The Professional Responsibilities of Software Engineers

David Lorge Parnas

Abstract

Registered Engineers are expected to be aware of their responsibilities as professionals. Those who practice Software Engineering often enter that profession without either an engineering education or professional registration. The primary responsibility is to make sure that their products are “fit for use”.

Department of COMPUTING AND SOFTWARE
Software Engineering Programme
“*connecting theory with practice*”

Personal Responsibility, Social Responsibility and Professional Responsibility

Is there a difference?

Can they conflict?

- *Personal Responsibilities* are general obligations towards other individuals; most are shared by all persons (e.g. honesty, concern for others).
- *Social Responsibilities* are responsibilities towards society as a whole. We have a debt to repay because society has supported us when we needed it. (e.g. environmental activism, peace activism, national defence)
- *Professional Responsibilities* are additional responsibilities shared by members of a particular profession (e.g medicine, journalism, or engineering) Usually a code of responsibilities exists.

Professional responsibilities include, but are not limited to, contractual obligations to an employer.

These obligations may *appear* at times to conflict with Personal and Social responsibility.

The primary responsibility of an engineer is always to the safety and well-being of the public.

Computers are Everywhere

Almost all of today's engineering products were designed using computers.

An increasing number of engineering products contain computers.

Almost all of today's software is wrong!

There is growing concern about the quality of software.

There is an intense effort to improve the process of programming.

References

(1) Neumann, P.G. "Computer Related Risks" ISBN 0-201-55895- X, 1995, ACM Press, Addison Wesley

(2) Wiener, L.R. "Digital Woes, Why We Should Not Depend on Software", 1993, ISBN 0-201-62609-8, Addison Wesley

(3) Forrester, Tom, Morrison Perry, "Computer Ethics: Cautionary Tales and Ethical Dilemmas in Computing, The MIT Press.

Why is Software of Growing Importance?

Computers are increasingly powerful yet decreasing in cost because they can be mass produced.

Massed produced means potentially good for everything but not good for anything.

Software is needed to tailor general purpose tool to specific use.

Few hardware design jobs, but many software design jobs.

Hardware designed with discipline by engineers and errors are relatively rare.

Software designed intuitively by all kinds of people; errors the normal case.

Difficulty of software always underestimated. Systematic methods are “not needed”.

Personal Responsibility, Social Responsibility and Professional Responsibility

An Illustration - SDI (Star Wars):

Service on the "Committee on Computing in Support of Battle Management".

Some questions that arose:

(1) Was it honest? (personal responsibility)

(2) Had I made a professional commitment?
Was our activity designing a system that would meet the needs of the customer as required by professional codes?
What should a professional do if the answers were "yes" and "no"?

(3) Was this project good for society?
Should I explain my views to the public?

Some regarded a "Yes" to (3b) as unprofessional.

The conflict in (2) was resolved by a detailed explanation.

Why would I work on Nuclear Plants but not Star Wars?

A Comparison of Technical Problems

Problem Characteristics	SDI	NPGS
Must deal with unknown physical properties, deliberate deception	*YES*	*NO*
Network with unreliable channels	*YES*	*NO*
Can be tested under realistic operating conditions	*YES*	*NO*
Possibility of human intervention during use	*YES*	*NO*
Short real-time deadlines	*YES*	*NO*
Frequent addition of devices	*YES*	*NO*
System easily overloaded in use	*YES*	*NO*
Component failures, statistically correlated	*YES*	*NO*
Precise synchronisation needed	*YES*	*NO*

Above a question of professional responsibility. Questions of "Peace" are issues of social responsibility. Honest claims, personal responsibility.

The Social Responsibility of Scientists And Engineers

"In the land of the blind, the one eyed man is king".

In a world increasingly dependent on science and technology, Scientists and Engineers are the one-eyed people.

The majority of our decision-makers are blind.

Consider the following public issues:

- Can we reduce our energy expenditures without great disruption in people's lives?
- How urgent is the need to reduce the level of greenhouse gasses?
- Should we build more nuclear power generating stations?
- Is it safe to allow nuclear power generating stations to be controlled by computers?
- Can technology help us to reduce the amount of the paper that we use? Should we do that?
- Is it safe to allow computers to control cars and trucks?

Decisions will be taken by non-specialists, but the input will come from people like us. We must give them complete and accurate information.

The Social Responsibilities of Scientists & Engineers

Science and technology are the "black magic" of our age.

We use arcane rituals and obscure terminology.

The public thinks that science can solve any problem if given enough funds.

Public officials share this attitude. They fall for scientific fads.

Buzzwords and big promises, favoured over solid scientific work.

The rewards often go to the illusionists.

The successful do not speak out. The others are ignored ("sour grapes").

Most of us "go along" to get funds.

Don't we have a responsibility to see that society's funds are well used?

In your career you will often have to decide whether or not to participate in a project and, if you decide not to participate, whether you should make your decision public.

The Professional Responsibilities of Engineers

Unfortunately, Software Engineers are not always Engineers.

"Software Engineering" is a shallow course on programming, taught in a science department, not a professional programme in Engineering

Many "software engineers" have no technical education.

Many *could not* be Professional Engineers.

Many confuse software engineering with configuration management.

An Engineer is someone who uses advanced knowledge of science, mathematics, and technology to build objects for use by others.

Most programmers or software engineers, are Engineers, underqualified, unlicensed, and often unprofessional.

They are unaware of their professional responsibilities.

Programmers need to learn about the professional responsibility of engineers.

Why do we have licensed Professional Engineers?

An old system introduced because:

- Some products potentially dangerous. Incompetent designs a danger to public.
- Purchasers and some employers are often unable to judge the competence of designers.
- Competent, conscientious, disciplined professionals want public to distinguish between themselves and others. Bad work by a few damages the reputations and business prospects of all.
- Financial pressures may tempt employers to "cut corners". We are protected better when professional obligations go beyond loyalty or obedience to an employer. Professionals do say "No".

Don't all of these reasons apply to software construction?

What is the Professional Engineering system?

Professional Engineering Societies were established by legislation to assure competence and awareness of professional responsibilities.

Regulations require that certain products be produced or approved by a recognised Professional Engineer.

There is a separate committee to accredit programs. Accreditation is a very serious process.

Graduates of accredited programs have an easier path to recognition as a Professional Engineer.

An exam on responsibilities is required in any case.

Why are "Software Engineers" different?

The result of a "software crisis" (optimism).

Originally dealt with as a scientific problem. The basis of software engineering was not well understood.

First meetings attended by many mathematicians and scientists, few engineers.

Many engineers were still blissfully unaware of the importance of computers in their profession.

The word is "Engineering" used to indicate practical concerns, not a profession.

Professional societies did not take it seriously.

Software Engineering has developed *outside* of the Engineering Community.

It has been left to Computer Science departments, taught by people who are not Engineers.

Because badly designed computer programs are hard to manage, emphasis has been on project management, project scheduling, version control, etc.

Today, the engineering societies are beginning to do what they were always required to do.

What are the obligations of the engineer?

<u>The following are the responsibility of any kind of engineer:</u>

(1) Accept individual responsibility.

- Following orders does not justify approving bad designs.
- One cannot always be a “team player”.
- Professional standards have priority over other pressures.

(2) Solve the real problem

- Look beyond the customer’s opinions.
- Have a precise description of a problem.
- Get that description reviewed before building.

(3) Be honest about capabilities

- Don’t offer technical solutions where there are none.
- Don’t do studies when you already know the answer.

(4) Produce reviewable designs

- No individual is infallible.
- Document to make reviewing easy.

(5) Maintainability

- Produce a product that can be maintained without you. - It’s not your personal product.

Professional Practice in Software Development

Some responses to a critical consultant:

- "Of course it's wrong, but that is what my boss told me to do."
- "We already know the answer, but they will pay us $1,000,000 for the study.
- "It's not the right way, but it's the customer's suggestion."
- "At XYZ corporation, we don't tell our customers that they are wrong, we take their contracts."
- "That's not the real problem, but they asked us to do it."
- "We can't give them what they need, but we'll do the best we can."
- "We've got a deadline; we'll worry about maintainability when we get the maintenance contract.
- "We don't like people criticising our designs!"

These remarks showed that the speakers were unaware of the professional responsibilities of engineers.

Some had not heard of those responsibilities.

Some had no such excuse!

A Simple Example: Pacemakers

Their importance to the user is obvious!

They are also important to those nearby.

They are controlled by software.

- Many modes of operation
- Computer controlled telemetry system
- Data collection
- “Programmable” by remove control
- “When needed” intervention.
- Rate responds to body activity.
- Packaged in a small sealed unit
- Must survive in a “hostile environment”

Clearly the type of device that should be built by engineers.

The program is critical and should be well documented and reviewable.

What Should be Done for Pacemaker Software?

(1) Programmer should have a precise description of the environment and requirements

(2) Black box description should have been produced for review.

(3) Document should have been reviewable and reviewed by Cardiologists.

(4) The code should have been documented in a way that permitted systematic review and revision.

(5) Code should have been subject to systematic inspection.

(6) Doctor should have been provide with well-organised precise documentation that explained the behaviour of the device to him.

<u>All of these things would be expected of a professional engineer.</u>

A Personal Anecdote

Pacemaker "refused" the surgeon's command; neither surgeon nor technician understood why.

I found the explanation in a footnote after several hours of reading

It took 30 minutes to find it the second time.

Engineer responsible could explain the hardware aspects in great detail.

He referred us to a programmer, who could not be found, to explain the code.

Programming had been viewed as a trivial task; Responsible engineer did not review it.

As a result of inadequate review, there are fundamental weaknesses.

- Motion sensor does not measure physical activity
- Expected rate adjustment is inflexible.

The problem solved was not the real problem.

This was a typical software product.

The software was written as it would have been written 25 years ago.

Software used by Professional Engineers

Professional Engineers take responsibility for their products, but, ...

- to design those products they use software that comes with a disclaimer instead of a warranty,
- Professional Engineers belong to a society that enforces codes of professional behaviour,
- they must use tools produced by people who do not belong to such a society.

<u>This cannot be a stable situation!</u>

The "Know How" isn't There!

If we look at other areas of engineering, we know what software engineers should do.

If we look at current practice, those things are not done.

It's not just a matter of lack of will.

It's not just a matter of lack of awareness.

Most programmers do not know how to do the things that they should do.

<u>They do not know how to:</u>

- document requirements in a way that can be reviewed by subject matter experts,
- document code precisely and completely,
- inspect code systematically.

<u>We are trying to give you that "know how".</u>

APEO Software Guidelines

Engineers are responsible for verifying that results obtained by using software are accurate and acceptable.

The engineer should ensure that professional engineering verification of the software's performance exists.

Software reliability requires a disciplined approach to quality assurance.

Functional requirements that cannot be verified, or are inadequately defined, ambiguous or infeasible will lead to the development of an inadequate software product.

Engineers owe a duty of care to the client and the public. That duty of care may be breached if engineers are negligent in rendering services to the client, and they may be liable for damages... That liability may arise if a computer program developed by an engineer and used by a client or third party is defective and causes harm...

Quality Characteristics

Functionality, Maintainability, Reliability, Reviewability, Safety, Usability, Completeness, Modifiability, Robustness

Improving Professionalism in Software Development

Three steps:

(1) Work with Professional Engineering societies.

(2) Develop better educational programmes.

(3) Develop accreditation procedures for Software Engineering programmes.

Why discuss the design process?

We can't tell people how to think!

Nobody will follow the process we define!

but,

We all look for guidance in difficult situations

- Where do we start?
- What should we write?
- When are we done?

The process can affect the quality of the product.

Progress measurement requires a model of the process.

Why is software commonly the product of an irrational process?

We start to build before we know what we want.

We learn what we want as we start to build.

Sometimes the basis of our work is a new technology or implementation concept.

We make decisions without being able to justify them relative to a statement of goals.

We simply do not understand enough to be rational designers.

Why do we want a rational process?

We want to derive programs from their requirements to be sure that they meet those requirements.

We want to be systematic, so that we don't overlook anything.

We want to be able to give a rational explanation of our work to help others understand.

We want our work to be more easily understood, more safely modified.

We want to make good decisions, decisions that won't have to be reversed or revised.

We want to look good.

Bad News:

Any proposed “rational process” will always be an idealisation - We will never really do it.

Good News:

We can fake it!

It pays to fake it!

Why is any proposed rational process an idealisation, a dream?

(1) Customers do not know what they want at the start of the design process.

(2) Even if you knew what you wanted, there are unknowns in the environment that become known only during the development.

(3) The implications of some details only become clear during development.

(4) One must then either backtrack or produce a less-than-ideal design.

(5) At the start of the process, the details are overwhelming.

(6) There are always changes in both intent and environment.

(7) One can only avoid errors by avoiding humans.

(8) We often have strong preconceptions about how to build something.

(9) Reuse of previous work is encouraged.

(10) Some projects are driven by the desire to exploit new technology or machines.

Why should we care about a process we cannot follow?

Documentation that simulates the ideal can be produced.

Understanding the ideal process guides the designers.

Having the model helps us to approach it.

Large organisations benefit from a standard process:

- better reviews
- better progress measurement

A Rational Design Process

(1) A. Establish and document system requirements (black-box view)

(2) Select Hardware Components and Document the System Design

(3) Document the Desired Software Behaviour

(4) Design and document the module structure

(5) Design and document the module interfaces

(6) Design and document the uses hierarchy

(7) Establish process structure guidelines

(8) For each module that is too big to throw away, repeat 4...8, then

(8.1) Design the module internally

(8.2)Code to the internal design

Why do we need requirements documents?

How will it be used?

Decide what to build before starting to build it.

Provide an organised reference document for the software engineers.

Provide a reference document for the Quality Assurance Group.

Specify the constraints for future improvement actions.

Provide input to those who write user manuals and other less-formal documents.

What goes in the system requirements document?

- Everything you need to know to design the system, no more--no less.
- Every statement should be valid for all acceptable product.
- If a product satisfies every statement, it should be acceptable.
- Requirements are “life cycle” requirements.
- Requirements are not “ostrich-eye” requirements.
- Should be an engineering document, not an introduction.
- Should provide descriptions, not stories.
- Areas of incompleteness should be specified.
- Only monitored and controlled environmental variables should be mentioned

What goes in the system design document?

- The characteristics of the computer hardware input and output registers.
- The functional characteristics of the peripheral devices that sense the monitored variables and control the control variables.

The relationship between the input/output variables and the system environmental variables.

What about the software requirements?

- Everything you need to know to write the software, no more--no less.
- Every statement should be valid for all acceptable product.
- If a product satisfies every statement, it should be acceptable.
- Requirements are “total life cycle” requirements.
- Requirements are not “ostrich-eye” requirements.
- Should be an engineering document, not an introduction.
- Should provide descriptions, not stories.
- Areas of incompleteness should be specified.

<u>This information is provided by the two previously described documents.</u>

Decomposing the product into modules

What is a module guide?

- A document describing the responsibilities of individual modules.

Why do we need a module guide?

- To avoid duplication.
- To avoid gaps.
- To help an ignorant maintainer.

What criteria is used?

- Information Hiding - Separation of Concerns
- Things that can change separately are separated.

What does the module guide show?

- Secrets.
- Not Interfaces.
- Not Roles.

Documenting individual module interfaces

Why do we need module interface documents?

- Allows independent development of modules.
- Reduces unintended links between modules.
- Reduces the amount of information one needs to know.

How do we provide an abstract interface description?

- Focus entirely on externally visible programs.
- Make assertions about the effects of traces.

The design of the "uses" relation

What is the "uses" relation?

- The decision about which programs a program may use

What are the goals of the "uses design?

- Phased development
- Fall back during development and delivery
- Fail soft product
- Minimise need for scaffolding
- Provide for reduced capability (cheaper) products

How do you document the uses relation?

- Boolean matrix
- Picture

The "uses" relation should be hierarchical
It will be modified during development.

Designing the implementation of individual modules

<u>What should be in the module design document?</u>

- Internal data structures (often created by submittals) may be implemented using other modules.
- Function/LD-relation of each access program.
- Abstraction Function: data → abstract value.

<u>What will the design document be used for?</u>

- This document should be available to guide the coders.
- This document is the basis for outside review.
- This document is a guide to future maintainers

Documentation:

Necessary evil or designer's medium

Most software designers view documentation as a necessary evil.

Those charged with maintenance agree that what they get is evil (or at least misguided).

Rush to Code: Software just seems to happen.

“Pre-Implementation Phases”--they produce a lot of blah blah blah.

Resolving the dilemma--Documentation that is both maintenance document and design medium

- Used as a medium for abstract design
- Used for design reviews
- Used as a manual in the design phase
- Used to bring new staff up to speed
- Used by the maintainer

The Rational Design Process

We will never follow it, but:

We can produce all the documents that we would have produced if we had.

Those are the documents that we want.

The "myth" of the rational designer came about because we wanted the reality of the rational design documents.

We are not misleading students, unless we pretend that we complete those documents in the order that they are described.