My Position: Tom Gilb (Tom@Gilb.com), Version 31Jan2010.

SOME BASIC CONCEPTS *
A **software engineer** is an engineer with specialty in software.
Software engineering is the discipline of making software systems deliver the required value to all stakeholders.

Engineering is an Evolutionary Process, using practical Principles, in order to determine, and identify the Means to deliver, the best achievable Performance and Cost levels balance, for optimal Stakeholder satisfaction, in a complex risk-filled environment.

Software refers to the 'non-hardware' aspects or components of a system.
A **Softcrafter** is a person who practices the craft of programming software for computers
***http://www.gilb.com/tiki-download_file.php?fileId=25**
Source: Planguage Glossary. For far more detail and far more concepts.

Assertion: As several have already pointed out: we need to agree on the very basic concepts of software + engineering. In particular we need to carefully distinguish between engineering, and the craft of programming. And several participants have already, in my view, totally failed to do so. The people who are going to discuss programming should join a different organization, "Improving Softcrafting" or "Better Coding". SEMAT is about ***software engineering***.

Engineering Core: The core of all engineering is deeply reasoned by Billy Koen in "DISCUSSION OF THE METHOD: Conducting the Engineer's Approach to Problem Solving", see paper at
http://www.cse.hcmut.edu.vn/~minhle/congtackysu_2008/Engineering_Method.pdf
I have paraphrased his definition in "**Engineering'** above.

SOFTWARE SPECIAL?
The **only distinction** between ***software*** engineering, and all *other* engineering disciplines, should be the soft engineering **artefacts** themselves, and corresponding knowledge of their probable characteristics.

CENTRAL GUIDING ORACLE (what's it all about?)
The central guiding principle, is that the software engineer works to deliver best and sufficient **value for resources**, within constraints.

WHAT'S USEFUL TO TEACH AND DO?
Consequently **any means** – processes, rules, standards, principles, tools, etc. – that **currently** contribute to that '**value for resources',** are *valid, interesting, and useful* software engineering components. Any provably *better* means, are better where proven better: even if they are not currently popular or consensus. Even if they have no scientific validation or long history yet. Naturally, promising methods should be validated scientifically in the long term.

Kernel:
Overview: http://www.gilb.com/tiki-download_file.php?fileId=98
Paper: "Undergraduate Basics"

Concepts:
1. independent of cross references to them such as words, symbols.
See: http://www.gilb.com/tiki-download_file.php?fileId=25
And CE book (a subset) for practical example, 655 Concepts.

Principles:
1. eternal, powerful, general, practical, wisdom
See: http://www.gilb.com/tiki-download_file.php?fileId=352
For the CE Planguage Collection.

Measures:
1. A proven method for capturing knowledge about a discipline
2. absolutely essential to 'engineering', 'management', science, reasoning
3. a language to describe almost any
discipline artifact: processes, rules, principles, tools, etc.
A key to objectivity and clarity.
See: http://www.gilb.com/tiki-download_file.php?fileId=26
Scales of Measure Chapter 5 of CE.

Processes
1. logical steps of actions to capture wisdom, and define work
2. a way to transfer wisdom

See:
http://homepage.mac.com/tomgilb/filechute/%20%20Gilb%20Competitive%20Engineering%20Book%20copy%201.pdf The CE book, **Processes** every chapter.

Rules:
1. necessary powerful practices of engineering specification
2. a way to transfer wisdom
See:
http://homepage.mac.com/tomgilb/filechute/%20%20Gilb%20Competitive%20Engineering%20Book%20copy%201.pdf The CE book, **Rules** every chapter

Representations (any and all useful representations are fine): But representations must include the ability to model all costs and qualities!
- Views
- Modelling
- Templates: see examples in CE book, URL above.
- Tools
- Icons: See Examples in Planguage Glossary, URL Above, and paper
- http://www.gilb.com/tiki-download_file.php?fileId=37&highlight=plicons
-

KEY CONCEPTS FROM THE 655 CONCEPT PLANGUAGE GLOSSARY

http://www.gilb.com/tiki-download_file.php?fileId=125

Software Concept *570 March 12, 2003
Software refers to the 'non-hardware' aspects or components of a system.

- It specifically includes computer programs, data (computer readable files and databases), and software documentation and plans (any form of specification or plans made by people concerning software).

Software Engineer: Concept *571. July 12, 2002
A software engineer is **an engineer with specialty in software**.

- They are characterized by the ability to assemble software components based on quantified attributes. This ability is aimed at the need to meet multiple quantified requirement performance levels, within specified resource constraints, and other constraint limitations.
- Consequently software engineers think in terms of measurable system performance (including quality) characteristics, and costs for design, implementation, decommissioning, adaptation, and operation. They know how to access the multiple quantified attributes of a design component and how to measure these attributes in the systems they engineer.

Software Engineering Concept *572 March 12, 2003
Software engineering is **the discipline of making software systems deliver the required value to all stakeholders.**

Software engineering includes determining stakeholder requirements, designing new systems, adapting older systems, subcontracting for components (including services), interfacing with systems architecture, testing, measurement, and other disciplines. It needs to control computer programming and other software related sub-processes (like quality assurance, requirements elicitation, requirement specification), but it is not necessary that, these sub-disciplines be carried out by the software engineering process, itself.

The emphasis should be on control of the outcome – the value delivered to stakeholders, not of the performance of a craft.

The concept 'required value' (above) is used to emphasize the obligation of the software engineer to determine the value or results truly needed by the stakeholders, and not to be fooled by omissions, corruptions and misunderstandings of the real world value.