Non-Technical Issues in Software Development

David E. Kieras

University of Michigan

Introduction

Question: Why isn't software of higher quality?

- More useful, more usable, more reliable?
- Many large software projects fail (~50+%).
  - Over budget, behind schedule, abandoned, unusable, or simply don't work when delivered.

Answer: There are problems in how sofware is developed.

- Organizational and social rather than technical.
- Frequently a disconnect between the developers and the users.
  - Developers don't build what users actually want.
- How software is typically developed makes the situation worse, not better.

Players:

- Users - people who actually interact with the software.
- Customers - people who choose to buy the software.
- Developers - people who design and write the software.
- Managers, administrators - people who make decisions about what will be developed or purchased.

Describe in terms of two aspects of software development:

Processes - how to organize the development process.

Contexts - the kind of organization developing the software.

Software Development Processes
- how development is organized

Topic in Software Engineering:

- Standard stage or waterfall model.
- Evolutionary model.
- Spiral model.
- Agile development and other newer ideas.

If a software development organization doesn't have some kind of process in place, it is either:

- Very small - developers keep it all in the heads;
- Likely to fail, probably sooner rather than later, due to the 50+% failure rate.

Software Development Contexts
- the kind of organization

Contract development organization

- Customer specifies the system, contractors design and build it in response to contract specifications.

Product development organization

- Products developed for discretionary purchase in a large market.

In-house development organization

- One group in an organization develops software for another group in the organization.

The processes originally developed in the contract organization.

- Present this organization first and summarize how processes developed, then on to the other forms of organization.

Contract Software Development

Earliest software development process was code-and-fix.

- Hack some stuff out and then try to make it work - nothing formalized.

Better process developed in response to government (especially military) procurement of very large systems.

- U.S. Government is the original large computer purchaser and user.
- Accountability required to counter tendency toward corruption.

Customer specifies the system, then contractors design and build it in response to contract specifications.

Users known initially, developers identified when contract awarded.

Driven from specification documents.

Emphasis on controlling development process.

- Problems are much more expensive to fix after the code has been written.
- So try to figure everything out beforehand.

Developer's success depends on adherence to specifications.

Contract Development Led to Defined Processes for Software Development

Need for accountability led to a well-defined development and procurement process.

Administrative Process:

- Customer prepares specifications
  - e.g. Naval Bureau of Personnel wants a new personnel records system.
  - Navy officials write specifications for the system.
- Government solicits bids for design, coding, other phases of project.
- Contractors bid for each phase.
  - Not necessarily the same contractor for different phases.
- Information available to contractors is normally restricted to paper documents.
  - May have in-house or consultant experts, but often no access to actual users - might even be illegal to talk with them.
- Big premium on the products of each phase being documented.
  - Enables "throwing it over the wall" to the next phase, which might be a different contractor.
  - Helps customer avoid getting "locked in" to a single contractor.

Overall Contract Process

User organization:

- Establish feasibility.
- Specify requirements - usually in terms of functional capabilities.
  - Usefulness and usability might be mentioned, but only in general terms.
  - E.g. "The system shall be easy to operate efficiently."
  - Contractors can't really adhere to such vague specifications, so tend to be ignored in favor of hardware/software specifications.

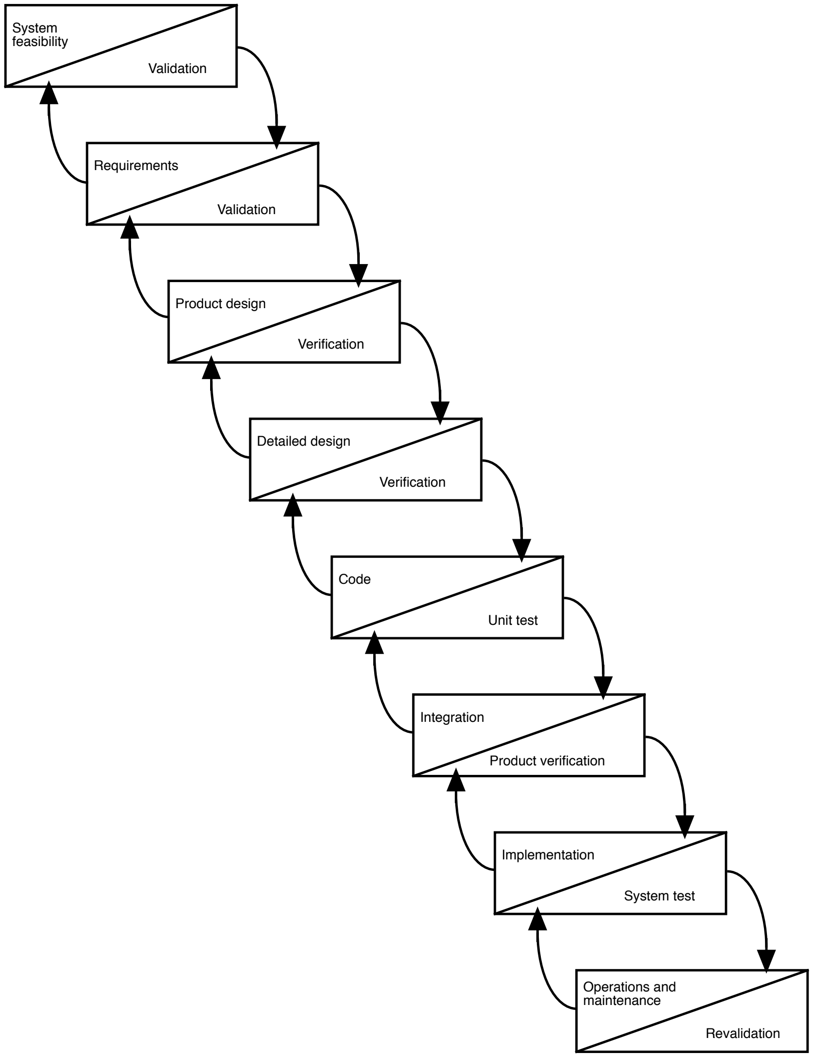
Contractor or Contractors:

- Design system.
- Do detailed design.
- Write code.
- Test system.
- Integrate & Install system.
- Operation.
- Maintenance.

Emphasis is on doing complete design before coding.

- Prevent costly errors from having to change design during coding.
- Stage & waterfall models of software development.
  - Stage model - 1956
  - Waterfall model - about 1970 - current industry standard.
- A major advance over earlier hacking-it-out, code-and-fix approach.

The Waterfall Model for Software Development



A series of stages, with limited feedback back to previous stage.

Why Contract Development Often Fails

Design must be "signed off" on before any coding is done, so no testing of design concepts is possible prior to completion!

Whole point is to avoid design iteration, so there is no test-modify loop for the choice of functionality and user interface testing.

If usability and functionality is not adequately captured in system specifications, then not likely to be designed into the system!

- Little opportunity for contact between developers and users.
- Anything specified about the interface or system, no matter how bad, has to be supplied
  - "Green suit" principle.

Quote from Boehm (one of the gods of software engineering):

- "[The waterfall model] does not work well for many classes of software, particularly interactive end-user applications. Document-driven standards have pushed many projects to write elaborate specifications of poorly understood user interfaces and decision-support functions, followed by the design and development of large quantities of unusable code."

Change coming only very slowly.

- It would help if the specifications included even a little user task information.
  - "Concepts of use" document - or use cases - still not normally supplied!
- Need different model of software development!

Concept of Boehms's "Spiral" Model

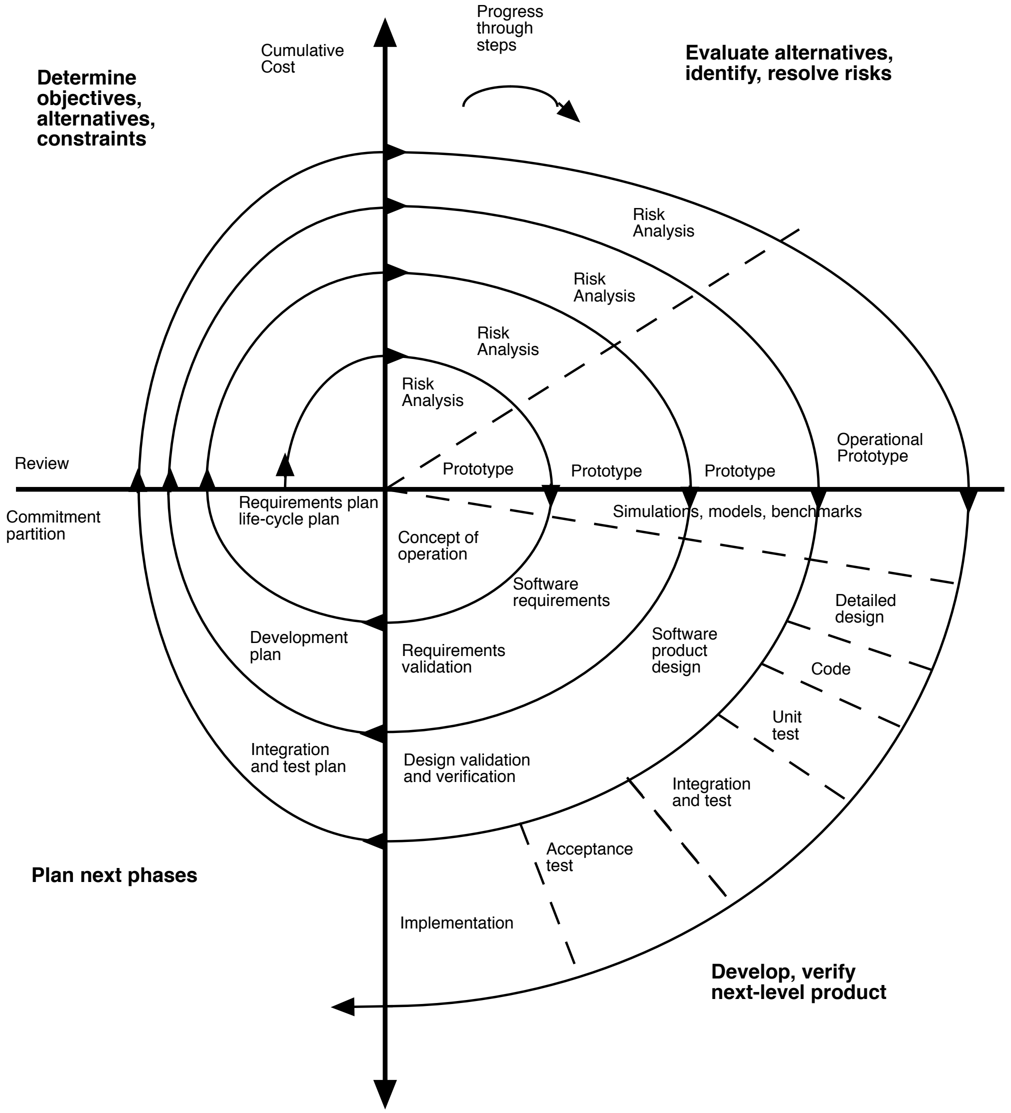
Driven by repeated identification and resolution of risks: A cycle of

- Determine objectives, alternatives, constraints.
- Evaluate alternatives, identify, resolve risks.
  - Analyze risks, construct prototypes, models, etc to resolve risks.
- Develop, verify products at each level, depending on cycle.
  - Includes waterfall model stages at the end.
- Plan next phase.

Risks and resolution approaches - descending priorities:

- Personnel shortfalls.
  - Staffing with top talent, pre-scheduling key people, etc.
- Unrealistic schedules & budgets.
  - Detailed cost & time estimation, design to cost, incremental development, etc.
- Developing the wrong software functions.
  - Organization analysis, mission analysis, operations-concept formulation, user surveys, prototyping, early users' manuals.
- Developing the wrong user interface.
  - Task analysis, prototyping,scenarios,user characterization.
- Gold plating.
- Continuing stream of requirement changes.
- Real-time performance shortfalls.
- Straining computer-science capabilities.

Boehms's "Spiral" Model



Risk identification/resolution at beginning, waterfall at end.

Agile Development Process

A relatively new and still-developing approach.

- Began to develop in the late 1990s, announced in the 2000's.
- A variety of techniques and ideas, but no single overall process description.

Basic concept: Requirements change frequently, so adapt to changes rapidly instead of wasting time planning everything in advance.

- Deprecate role of written design and planning documents.
  - Requirements change too rapidly - you could be revising code instead of the documents!
- Emphasize role of getting software written, tested, and revised rapidly.
  - Frequent, fast iterations each creating small pieces of functionality.
  - OOP techniques make it possible.
- Rely on small teams of expert programmers in direct contact with customers and each other - a craft workshop, not a factory.

Example techniques - good ideas!

- Pair programming - review and fix the code as it is written.
- Test-driven development - the tests are the specifications.
- Frequent refactoring to maintain high code quality.

Controversial, concepts and practices still in flux.

- A fad, or enduring approach?

Product Development Organization Context

Products developed for discretionary purchase in a large market.

- Organization identifies potential products, develops and markets them.
- E.g., almost all personal computer software vendors.

Developers known immediately, users identified at time of purchase.

Driven by perception of market:

- Historical shift in emphasis from functionality-only to quality user experience.

Success depends on whether there are enough users "out there" who buy the product.

Problems in the Product Development Organization Context

Product development organizations adopted the waterfall model, although situation was actually quite different from Contract Development.

- Management need to control, routinize, development process.
- No preexisting specifications.
- Actual users of product are not really known until they buy the product.
- Developers have to guess who the users will be, and must develop their own product specifications.

Have all of the disadvantages of contract development, with none of the advantages.

- Can't just play it safe by writing to customer's specification, and own specifications might be wrong or poor choices.
- Many failed products because there were not enough customers.
- Constant competitive pressures from other companies - no bid winners!

Slow feedback from the market.

- Often distorted because feedback is through customer (e.g. administrator) who actually makes purchase, but is not the user.

Situation is changing in product development organizations.

- Typically are continually revising and improving products, so have an iterative process already.
- Tend to be market driven, so quality can get considered at least somewhat.

In-house Development Context

One group in an organization develops software for another group in the same organization.

- Characteristic of many information-intensive organizations.
  - E.g. manufacturers, banking, University of Michigan.
- Both developers and users known from the beginning.
- Driven by needs of user group.
- Success depends on whether the user group accepts the software.

Advantages and Problems for Usability in In-House Development

- Since users & developers are under the same roof, user contact is easiest.
  - Users often participate in the design.
  - Also, developers often "paid" by the users, so users often have to be consulted and satisfied.
  - Schedules are usually more flexible.
- But conflicts can arise over roles and jurisdiction:
  - Software development group may have its own agenda.

    E.g. really interested in Unix, wants to use a particular web techniology, etc.
  - Customer may not the same as the user:

    Users may not be empowered to insist that developers meet their needs.
    E.g. UM administrative software - we don't get to choose.
- Management can default to inappropriate process like waterfall model.
  - Administrative advantages.
  - Relieves some conflict situations, temporarily.

Concluding Remarks

Software development is difficult!

- For both technical and non-technical reasons.
- Possibly the most difficult activity the human species has created for itself.
- We don't really know how to do it well!

What kind of organization are you in?

- What control does it have over what gets built?

What kind of development process is being followed?

- Do you know what it is?
- Does it really suit the needs of the situation?

It might help if you understand what's going on.

- What constraints apply in your professional situation?
- Should you keep your resume up to date?

References

Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computer*, 21 , 5, 61-72.

Grudin, J. (1991). Interactive systems: Bridging the gaps between developers and users. *IEEE Computer*, 24, 59-69

Grudin, J. (1991). Systematic sources of suboptimal interface design in large product development organizations. *Human-Computer Interaction*, 6, 147-196.