

The Axioms of Subsumption Ethics¹

By David H. Gleason

Principal Consultant, ITforProgress.com

www.Linkedin.com/in/dgleason

Copyright © 1999

Rev. 11/12/2009

Abstract:

Subsumption ethics is the process by which decisions become incorporated into the operation of information technology (IT) systems, and subsequently forgotten. There are four axioms of subsumption ethics: A. Information systems subsume design, policy and implementation decisions in programming code and content; B. Subsumed objects have determinate moral value; C. Subsumed objects have a high “invisibility factor;” and D. Subsumptive complexity increases over time. These axioms can be applied to practical problems in IT by using them in conjunction with established ethical frameworks.

1. Introduction:

This paper first describes the principle of subsumption ethics, then discusses its four axioms in detail. It concludes with a discussion of the application of the axioms to IT development within the context of software development models, codes of ethics and impact analysis procedures.

Subsumption ethics is the process by which decisions become incorporated into the operation of information technology (IT) systems, and subsequently forgotten. IT systems, by nature, repeat operations over and over. If those operations have unethical impacts, the system will continue to execute them anyway. Unlike a human operator, there is no point in the cycle where the machine pauses to ask, “Should I do this?”

In computer systems, small components are developed and tested, and once they are working reliably they are subsumed into larger systems. This is the enabling technique of object oriented programming. The larger systems, in turn, are subsumed into still larger systems. Once components, subsystems and applications are operating, subsumed processes become invisible and *unavailable* to the user, what James Moor calls the “invisibility factor” [Moor 1995].

Systems seem like they should be extremely malleable. Because programming is just a set of instructions, and not like a building made up of hard materials, people tend to think that changes to software should be easy. However, the principle of subsumption makes it clear that changing base components is like moving building foundations, and can require changes to entire systems. The year 2000 problem, for example, is a result of subsumed date processing. There are thousands of layers of subsumption in a typical computer system.

There are four axioms of subsumption ethics:

- A. Information systems subsume design, policy and implementation decisions in programming code and content. Code segments and content become “subsumed objects”
-

¹ This paper was presented at ETHICOMP 1999 in Gdansk, Poland.

(SOs). This axiom posits that the decisions themselves, including many subtle factors, are incorporated into systems operation.

B. Subsumed objects have determinate moral value. The moral value of some SOs may be insignificant, while the moral value of others is important. Anecdotally, we can see the moral value of SOs. The data entry screen of the Therac 25 Radiotherapy machine caused great harm. By contrast, a subsumed object that provides life-saving information has a high moral value. In each case, moral value has to do with outcomes, the impact that a SO has.

C. Subsumed objects have a high “invisibility factor.” SOs are invisible to most users. It is not possible, for example, to know all the calculations that mortgage eligibility software might use without seeing the source code. Such software could systematically discriminate without a user’s knowledge.

D. Subsumptive complexity increases over time. As systems are developed, components become subsumed more and more deeply.

Since the axioms are not dependant on a specific ethical framework, they can be applied in many situations, across cultural as well as industrial boundaries. Subsumption ethics therefore offers developers a powerful tool to use to enhance project management and facilitate software impact analysis.

2. The Axioms of Subsumption Ethics

This section will describe each of the axioms in detail, and provide examples of their operation.

2.1. Axiom A: Information systems subsume design, policy and implementation decisions in programming code and content.

2.2. Design decisions

Software design requires thousands of decisions. From the layout of a title page to the format of fields on screen forms, software incorporates the decisions of designers.

For example, in one project, the developers were to build software to manage photography studios. Early on, the decision was made to use PowerBuilder as the programming platform. While the software eventually worked, it took up to a minute for some screens to refresh, and several minutes for the program to load. The project was a complete loss. The application was too complex for PowerBuilder, and should have been written in C. However, the design decision was subsumed, and by the time the developer knew it wouldn’t work, it was too late to change it.

The Macintosh and Windows operating systems have icons that are displayed in virtually every window on the screen. These window controls represent subsumed design decisions. The designers of commercial software assume that these controls will be ubiquitously available to programs. The controls are subsumed objects.

2.3. Policy decisions

Similarly, software codifies policy decisions. Once codified, these policies will be enforced by the system indefinitely.

A company recently programmed an order entry screen. According to policy, once a “Promised Delivery Date” was set, it could not be changed. They therefore wrote a subsumed object that would not allow the user to change the date after the record was saved.

Credit card approval systems are essentially policing systems. When a cardholder makes a purchase, they check, among other things, the card’s activation status, credit availability and recent activity. They have established routines to adjudicate problems.

The codification of policy is so pervasive that Microsoft sells a “Transaction Server” (MTS), designed specifically to store “business rules.” These rules determine the response of the system to activity. As an MTS application is built, the business rules (policies) become subsumed objects.

2.4. Implementation decisions

As above, implementation decisions become subsumed objects. Once implementation is underway, change is difficult.

For example, data conversion from one system to another is often very complex. New systems seldom have identical data structures to their predecessors. Data conversion becomes a subtle process of parsing and reorganizing information. It is fraught with the possibility of error.

In one case, a set of data files were converted from Symantec’s *Act* to Commence Corp’s *Commence*. The systems have entirely different data structures — *Act* is essentially a flat file database, and *Commence* is relational. In the process of the conversion, a number of individuals were inadvertently linked to incorrect companies. A good deal of additional work was done on the system before the problem was discovered, and management decided not to go back and redo the conversion. Rather, they thought they would go back and clean up the data manually, a cleanup that was never done. The implementation decision not to redo the conversion has been subsumed, and these incorrect links will remain in the system for a long time.

Effective implementation decisions are similarly subsumed. An unreliable file server can be devastating to a company, causing lost business hours and negatively impacting customer and employee confidence. It is usually in a company’s best interest to implement a reliable, brand name and relatively expensive file server. The company can then rely on this stable subsumed object.

2.5. Programming code

Programming code is invisible to the user, but contains the actual systems processes. Every line of code subsumes decisions.

2.6. Content

Unlike code, content is the user’s stock and trade. Nevertheless, it is often difficult to change. In the data conversion example above, the content is incorrect, but has been subsumed. The industry term GIGO — Garbage In, Garbage Out — refers to the widespread problem of inaccurate data.

2.7. Axiom B: Subsumed objects have determinate moral value.

The moral value of any object or action is related to its impact. When an action has a negative impact, such as an accident caused by a drunk driver, we call the drinker's morals into question. Driving while drunk is an immoral act, because the potential consequences are deadly.

The moral value of the car in this example may be described as neutral. Morality becomes the operator's responsibility.

However, there are cases where subsumed objects within cars have been described as having low moral value. The Ford Pinto of the 1970s had a gas tank that could explode in rear-end collisions. Similarly, Fiat was forced to recall some of its cars in the early 1980s because they rusted so badly that the suspension systems were at risk of collapse during high-speed travel.

Furthermore, in computer systems, the distinction between object and action is blurred because *software takes action*. Most contemporary fuel injection systems are under software control. Autopilot systems, intrusion alert systems and microwave ovens are all run by computers.

The actions of these systems have been programmed ahead of time. These actions may have negative moral consequences, e.g. allowing a microwave to operate with the door open. It is therefore up to the system designers and developers to think through the impact of their programming, because by the time the software is operating, its moral value has already been determined. The computer does not pause to reflect on the impact of what it does... it simply follows instructions.

It should be said that software in general is remarkably effective. Word processors and spreadsheets save time and empower people. There are thousands of subsumed objects at work in Microsoft Word. The vast majority of them function perfectly, and interact well with the user. The value of these subsumed objects is high, particularly in their relationship to the operation of the complete product.

Subsumed objects are likely to have positive moral values if they meet ethical criteria, for example, if they conform to the Software Engineering Code of Ethics and Professional Practice, which embodies such principles as acting "consistently with the public interest" and "meeting the highest professional standards possible" [SECODE 1999].

Subroutines that directly fail are simple examples of subsumed objects with low determinate moral value — they cause lost time, missed deadlines, user frustration and worse. Most software contains known bugs. These are often fixed with "patches" — programs that change subsumed objects within the software. This improves the moral (and operational) value of the objects.

2.8. Axiom C: Subsumed objects have a high "invisibility factor."

Subsumed objects are invisible to most users. Code is hidden. The operation of the code results in a user interface that is a by-product of the code's operation.

Computers still run on "machine language" — a series of 0's and 1's that fly through the processor. The actual program that a computer runs looks like this: "01000101011001011101011101100001101101...." If Microsoft Office takes 120 Megabytes of storage, then, all told, the program and its supporting files is a string of 960 million 0's and 1's.

(Actually, the 0's and 1's are themselves an abstraction, representing the off-on state of electronic gates.)

We use higher level languages such as C or Visual Basic to write the source code that gets translated into machine language. But once the software has been compiled, it is virtually impossible to go back and understand it. Picking the subsumed objects out of machine language is like finding a thousand needles in a million haystacks. It is not possible.

If source code is available, programmers can usually go back and, with effort, make sense of the software's processes, particularly if the code has embedded documentation. However, they still don't have access to the *intent* or design of the software. Those decisions are subsumed and largely unavailable.

Even if source code is available, most users cannot read it. Nor would they want to. Most "knowledge workers" use software to do their work. They want their document to print when they press the print button. They want their e-mail to be sent, or their on-line transaction to go through. They don't really care much how it works. Nor will they ever know.

2.9. Axiom D: Subsumptive complexity increases over time.

As systems are developed, components become subsumed more and more deeply. There are still components of MS-DOS in Windows 98. Windows 2000 has something like 50 million lines of code in it. No one knows what all those lines do. Some of them were written ten years ago. They have been subsumed into an extremely complex system.

One small company that has grown from 14 users in October of 1998 to over 40 users in July of 1999 has subsumed thousands of hours of design and development effort. It now has three file servers, two routers, six ethernet hubs and switches, Microsoft Office, database access, Internet access, e-mail and access to a remote system and cable running everywhere.

During this period they have also developed a database system through which the company does all its business. One of their new service delivery policies is "We write nothing on paper" — a decision that will be deeply subsumed by the system. At this point, there are thousands of layers of complexity in this system. Changing subsumed objects, such as a file server or data structure now takes weeks, because the system has grown complex over time.

The company expects to grow to hundreds of users over many cities. The decisions that are made now will be subsumed into the systems that others will have to work with later. Developers, therefore, have an obligation to help their clients create subsumed objects that meet a number of ethical standards.

3. Project Management, Codes of Ethics and Impact Analysis

Inattention to subsumption inevitably leads to projects from hell. This is because bad decisions become subsumed, and are extremely difficult to undo, or even identify, later on. Tools are needed at the level of subsumed objects to ensure that they are developed to operate ethically within systems. Some of these tools already exist.

3.1. Project Management

At a basic level, software development should be managed. Lack of management leads to failed, unsatisfactory, late and over-budget projects. Because of these negative consequences, it is unethical to run software development projects with managing them properly. There are many methodologies available for software development.

3.2. The Capability Maturity Model

Once basic project management procedures are established, developers should continuously improve their methodologies, knowledge and practices. The Software Engineering Institute's *The Capability Maturity Model* (CMM) describes how companies can build increasing quality into their development methodologies. It describes 5 levels of software development skill in its "Capability Maturity Model." The levels are 1. Initial (Ad Hoc); 2. Repeatable (Stable, controlled, project management); 3. Defined (Codified processes rigorously followed); 4. Managed (Process measurement, quality targets); 5. Optimized (Continuous process improvement)

3.3. Codes of ethics

In order to add proactive ethical content to the process, a code of ethics is required. The ACM and IEEE have recently adopted a code of ethics to guide developers. *Software Engineering Code of Ethics and Professional Practice* provides a framework within which to plan and develop subsumed objects. This code of ethics can be applied to practice as subsumed objects are created.

3.4. Software Development Impact Statement

Simon Rogerson and Don Gotterbarn are building a tool, called the "Software Development Impact Statement" (SoDIS). SoDIS is a process that applies ethical criteria to software project management plans and work breakdown structures (WBS) to identify significant issues before software design is underway.

As a development project is planned, it is broken down into tasks that are manageable and predictable, known generally as a WBS. This minimizes risk and enables scheduling and estimating. Each of these tasks has a number of data elements associated with it, such as due dates, human resources required and dependencies on other tasks.

SoDIS adds an impact analysis process to project planning. It first helps the user identify the stakeholders — people and organizations that will be affected by the software — and asks a series of questions for each of the tasks, such as, "will this task cause harm to any stakeholder" and "will execution of this task compromise data integrity?" By running this analysis, developers can anticipate unwanted impacts early in the project, before objects become subsumed and difficult to change. In particular, SoDIS seeks early adjudication of issues that would require rework of the product after delivery.

There are at least three methodologies in place that can serve as ethical guides to the development of subsumed objects: The Capability Maturity Model, the IEEE/ACM code of ethics and the SoDIS tool. The concept and axioms of subsumption ethics can be used to focus such methodologies on subsumed objects, the building blocks of systems. The value of these considerations is subsumed objects that are more likely to behave ethically.

4. Conclusion

Information systems are growing at astronomical rates. Subsumption ethics is one tool that can help ensure desirable results. The axioms of subsumption ethics point to the care that is needed in the early design of even small systems components.

5. Acknowledgments

My lifelong thanks to Sonia and Christopher for their support. Thanks also to Don Gotterbarn, Simon Rogerson, Jim Moor, Terry Bynum, Lucas Introna and Deborah Johnson for their advice and feedback.

6. References

Gibbs, W. (1994), "Software's Chronic Crisis," Scientific American September, 87.

Kelly, K. (1994), Out of Control, Addison-Wesley, 39-41.

Moor, J. H. (1995) "What Is Computer Ethics" Johnson & Nissenbaum, Computers, Ethics and Social Values, Prentice Hall, 13.

SECODE (1999), IEEE-CS/ACM, www-cs.etsu.edu/seeri/secode.htm, version 5.2, Accessed 1/4/99.