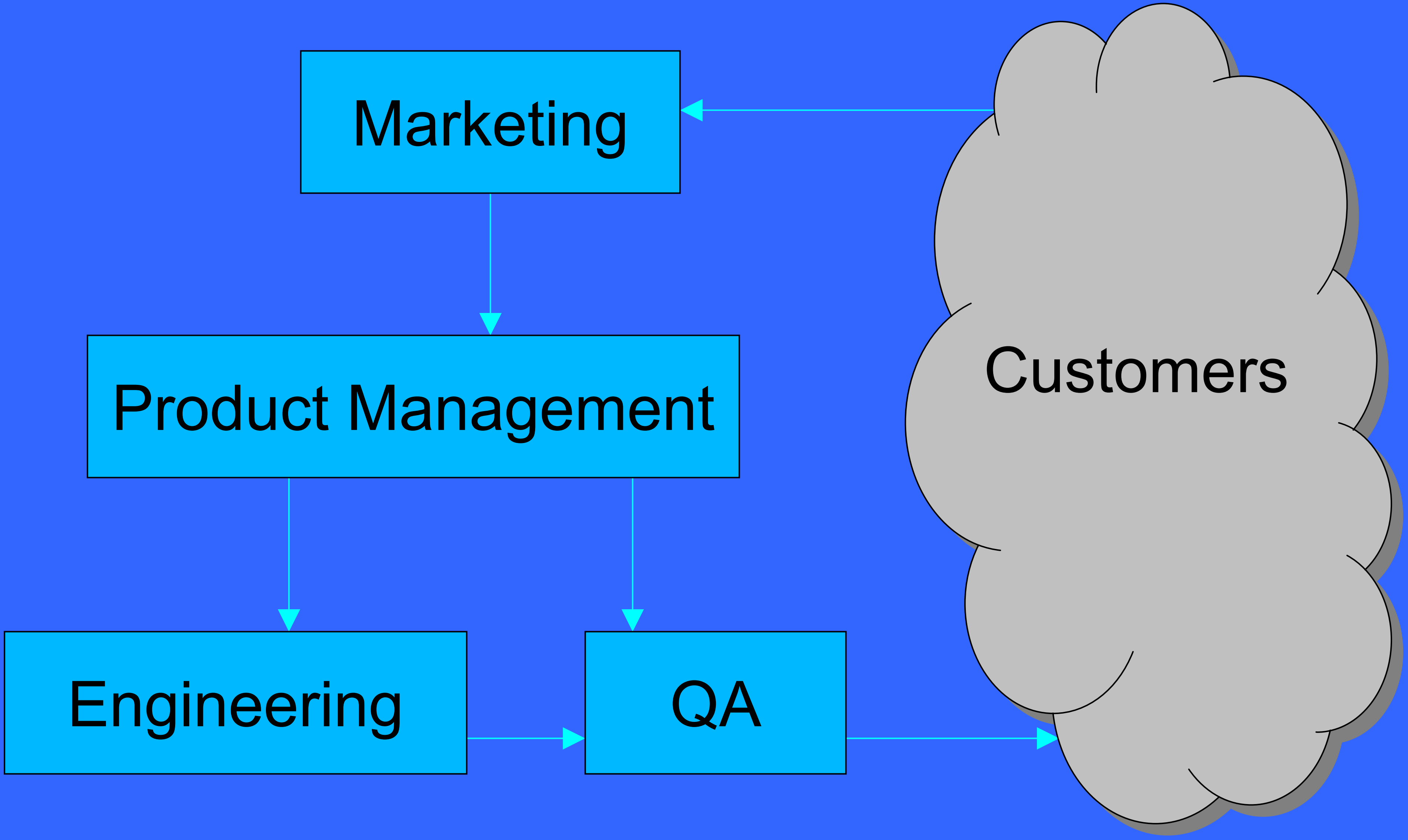
Background

- Our task – system for trading convertible bonds
- Our (prior) experience
- Our team – best people we'd ever worked with
- Our challenge:
  - High reliability
  - Complex (user) requirements
  - Technically challenging
  - Demanding schedule

Background

- Your experiences?
  - Written software (or just programs)?
- Your teams?
  - Size, project duration?
- Your challenges?

Simplified Organization



Requirements & Estimation

Requirements & Estimation

- First implementation
  - Started with partial outsourced version
  - Screen shots used as requirements
- No product management
  - Responsibilities shared by marketing & engineering
- Internal customer
  - Frequent delivery, rapid feedback

First Solution(!) Results

- Sparse documentation
  - Both requirements and implementation
  - Verbally conveyed = many changes
  - Written by developers
- Success!
  - Very flexible, agile process
  - System launched in 8 months
  - Many lessons learned

Product Management

- Second solution
  - Now enterprise software not service
  - External customers
  - Demanded clearer definition of product
- Feature by feature description
  - Hierarchical, outline format
- Specification change process
  - Manage document updates
  - Understand effects of changes

Second Solution

- Problems:
  - Lacked coherence
  - Serving many different parts of the company
    - Marketing, product design, engineering
  - Didn’t convey understanding
  - Delivered on-time but with poor set of features

More Documentation

- Third solution (attempted, not fully implemented):
  - Several levels of documentation, one for each use, e.g.
    - MRD (Marketing)
    - HLD & DLD (Product Management and QA)
    - TD (Engineering and QA)
- Conventional big company approach

Third(!) Solution Results

- Problems:
  - Lots of effort, difficult to manage
    - Many dependencies, gated tasks
  - Skew between different documents
  - Focus on **documents** more than on **development**

Interleaved Stages

- Our final solution: incremental!
  - Alternate requirements with estimates
  - Start with quick, rough ideas; work towards details
  - Drive to ship date – cut features to do so

Interleaved Stages

- Requirements
  - Business need, short descriptions, detailed functional and UI specs
  - Reprioritize as estimates established
- Several levels of specs and estimates
  - Day-week-month, “factor of 2 guess”, then +/- 25% with *small* tasks
  - More specific estimates derived from more detailed specs

Ongoing Challenges

- “Delta” specifications – note changes to product
  - Need both complete and difference spec
- Product team gaining understanding of implementation
  - Can find more workable solutions
  - More difficult to think independently
- Meet the needs of testing
  - Function point combinations
  - Workflow sequences

Lessons Learned

- Communication is important
  - Business needs → engineering
  - Estimates & implementation → PM
- Conflicting forces:
  - Include the best features
  - Ensure maintainability
  - **Ship on time**
- Make sure the **process** focuses resources on getting **product** done

Terminology

Terminology

Principle

A comprehensive and fundamental law, doctrine, or assumption. Principles may be universal, or they may apply only to certain types of projects.

Terminology

Principle

A comprehensive and fundamental law, doctrine, or assumption. Principles may be universal, or they may apply only to certain types of projects.

- Predictive
- Broadly applicable
- Relates to experience
- Expands understanding

Terminology

Problem

Something that can get in the way of rapidly developing high quality software that meets customer needs, while having fun doing it.

Terminology

Problem

Something that can get in the way of rapidly developing high quality software that meets customer needs, while having fun doing it.

- Observable
- Describes a state of being
- To be identified, minimized, avoided, solved

Terminology

Practice

A way of acting or working so as to avoid or to alleviate problems in developing software.

Terminology

Practice

A way of acting or working so as to avoid or to alleviate problems in developing software.

- Most importantly: an **action**
- Still abstract (as opposed to implementation)
- Focus of many methodologies

Our Goals

Principles

Problems

Practices

- Explicitly enumerate
- Study interactions
- Compare results

BUT…

…keep them separate!

Principles, Problems and Practices

Principles, Problems and Practices

- Seen some Problems and Practices related to requirements and estimation
- Consider some underlying Principles
  - Sometimes competing
- Relate to the experiences in specification and estimation

Domain Expertise

Principle: Understanding the domain is critical to understanding, explaining, and interpreting user requirements.

Domain Expertise

Principle: Understanding the domain is critical to understanding, explaining, and interpreting user requirements.

- Key to many of our initial practices
- Important for communication
  - Understand language; interpret spec
  - Critical for QA
  - Understand user needs (and feedback)

Changing Requirements

Principle: Requirements change, both because the understanding of the needs of users change and because the needs themselves change.

Changing Requirements

Principle: Requirements change, both because the understanding of the needs of users change and because the needs themselves change.

- When is a specification finished? Never!
  - But need to ship the product
- All requirements change
  - More changes for new products

Specification Cost

Principle: There is a high cost to writing and maintaining detailed specification documents that are accurate and effectively convey understanding.

Specification Cost

Principle: There is a high cost to writing and maintaining detailed specification documents that are accurate and effectively convey understanding.

- What form of specification is most useful? E.g.
  - None, note cards, templates
  - Functional, UI, relationship, sequence

Little or No Specification?

- Advocated by eXtreme Programming
- Successful practice for our first implementation
- For large and/or new projects, cost of maintenance can be astronomical
  - E.g. “big company” approach

Why do we need a specification?

Unreliable Memory

Principle: Personal memory is a poor substitute for a written document.

Unreliable Memory

Principle: Personal memory is a poor substitute for a written document.

- Verbal communication can easily be misconstrued
- Memories fade over time
- People make expensive storage devices

Adequate Specificity

Principle: When customer needs admit many interpretations, precise descriptions help ensure usability and quality.

Adequate Specificity

Principle: When customer needs admit many interpretations, precise descriptions help ensure usability and quality.

- Applies differently to different projects
  - Depends on complexity of user needs
  - E.g. compression software

Detailed Specification?

- Specification is important for establishing obligations
  - What will be implemented
  - What will be tested
  - What will be delivered
- Specification evolves with product
  - As opposed to Waterfall, where spec drives remainder of process

Competing Principles

Changing Requirements

Unreliable Memory

Specification Cost

Adequate Specification

How do we achieve a balance?

Clear Statement

Principle: A clear and concise statement of user needs generally results in the development of better software.

Clear Statement

Principle: A clear and concise statement of user needs generally results in the development of better software.

- Accessible to company & customers
- Guidelines:
  - Use informal communication to establish understanding
  - Use documentation to preserve it

Incremental & Iterative

- Incremental specification avoids risks inherent in this set of Principles
- Other iterative practices can be found in:
  - Technical design
  - Scheduling releases
- Negotiation between competing forces
  - Taking "small steps" reduces the chance that one force will "defeat" the others

Driving Force: Ship Date

- For new companies
  - Establish reputation and credibility
- Balancing force
  - Counteracts “feature creep”
  - Millions of ways ***not*** to ship!
- Healthy part of project lifecycle
  - Allow for personnel & process transitions
- It is an **external** and **concrete** goal!

Real Use

Principle: Understanding the needs of users and validating that those needs have been met is best done with a real implementation of the software and real users.

Real Use

Principle: Understanding the needs of users and validating that those needs have been met is best done with a real implementation of the software and real users.

- "Line in the sand"
- Ultimate validation
  - Not just for the spec, but for the product

… & Incremental Processes

- Real Use – push the product all the way through the process
  - At regular intervals
  - To validate feature set (incremental specification)
  - To check implementation (incremental delivery)
  - To get feedback on the process itself (projects change – no single process is “correct”)

Summary

- Use Principles to understand working constraints
  - Abstract away from Problems/Practices
- Be aware of competing Principles
- Use incremental and iterative processes to alleviate risk caused by conflicts
  - Take small steps and re-evaluate