Software Development's Low Hanging Fruit

www.construx.com

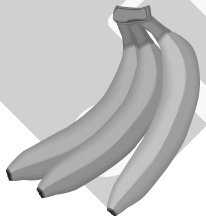
Construx

Delivering Software Project Success

Why Talk About Low Hanging Fruit?

Why Talk About Low Hanging Fruit?

Numerous Good Practices Have Existed for Decades



Best Practices
(year first available)

- **Project planning and management practices**
  - **Automated estimation tools (1973)**
  - **Evolutionary delivery (1988)**
  - **Measurement (1977)**
  - **Productivity environments (1984)**
  - **Risk-management planning (1981)**
- **Requirements engineering practices**
  - **Change board (1978)**
  - **Throwaway user interface prototyping (1975)**
  - **JAD sessions (1985)**

Best Practices
(year first available, cont.)

- **Design practices**
  - **Information hiding (1972)**
  - **Design for change (1979)**
- **Construction practices**
  - **Source code control (1980)**
  - **Incremental integration (1979)**
- **Quality assurance practices**
  - **Branch-coverage testing (1979)**
  - **Inspections (1976)**
- **Process improvement**
  - **SW-CMM (1987)**
  - **Software Engineering Process Groups (1988)**

Why Talk About Low Hanging Fruit?

ROI of Good Software Practices is Well Established

ROI for Selected Practices

Practice	12-month ROI	36-month ROI
Formal code inspections	2.5	12
Formal design inspections	3.5	10
Cost and quality estimation tools	2.5	12
JAD Workshops	2.3	7.5
Process assessments	1.5	6.0
Management training	1.2	5.5
Technical staff training	0.9	5.0
Prototyping (full)	2.0	5.0

Source: Capers Jones, Assessment and Control of Software Risks, Prentice Hall, 1994.

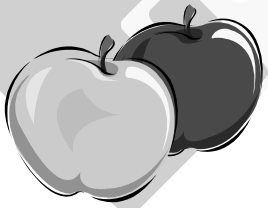
ROI

- Improved software practices pay an average ROI of 5-to-1 (including false starts), and continued improvement is sustainable for many years
- The best organizations have sustained ROIs of 9-to-1 on software improvement initiatives for many years

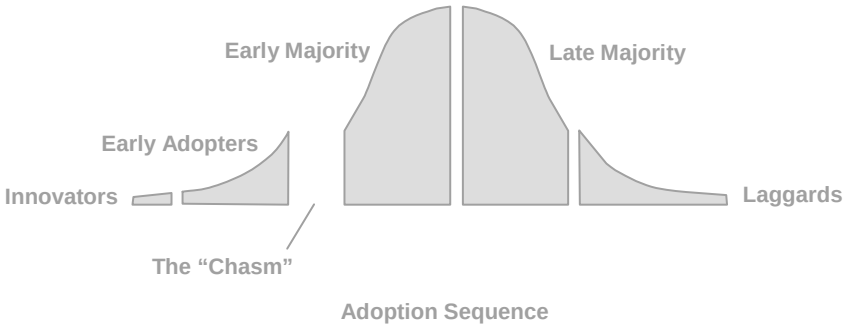
Source: James Herbsleb, et al, "Benefits of CMM Based Software Process Improvement: Initial Results," Pittsburgh: Software Engineering Institute, Document CMU/SEI-94-TR-13, August 1994.

Why Talk About Low Hanging Fruit?

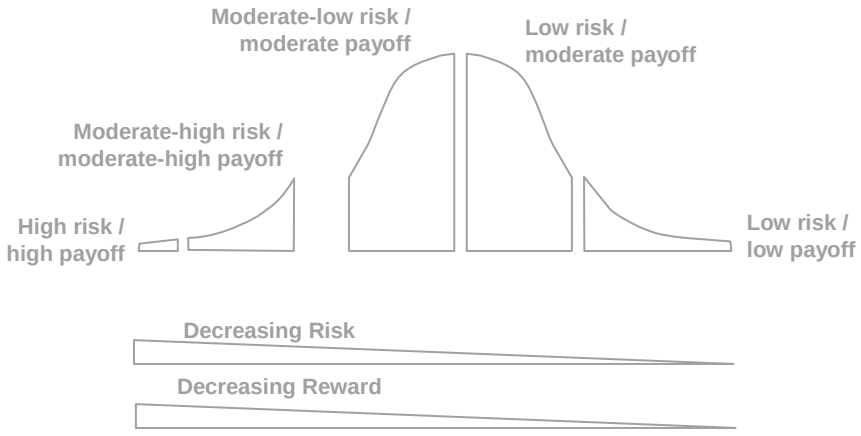
These Practices *Should Have Been* Adopted Long Ago...



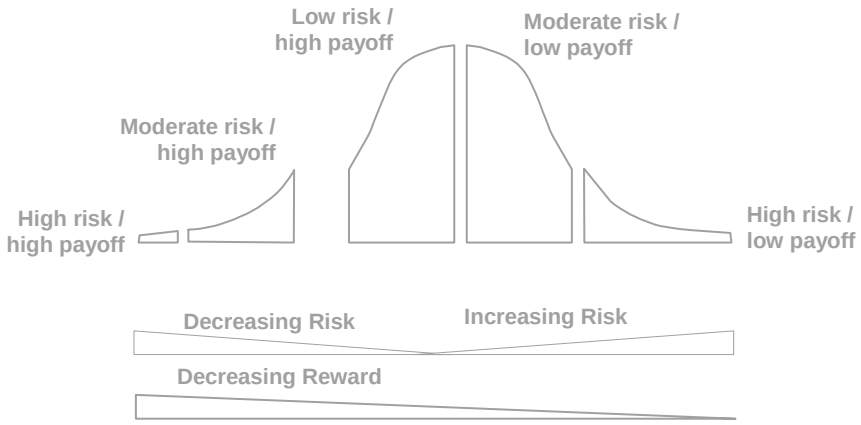
Cycle for Diffusion of Innovations



Normal Risk/Reward Structure



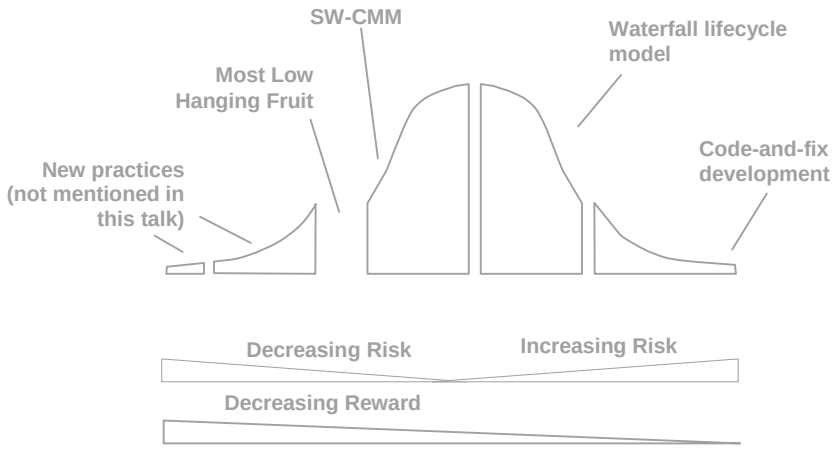
Software's Unusual Risk/Reward Structure



Why Talk About Low Hanging Fruit?

... but Many Good Practices Have Not Been Commonly Adopted

Some Software Examples



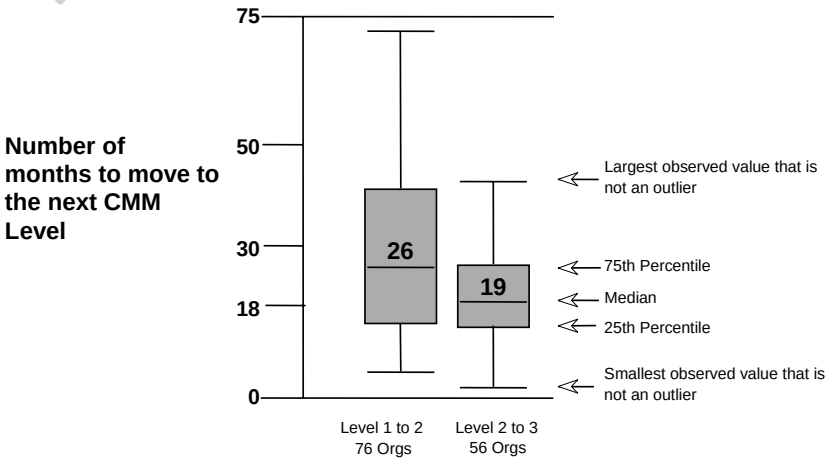
State of the Practice

- **Lots of proven practices are available**
- **Risk of not using these practices is substantially higher than of using them**
- **Many of these tried-and-true practices are readily available, *easy* to adopt, and provide immediate returns**

Why Talk About Low Hanging Fruit?

Does It Seem Like We're Always Talking About *Long Term* Improvements?

Schedule Required to Move Up One CMM Level



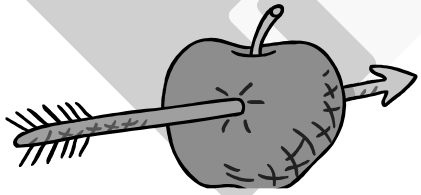
Example Process Improvement Results

Median Results (13 organizations)

- **Duration: 3.5 years**
- **Productivity gain per year: 35% (185% total)**
- **Schedule reduction per year: 19% (52% total)**
- **Reduction in post-release defect reports per year: 39% (82% total)**
- **Business Value of the Investment: 5.0 to 1**

Why Talk About Low Hanging Fruit?

A Low Hanging Fruit Hypothesis



Hypothesis:

The industry focus on "long term improvements" has created the impression that improvements are attainable *only* in the long term. In fact, many significant improvements are attainable in the short term!

Low Hanging Fruit (LHF)

Criteria for LHF

- **Low cost of adoption**
- **Good or very good chance of first-time success**
- **Excellent chance of long term success**
- **Short time to positive ROI**

Candidates for LHF

- Software Best Practices
- Software Fundamentals that aren't currently being used

LHF Candidates, part 1

- 4GLs
- Architectural design
- Buy vs. build planning
- Change board
- Cleanroom development
- Coding standards
- Customer orientation
- Daily build and smoke test
- Defect tracking, full lifecycle
- Designing for change
- Education, management
- Education, technical staff
- Error-prone modules, identification of
- Estimating tools, use of automated
- Estimation and scheduling, accurate
- Evolutionary-delivery lifecycle model
- Evolutionary-prototyping lifecycle model
- Feature-set control

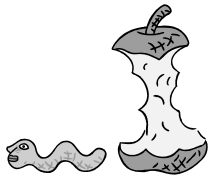
LHF Candidates, part 2

- Goal setting
- Hiring top talent
- Inspections
- Incremental Planning
- Incremental Integration
- Joint Application Design (JAD)
- Lifecycle model selection
- Measurement
- Milestones, miniature
- Minimal specification
- Motivation
- Outsourcing
- Planning tools, automated
- Principled negotiation
- Productivity environments
- Productivity tools
- Rapid-development languages (RDLs)

LHF Candidates, part 3

- Requirements scrubbing
- Reuse
- Risk management, active
- Signing up
- Software configuration management, full
- Software engineering process group (SEPG)
- Source code control
- Spiral lifecycle model
- Staff specialization
- Staged-delivery lifecycle model
- Team structure, matching to project type
- Test-first coding
- Theory-W management
- Throwaway prototyping
- Timebox development
- Tools group
- Top-10 risks list
- Project Tracking, Active
- Up-front Design
- Up-front Planning
- Up-front Requirements
- User-interface prototyping

Is *Everything* Low Hanging Fruit?



- There are *many* good candidates for LHF (58!)
- What will constitute LHF from one organization to the next will vary
  - Low cost of adoption
  - Good or very good chance of first-time success
  - Excellent chance of long term success
  - Short time to positive ROI

Examples of Fruit That Isn't Low Hanging

Practice	Why Not LHF?
RUP	❖ More like a whole tree than individual fruit ❖ Not low cost to adopt ❖ Not high chance of first-time success
Spiral Lifecycle Model	❖ Not a high chance of first-time success
Developing Code for Reuse	❖ Not a short time to positive ROI ❖ Not a high chance of first-time success or long-term success
Pair Programming	❖ No evidence of positive ROI ❖ Not a high chance of long-term success

Examples of Fruit That Isn't Low Hanging (cont.)

Practice	Why Not LHF?
CASE Tools	❖ Not low cost to adopt ❖ Not a high chance of first-time success or long-term success
Statistical Process Control	❖ It's great fruit, just not low hanging, i.e., not a short time to positive ROI
Major Milestones	❖ Not a high chance of first-time success or long-term success
Use Cases	❖ Not a high chance of first-time success or long-term success

Where Do You Start?

Where Do You Start?

Depends on who "you" are:

- **Developer**
- **Technical Lead**
- **Manager**
- **Organization (Executive)**

A Developer's LHF

Assumptions

- **LHF cannot require more than one contributor**
- **LHF doesn't create any direct expense**
- **LHF must not create "atomic" work that would show up on a task list**
- **LHF is minimally visible to management**

A Developer's LHF

Can Own:

- Coding Standards
- Test-First Coding
- Designing for Change
- Incremental Integration
- Throwaway Prototyping
- Up-Front Design

Can Contribute to:

- Error-Prone Modules, Identification of
- Defect Tracking, full lifecycle
- Daily Build and Smoke Test
- Architectural Design
- User-Interface Prototyping
- Evolutionary Delivery
- Source Code Control Tool

A Developer's LHF

Observations

- ROI of developer-level LHF is relatively low, but lots of LHF is easily within reach
- Lots of LHF is partially reachable
- Construx has found *violent agreement* in upper management that developers should be using LHF
- Most of these practices are *invisible* to upper management

A Technical Lead's LHF

Assumptions

- **LHF is primarily technical in nature**
- **LHF doesn't create any direct expense**
- **LHF may require more than one contributor**
- **LHF may affect detailed task assignments, task ordering, etc.**
- **LHF is minimally visible to upper management, the customer, or other project stakeholders**

A Technical Lead's LHF

Can Own:

- **Coding Standards**
- **Test-First Coding**
- **Designing for Change**
- **Incremental Integration**
- **Throwaway Prototyping**
- **Up-Front Design**
- **Error-Prone Modules, Identification of**
- **Daily Build and Smoke Test**
- **Defect Tracking, full lifecycle**
- **Architectural Design**
- **User-Interface Prototyping**
- **Evolutionary Delivery**
- **Lifecycle Model Selection**
- **Inspections**
- **Requirements Scrubbing**
- **Planning, Incremental**
- **Change Control, Formal**
- **Top-10 Risks List**

A Technical Lead's LHF (cont.)

Can Contribute to:

- **Source Code Control Tool**
- **Miniature Milestones**
- **Timebox Development**
- **Up-Front Requirements**
- **Planning, Up-Front**

A Technical Lead's LHF

Observations

- **Most LHF is reachable by the technical lead**
- **Again, most LHF is invisible to executive management, and *implicitly supported* by executive management**
- **Most of it will work better with project management and executive support**
- **Problem at this level is really choosing which of numerous options is best—that's what expert help is for!**

A Manager's LHF

Assumptions

- LHF is not highly technical in nature
- LHF may create direct expenses
- LHF may affect more than one contributor
- LHF may affect detailed task assignments, task ordering, etc.
- LHF may be visible to upper management, the customer, or other project stakeholders
- LHF does not require multiple-project span of control

A Manager's LHF

Can Own:

- Inspections
- Requirements Scrubbing
- Planning, Incremental
- Change Control, Formal
- Top-10 Risks List
- Feature-Set Control
- Source Code Control Tool
- Miniature Milestones
- Timebox Development
- Up-Front Requirements
- Theory-W Management
- Planning, Up-Front
- Buy Vs. Build Planning
- Joint Application Design (JAD)

A Manager's LHF (cont.)

Can Contribute to:

- **Throwaway Prototyping**
- **Defect Tracking, full lifecycle**
- **Up-Front Design**
- **User-Interface Prototyping**
- **Evolutionary Delivery**
- **Lifecycle Model Selection**
- **Staff Specialization**
- **Education, Technical Staff**
- **Education, Management**

A Manager's LHF

Observations

- **Most LHF is reachable by the manager**
- **Most of the detailed work on the LHF will need to be done by technical staff**
- **Main problem here again is too many choices**

An Organization's LHF

Assumptions

- **LHF is not highly technical in nature**
- **LHF may create direct expense**
- **LHF may affect more than one project**
- **LHF may degrade single-project performance to boost overall organization performance**

An Organization's LHF

Can Own:

- **Staff Specialization**
- **Education, Technical Staff**
- **Education, Management**
- **Planning, Up-Front**
- **Buy Vs. Build Planning**
- **Change Control, Formal**
- **Top-10 Risks List**
- **Joint Application Design (JAD)**

Can Contribute to:

- **Planning, Incremental**
- **Defect Tracking, full lifecycle**
- **Timebox Development**
- **Up-Front Requirements**
- **Theory-W Management**

An Organization's LHF

Observations

- **Most organization-level LHF has high ROI, but longer lead times (opposite of developer-level LHF)**
- **Most detailed work still needs to be done by technical staff**

Bottom Line

- **Most of the work to harvest LHF occurs at the technical lead levels, but that work significantly benefits from support at the manager, organization, and developer level**

A Low Hanging Fruit Basket

LHF that are the Fastest to Adopt

- Coding Standards
- Daily Build and Smoke Test
- Source Code Control Tool
- Top 10 Risks List
- User Interface Prototyping

LHF that are the Lowest Risk to Adopt

- Daily Build and Smoke Test
- Defect Tracking
- Education, Technical Staff
- Inspections
- Planning, Incremental
- Source Code Control Tool
- Top 10 Risks List

LHF that will Not be Resisted by Individual Contributors

- Change Control, Formal
- Planning, Incremental
- Source Code Control Tool
- Top 10 Risks List
- Up-front Design
- Up-front Requirements

LHF that will Not be Resisted by Upper Management

- Coding Standards
- Incremental Integration
- User Interface Prototyping
- Defect Tracking
- Up-Front Design
- Architectural Design
- Inspections
- Test-First Coding
- Designing for Change
- Error-Prone Modules, Identification of
- Daily Build and Smoke Test
- Evolutionary Delivery
- Lifecycle Model Selection
- Planning, incremental
- Planning, up-front

Executives can't resist LHF they can't see!

Summary

Good News

- **Practically everything in software development is *fundamentals*. There are no advanced practices.**

  (This isn't quite true, but it's pretty close)
- **The worse off your organization is now, the higher the ROI of good practices will be!**
- **If you're not currently making substantial use of good practices, focus on doing *anything*; don't let "the best" become the enemy of "the good"**

More Good News

- Low hanging fruit is just the beginning. Once you harvest that, there is still more fruit higher up
- LHF can help an organization "learn how to change"—which is one of the hardest aspects of longer-term process improvement
- Construx specializes in helping organizations identify which LHF is best for them

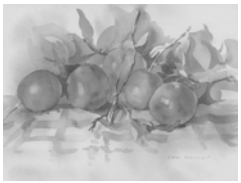
Construx Consulting Support

- Assessment and Recommendations (we identify *your* LHF)
- Improvement Roadmaps
- Project Chartering Workshops
- Project Planning Workshops
- Requirements Workshops
- Project Scoping & Estimation Workshops
- Best-Practice Deployment Workshops
- Project Recovery

www.construx.com

Construx

Delivering Software Project Success



- **Training**
- **Coaching & Consulting**
- **Software Projects**

sales@construx.com
www.construx.com
+1 (425) 636-0100