The 10 Most Important Ideas in Software Development

© 2006 Construx Software Builders, Inc.
All Rights Reserved.

www.construx.com

Most Key Ideas Are Not New

Q: What are the most exciting/promising software engineering ideas or techniques on the horizon?

A: I don’t think that the most promising ideas are on the horizon. They are already here and have been here for years but are not being used properly.

— David L. Parnas
#1

Software Development Work is Performed by Human Beings

Cocomo II’s View of Software Project Influences

<table>
<thead>
<tr>
<th>Factor</th>
<th>Influence Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Complexity</td>
<td>2.00</td>
</tr>
<tr>
<td>Analyst Capability (general)</td>
<td>1.76</td>
</tr>
<tr>
<td>Programmer Capability (general)</td>
<td>1.63</td>
</tr>
<tr>
<td>Time Constraint</td>
<td>1.59</td>
</tr>
<tr>
<td>Personnel Continuity</td>
<td>1.56</td>
</tr>
<tr>
<td>Multi-site Development</td>
<td>1.54</td>
</tr>
<tr>
<td>Required Software Reliability</td>
<td>1.52</td>
</tr>
<tr>
<td>Documentation Match to Lifecycle Needs</td>
<td>1.50</td>
</tr>
<tr>
<td>Applications Experience</td>
<td>1.50</td>
</tr>
<tr>
<td>Use of Software Tools</td>
<td>1.50</td>
</tr>
<tr>
<td>Platform Volatility</td>
<td>1.49</td>
</tr>
<tr>
<td>Storage Constraint</td>
<td>1.48</td>
</tr>
<tr>
<td>Process Maturity</td>
<td>1.46</td>
</tr>
<tr>
<td>Language and Tools Experience</td>
<td>1.43</td>
</tr>
<tr>
<td>Database Size</td>
<td>1.42</td>
</tr>
<tr>
<td>Platform Experience</td>
<td>1.40</td>
</tr>
<tr>
<td>Architecture and Risk Resolution</td>
<td>1.38</td>
</tr>
<tr>
<td>Precededness</td>
<td>1.33</td>
</tr>
<tr>
<td>Developed for Reuse</td>
<td>1.31</td>
</tr>
<tr>
<td>Team Cohesion</td>
<td>1.29</td>
</tr>
<tr>
<td>Development Flexibility</td>
<td>1.26</td>
</tr>
</tbody>
</table>
Importance of Human Influences

- Human Influences make a 14x difference in total project effort & cost, according to Cocomo II
- Capability factors alone make a 3.5x difference
- Experience factors alone make a 3.0x difference

Why Do These Variations Exist?

- Experience?
- Technology knowledge?
- Business knowledge?
- Personal processes?
This Just In …

Newsweek

Human Beings Exhibit the Same Variations in Performance That Programmers Do!

Conclusions You Can Take to the Bank

With 20x differences in individuals and 10x differences in teams commonly reported…

- Technical successes of Google, Amazon, Microsoft and similar companies are not accidents
- Recruiting top staff is easily cost justified
- Even elaborate staff retention programs are also easily cost justified
Incrementalism is Essential

Definition: “The use of a series of regular additions or contributions”

An “incremental” approach is one that is done a little bit at a time.

The final result may be completely mapped out in advance
Conclusions You Can Take to the Bank

What do you get from incrementalism?
- Feedback! (on the software itself)
- Feedback! (on the dev process)
- Feedback! (on the people/dev capability)
- Ability to adapt

Or, Another Perspective on Feedback …

If the map and the terrain disagree, trust the terrain!
Iteration is Essential

Definition: “Recital or performance a second time; repetition”

An “iterative” approach is one that converges to a solution a little bit at a time

The result is not known in advance
News Flash!

People Are Not Computers

Practice Helps!

And making small mistakes that prevent large mistakes helps too
This Just In …

One Definition:
Doing the same thing over and over and expecting different results.

One More Definition:
Not doing the same thing over because you think the results won’t change.

In the Midst of These Important Ideas, Some Are Just Plain Silly

Software Silly Meter

Mildly Serious

Somewhat Silly

Incredibly Serious

Incredibly Silly

Neutral
Silly Idea: The only two development options in existence are “Iterate Nothing” and “Iterate Everything”

Conclusions You Can Take to the Bank

- Some practices derive their power from incrementalism (doing a little bit at a time)
- Some practices derive their power from iteration (repeating the same task)
- You can iterate within an activity or phase (e.g., within requirements)
- You can iterate across any pair of activities or phases (e.g., requirements & architecture)
- You can iterate across entire development cycles
- The degree of iteration can vary from practically 0-100% either within or across activities
#4

The Cost To Fix A Defect Increases Over Time

Defect Cost Increase (DCI)

Activity in which a Defect Is Introduced

Requirements

Architecture

Construction

Requirements

Architecture

Construction

System test

Post-Release

Activity in Which a Defect Is Detected

Average Cost to Correct
**Notable Historical Mistakes**

**Greatest Mistakes in History**

**U.S. News**

- **480 BC:**
  - “This is a nice gift!”

- **1999:**
  - “In the New Economy, companies won’t need to make a profit to be successful.”
Notable Historical Mistakes

2006: "Agile projects aren't affected by Defect Cost Increase."

Conclusions You Can Take to the Bank

- Defect creation is a function of effort
- Defect detection and removal is a function of QA activities
- Fix more defects earlier!
- Use practices that reduce the magnitude of DCI
#5

There's an Important Kernel of Truth in the Waterfall Model

Remember This?

“I’m Not Bad. I’m Just Drawn That Way.”
--Jessica Rabbit
**Intellectual Phases**

Focus

**Schedule**

- Discovery
- Invention
- Construction

**Cost of Overlapping Intellectual Phases**

- Overlap =
  - Dependencies
  - Uncertainty
  - Risk
  - Rework
  - Higher costs
  - Longer schedules
  - Lower quality
Conclusions You Can Take to the Bank

- Some degree of wickedness is inevitable. Plan for it.
- Much wickedness is avoidable. Plan for that, too.
- Attack wickedness actively, especially via incremental and iterative approaches.

#6

Ability to Create Useful Software Estimates Can be Improved Over Time (The Cone Of Uncertainty)
Conclusions You Can Take to the Bank

- Estimation must be iterative
- Project planning must be incremental
- An estimate isn’t meaningful unless it contains a description of its inaccuracy
The Most Powerful Form of Reuse is *Full* Reuse

History of Reuse

- First idea was to reuse code
- Later idea was to reuse code + design
- Current idea is to reuse as much as possible, including processes and plans
Effect of Adding Process the First Time (What I Wrote in SPSG in 1997)

Effect of Adding Process the First Time (What I Think Now)
Effect of Adding Reused Processes

![Chart showing the effect of adding reused processes]

Conclusions You Can Take to the Bank

Consider reusing any or all of these:
- Coding standards
- Change control policies
- Estimation procedures
- Formats & outlines of project plans, requirements doc, design docs, QA plan, test plan, etc.
- Checklists for plans, estimates, change control, inspections, QA, etc.
- Roles & responsibilities
- Training
Risk Management Provides Critical Insights into Many Core Software Development Issues

Risk Management Type 1: Extrinsic

- Added on to the project primarily for purposes of risk management
- Examples of Extrinsic Risk Management
  - Top 10 Risks list
  - Risk management plans
  - Risk officer
  - Etc.
Risk Management Type 2: Intrinsic

- Built into the project for other reasons; risk reduction is an additional benefit
- Examples of intrinsic risk management
  - Active project tracking
  - UI Prototyping
  - End-user involvement
  - Incremental delivery
  - Upstream technical reviews
  - Etc.

A View of Software Risk Reduction

Typical Relationship between Planned Work and Variable Work:

![Bar chart showing typical relationship between Planned Work and Unplanned, Variable Work (typically >50% of total)]

Better Relationship:

![Bar chart showing better relationship between Planned Work, Planned "Overhead", and Unplanned, Variable Work]
Conclusions You Can Take to the Bank

- **Risk** is the key to many tough decisions in project management:
  - What is the best lifecycle model?
  - How much requirements work is enough?
  - How much design work is enough?
  - Can you use junior staff instead of senior staff?
  - Should you do design reviews? Code reviews?
  - How much schedule buffer do you need?
A Silly View of Risk …

- “We’re an entrepreneurial company. We can’t be afraid of risk”
- Not separating business risk from project risk from product risk

#9

Different Kinds of Software Call For Different Kinds of Software Development (The “Toolbox”)
**Examples of Overreaching Claims**

“The pace of information technology (IT) change is accelerating and agile methods adapt to change better than disciplined methods therefore agile methods will take over the IT world.”

“Software development is uncertain and the SW-CMM improves predictability therefore all software developers should use the SW-CMM.”


---

**Time for More Silliness**

- There is one single development approach that works best for all projects.
This is a saw ...

... this is also a saw ...
... and so is this ...

... and so is this ...
... and so is this ...

... and so is this ...

Construx

“Software Development Best Practices”
... and so is this ...

... and so is this …
News Flash!

SOFTWARE DEVELOPMENT IS AS DIFFICULT AS SAWING!

MARTIAN RESEARCHERS MAKE KEY DISCOVERY!

Aliens donate research results to Software Community

Different Projects Call for Different Development Approaches!

Conclusions You Can Take to the Bank

- Which software development approach works best depends on the kind of project
#10

Software Engineering Body of Knowledge (SWEBOK)

The SWEBOK (Software Engineering Body of Knowledge)

- Software Configuration Management
- Software Construction
- Software Design
- Software Engineering Management
- Software Engineering Process
- Software Maintenance
- Software Quality
- Software Requirements
- Software Testing
- Software Tools and Methods
Effect of the SWEBOK

To organize something is to understand it.
– Aristotle

- The main contribution of the SWEBOK is to bring clarity to software development research, discussions, and application

Alternatives to the SWEBOK

- Maybe software is so complicated only gurus can understand it?
- Perhaps software can’t be reduced to words, and you just have to trust the developers to do the right thing?
- Maybe good development practices should be kept secret so they don’t fall into the wrong hands
Alternative #1 to the SWEBOK

I could try to explain software to you, but you wouldn't understand.

Alternative #2 to the SWEBOK

Don't ask questions. Just have faith in your team.
Alternative #3 to the SWEBOK

I could explain software to you, but I’d have to kill you.

Conclusions You Can Take to the Bank

SWEBOK provides a wide spectrum of support for software development practices:
- Defined, reusable software development processes
- Academic curriculums
- Career development
- Professional certification
- Employment interviewing
- Technical skills inventory

And we’re just getting started!
Is the SWEBOK the Ultimate Answer?

“Truth will sooner come out of error than from confusion.”
– Francis Bacon

Conclusions