Apple and the Global Environment

At Apple, environmental considerations are an integral part of our business practices. From the earliest stages of design through manufacturing, use and recycling, we take care to minimize the environmental impact of our activities.

Lifecycle phases and their environmental consideration:

- **Product Design**
  - Selecting environmentally acceptable materials and substances.
  - Developing features to optimize the energy efficiency of our systems and to facilitate recycling.

- **Manufacturing**
  - Reducing waste — water, emissions, by-products — from the manufacturing process.
  - Continually improving the energy efficiency of our manufacturing facilities.

- **Use**
  - Providing software tools that enable users to control the energy-saving features of their systems.

- **Recycling/EOL**
  - Providing product recycling programs for our customers worldwide.
  - Diverting a high percentage of returned equipment from landfill.

Learn More
At Apple, our commitment to the environment is an ongoing effort. To read more about the steps we take to foster environmental improvements in our products and the way we do business, visit our environmental [website](#).

- Between the first-generation and current-generation iMac, improvements in CPU power management enabled an 87% decrease in sleep-mode power usage.
- Our Customer-Installable Parts program puts the power of system expandability in the consumer’s hands, helping to extend the useful life of a computer.

Three areas in which we focus considerable attention are materials management, energy efficiency and recycling, which are key elements of a product’s lifecycle. Our efforts in these areas have resulted in some impressive environmental successes.
# Computing in a Nutshell

## Operating Systems
- Linux
- Windows
- Unix
- MacOS
- PalmOS
- Symbian

## Programming
- Assembly
- C/C++/C#
- Java/J++/J#
- Fortran
- BASIC
- PHP

## Platforms
- Desktop
- Laptop
- Handheld
- Server
- Embedded
- Networked/Parallel
The Cross-Functional Team

PRODUCT MANAGER
Defines the product's marketing and functional requirements.

PROJECT MANAGER
Manages team relationships and engineering schedule.

SOFTWARE ENGINEER
Creates code per assignment from project manager.

BUILD ENGINEER
Manages source code versions and creates software packages.

TEST ENGINEER
Creates and executes test plans. Tracks bugs and regresses fixes.

SUPPORT ENGINEER
Monitors bugs and plans support strategy. Anticipates user experience.

TECHNICAL WRITER
Provides user documentation.

PRODUCTION MGR.
Creates packages for user. Creates and enumerates BOMs.

SALES MANAGER
Interface with the user.
Software Engineering

• Product Life Cycles

![Product Lifecycle Curve](image.png)
Software Engineering

- Product Life Cycles

```
Egg Headz by D.A. Weeks

ANALYSIS

DESIGN

IMPLEMENTATION

END USER APPROVAL

THE JOYS OF THE PRODUCT LIFE CYCLE

THIS IS CRAP!
```
**Project Management Process**

**STEP 1**
Identify and frame the problem or opportunity.

**STEP 2**
Identify and define the best project solution.

**STEP 3**
Identify task and resource requirements.

**STEP 4**
Prepare the control schedule and resource allocation.

**STEP 5**
Estimate project costs and prepare a project budget.

**STEP 6**
Analyze risk and establish stakeholder relationships.

**STEP 7**
Maintain control and communicate as needed during execution.

**STEP 8**
Manage to an orderly close-out.

Skill Requirements:
- Project management process
- Interpersonal and behavioral
- Technology management
- Desired personal traits

*After Heerkens, 2002*
Software Engineering

**SKELETON**
Define the logic, data and shell structures. No features included.

**PROTOTYPING**
Proof-of-concept code. Primitive features.

**DEVELOPMENT**
Development of features as stated in requirements documents.

**α CANDIDATES**
Features are stable enough for testing.

**β CANDIDATES**
Whole package is stable enough for testing.

**FINAL CANDIDATES**
Whole package is stable enough for compatibility testing.

**GM CANDIDATES**
Whole package meets requirements for production.

**FROZEN PRODUCT**
Package is designated as to be sent into mass production.

**RELEASED PRODUCT**
Product that is released to users.
Software Quality Assurance

**DESIGN TESTING**
- Testing design ideas.

**WHITE BOX TESTING**
- Use understanding and access to the source code to develop test cases.

**BLACK BOX TESTING**
- Test planning
- Acceptance testing
- Initial stability assessment
- Function test, system test, verification, and validation
- Beta testing
- Integrity and release testing
- Final acceptance testing and certification

**REGRESSION TESTING**
- Make sure that a fix does what it's supposed to do.
- Make sure that the change didn't disturb anything else.
# Bugs

<table>
<thead>
<tr>
<th>REPORTING &amp; ANALYZING</th>
<th>TRACKING</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TESTER</strong></td>
<td>Team members can track the status of a bug.</td>
</tr>
<tr>
<td>Explain how to reproduce the problem.</td>
<td>Project Manager produces periodic bug reports.</td>
</tr>
<tr>
<td>Analyze the error so you can describe it in a minimum number of steps.</td>
<td>Project Manager leads bug review meetings to decide whether bugs are fixed, not fixed, causing a new problem, irreproducible, deferred, or not a bug.</td>
</tr>
<tr>
<td>Write a report that is complete, easy to understand, and non-antagonistic.</td>
<td>Deferrals are used to &quot;close&quot; all lingering bugs so a product can ship to a user.</td>
</tr>
<tr>
<td>Assigns severity rating.</td>
<td>Tracking system metrics is used to compare growing number of bugs versus rate of closure.</td>
</tr>
<tr>
<td><strong>PROJECT MANAGER</strong></td>
<td></td>
</tr>
<tr>
<td>Assigns bug to a programmer.</td>
<td></td>
</tr>
<tr>
<td>Assigns priority rating and current status.</td>
<td></td>
</tr>
</tbody>
</table>
Numerical Methods

15.2 Fitting Data to a Straight Line

A concrete example will make the considerations of the previous section more meaningful. We consider the problem of fitting a set of $N$ data points $(x_i, y_i)$ to a straight-line model

$$y(x) = y(x; a, b) = a + bx$$  \hspace{1cm} (15.2.1)

$$\Delta \equiv SS_{xx} - (S_x)^2$$

$$a = \frac{S_{xx}S_y - S_xS_{xy}}{\Delta}$$  \hspace{1cm} (15.2.6)

$$b = \frac{SS_{xy} - S_xS_y}{\Delta}$$

Equation (15.2.6) gives the solution for the best-fit model parameters $a$ and $b$.  

Press et al, 1992
void fit(float x[], float y[], int ndata, float sig[], int mwt, float *a, float *b, float *sigma, float *sigb, float *chi2, float *q)
Given a set of data points x[1..ndata], y[1..ndata] with individual standard deviations sig[1..ndata], fit them to a straight line \( y = a + bx \) by minimizing \( \chi^2 \). Returned are a, b and their respective probable uncertainties sigma and sigb, the chi-square chi2, and the goodness-of-fit probability q (that the fit would have \( \chi^2 \) this large or larger). If mwt=0 on input, then the standard deviations are assumed to be unavailable: q is returned as 1.0 and the normalization of chi2 is to unit standard deviation on all points.
{
    float gammq(float a, float x);
    int i;
    float wt,t,sxoss,sx=0.0,sy=0.0,st2=0.0,ss,sigdat;

    *b=0.0;
    if (mwt) {
        ss=0.0;
        for (i=1;i<=ndata;i++) {
            wt=1.0/SQR(sig[i]);
            ss += wt;
            sx += x[i]*wt;
            sy += y[i]*wt;
        }
    } else {
        for (i=1;i<=ndata;i++) {
            sx += x[i];
            sy += y[i];
        }
        ss=ndata;
    }
    Accumulate sums ...
    ...with weights
    ...or without weights.
}
Numerical Methods

\[
sxoss = sp / ss;
\]

if (mwt) {
    for (i=1; i<=ndata; i++) {
        t = (x[i] - sxoss) / sig[i];
        st2 += t * t;
        *b += t * y[i] / sig[i];
    }
} else {
    for (i=1; i<=ndata; i++) {
        t = x[i] - sxoss;
        st2 += t * t;
        *b += t * y[i];
    }
}

*b /= st2;
*a = (sy - sx * (*b)) / ss;
*sigma = sqrt(((1.0 + sx * sx) / (ss * st2)) / ss);
*sigb = sqrt(1.0 / st2);

Solve for \(a\), \(b\), \(\sigma_a\), and \(\sigma_b\).

Press et al, 1992
9.4 Newton-Raphson Method Using Derivative

Perhaps the most celebrated of all one-dimensional root-finding routines is Newton’s method, also called the Newton-Raphson method. This method is distinguished from the methods of previous sections by the fact that it requires the evaluation of both the function $f(x)$, and the derivative $f'(x)$, at arbitrary points $x$. The Newton-Raphson formula consists geometrically of extending the tangent line at a current point $x_i$ until it crosses zero, then setting the next guess $x_{i+1}$ to the abscissa of that zero-crossing (see Figure 9.4.1). Algebraically, the method derives from the familiar Taylor series expansion of a function in the neighborhood of a point,

$$f(x + \delta) \approx f(x) + f'(x)\delta + \frac{f''(x)}{2}\delta^2 + \ldots \quad (9.4.1)$$

For small enough values of $\delta$, and for well-behaved functions, the terms beyond linear are unimportant, hence $f(x + \delta) = 0$ implies

$$\delta = -\frac{f(x)}{f'(x)} \quad (9.4.2)$$
Numerical Methods

```c
rts=0.5*(x1+x2);
dxold=fabs(x2-x1);
dx=dxold;
(*funcd)(rts,&f,&df);
for (j=1;j<=MAXIT;j++) {
    if (((rts-xh)*df-f)*((rts-xl)*df-f) > 0.0) Bisect if Newton out of range,
        || (fabs(2.0*f) > fabs(dxold*df)) ( or not decreasing fast enough.
        dxold=dx;
dx=0.5*(xh-xl);
    if (x1 == rts) return rts;
    dxold=dx;
dx=f/df;
temp=rts;
    rts -= dx;
    if (temp == rts) return rts;
} else {
    if (fabs(dx) < xacc) return rts;
    (*funcd)(rts,&f,&df);
    Change in root is negligible.
    Newton step acceptable. Take it.
}
```

Figure 9.4.1. Newton’s method extrapolates the local derivative to find the next estimate of the root. In this example it works well and converges quadratically.

Press et al, 1992
**Roadmap**

- **SANDBOX**
  - Modeling concepts, scales and approaches

- **SANDBOX**
  - Programming languages, software engineering & numerical methods

- **DESIGN**
  - Project Proposal

- **IMPLEMENTATION**
  - Examination of Equilibrium-based Code

- **IMPLEMENTATION**
  - Examination of Reaction Rate-based Code

- **IMPLEMENTATION**
  - Examination of Existing Models for Complex Systems

- **READINESS**
  - Internal Testing and Code Freeze

- **RELEASE**
  - Final Presentations ("Rollout")
# Assignment: Project Proposal

<table>
<thead>
<tr>
<th>A. SCIENTIFIC LITERATURE REVIEW</th>
<th>B. CONCEPTUAL DESIGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide background information on the scientific or engineering area with which the project is related.</td>
<td>Description of compartments and processes. Schematic of model environment.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C. MARKETING REQUIREMENTS</th>
<th>D. FUNCTIONAL REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document how there is a need for the proposed product. Show how there is differentiation from existing products. Use information at <a href="http://www.epri.com/eprisoftware/processguide/srdc.html">http://www.epri.com/eprisoftware/processguide/srdc.html</a> as a guideline.</td>
<td>Provide a description of the software including functional capabilities and user interactions. Include how the user will learn to use your product. Check out <a href="http://www.epri.com/eprisoftware/processguide/fscontents.html">http://www.epri.com/eprisoftware/processguide/fscontents.html</a> as a guideline.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E. TEST PLAN</th>
<th>F. PROJECT SCHEDULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a list of tests that can be applied to the product to make sure that it is functional and usable, and meets the documented requirements.</td>
<td>Create a timeline showing project tasks, resource allocation, durations and relationships. Be sure to declare the frequency of checking the schedule during the project to determine if adjustments to the schedule need to be made.</td>
</tr>
</tbody>
</table>