is more keen to find suitable esthetical approaches for each category. In Borgmannian terms, demoscene practices are more focal.

Demoscene-inspired practices may not be the wisest choice for pragmatic software development. However, they can be recommended for the development of a deeper relationship with technology and for diminishing the alienating effects of our growth-obsessed civilization.

What to do?

I am convinced that our civilization is already falling and this fall cannot be prevented. What we can do, however, is create seeds for something better. Now is the best time for doing this, as we still have plenty of spare time and resources especially in rich countries. We especially need to propagate the seeds towards laypeople who are already suffering from increasing alienation because of the ever more computerized technological culture. The masses must realize that alternatives are possible.

A lot of our current civilization is constructed around the resource leak bug. We must therefore deconstruct the civilization down to its elementary philosophies and develop new alternatives. Countercultural insights may be useful here. And since hacker subcultures have been forced to deal with the resource leak bug in its most extreme manifestation for some time already, their input can be particularly valuable.

http://viznut.fi/texts-en/resource_leak_bug_of_our_civilization.html

The resource leak bug of our civilization

Viznut / Ville-Matias Heikkilä
5 aug 2014

A couple of months ago, Trixter of Hornet released a demo called "8088 Domination", which shows off real-time video and audio playback on the original 1981 IBM PC. This demo, among many others, contrasts favorably against today's wasteful use of computing resources.

When people try to explain the wastefulness of today's computing, they commonly offer something I call "tradeoff hypothesis". According to this hypothesis, the wastefulness of software would be compensated by flexibility, reliability, maintainability, and perhaps most importantly, cheap programming work. Even Trixter himself favors this explanation.

I used to believe in the tradeoff hypothesis as well. I saw demo art on extreme platforms as a careful craft that attains incredible feats while sacrificing generality and development speed. However, during recent years, I have become increasingly convinced that the portion of true tradeoff is quite marginal. An ever-increasing portion of the waste comes from abstraction clutter that serves no purpose in final runtime code. Most of this clutter could be eliminated with more thoughtful tools and methods without any sacrifices. What we have been witnessing in computing world is nothing utilitarian but a reflection of a more general, inherent wastefulness, that stems from the internal issues of contemporary human civilization.

is more keen to find suitable esthetical approaches for each category. In Borgmannian terms, demoscene practices are more focal.

Demoscene-inspired practices may not be the wisest choice for pragmatic software development. However, they can be recommended for the development of a deeper relationship with technology and for diminishing the alienating effects of our growth-obsessed civilization.

What to do?

I am convinced that our civilization is already falling and this fall cannot be prevented. What we can do, however, is create seeds for something better. Now is the best time for doing this, as we still have plenty of spare time and resources especially in rich countries. We especially need to propagate the seeds towards laypeople who are already suffering from increasing alienation because of the ever more computerized technological culture. The masses must realize that alternatives are possible.

A lot of our current civilization is constructed around the resource leak bug. We must therefore deconstruct the civilization down to its elementary philosophies and develop new alternatives. Countercultural insights may be useful here. And since hacker subcultures have been forced to deal with the resource leak bug in its most extreme manifestation for some time already, their input can be particularly valuable.

http://viznut.fi/texts-en/resource_leak_bug_of_our_civilization.html

8

The resource leak bug of our civilization

Viznut / Ville-Matias Heikkilä
5 aug 2014

A couple of months ago, Trixter of Hornet released a demo called "8088 Domination", which shows off real-time video and audio playback on the original 1981 IBM PC. This demo, among many others, contrasts favorably against today's wasteful use of computing resources.

When people try to explain the wastefulness of today's computing, they commonly offer something I call "tradeoff hypothesis". According to this hypothesis, the wastefulness of software would be compensated by flexibility, reliability, maintainability, and perhaps most importantly, cheap programming work. Even Trixter himself favors this explanation.

I used to believe in the tradeoff hypothesis as well. I saw demo art on extreme platforms as a careful craft that attains incredible feats while sacrificing generality and development speed. However, during recent years, I have become increasingly convinced that the portion of true tradeoff is quite marginal. An ever-increasing portion of the waste comes from abstraction clutter that serves no purpose in final runtime code. Most of this clutter could be eliminated with more thoughtful tools and methods without any sacrifices. What we have been witnessing in computing world is nothing utilitarian but a reflection of a more general, inherent wastefulness, that stems from the internal issues of contemporary human civilization.

1

The bug

Our mainstream economic system is oriented towards maximal production and growth. This effectively means that participants are forced to maximize their portions of the cake in order to stay in the game. It is therefore necessary to insert useless and even harmful "tumor material" in one's own economical portion in order to avoid losing one's position. This produces an ever-growing global parasite fungus that manifests as things like black boxes, planned obsolescence and artificial creation of needs.

Using a software development metaphor, it can be said that our economic system has a fatal bug. A bug that continuously spawns new processes that allocate more and more resources without releasing them afterwards, eventually stopping the whole system from functioning. Of course, "bug" is a somewhat normative term, and many bugs can actually be reappropriated as useful features. However, resource leak bugs are very seldom useful for anything else than attacking the system from the outside.

Bugs are often regarded as necessary features by end-users who are not familiar with alternatives that lack the bug. This also applies to our society. Even if we realize the existence of the bug, we may regard it as a necessary evil because we don't know about anything else. Serious politicians rarely talk about trying to fix the bug. On the contrary, it is actually getting more common to embrace it instead. A group that calls itself "Libertarians" even builds their ethics on it. Another group called "Extropians" takes the maximization idea to the extreme by advocating an explosive expansion of humankind into outer space. In the so-called Kardashev scale, the developmental stage

countercultural mirror that contrasts against the trends of industrial software development and helps grasp its inherent problems.

Other subcultures have been far less useful for me in this endeavour. The mainstream of open source / free software, for example, is a copycat culture, despite its strong ideological dimension. It does not actively question the philosophies and methodologies of the growth-obsessed industry but actually embraces them when creating duplicate implementations of growth-obsessed software ideas.

Perhaps the strongest countercultural trend within the demoscene is the move of focus towards ever tighter size limitations, or as they say, "4k is the new 64k". This trend is diagonally opposite to what the growth-oriented society is doing, and forces to rethink even the deepest "best practices" of industrial software development. Encapsulation, for example, is still quite prominent in the 4k category (4klang is a monolith), but in 1k and smaller categories, finer methods are needed. When going downwards in size, paths considered dirty by the mainstream need to be embraced. Efficient exploration and taming of chaotic systems needs tools that are deeply different from what have been used before. Stephen Wolfram's ideas presented in "A New Kind of Science" can perhaps provide useful insight for this endeavour.

Another important countercultural aspect of the demoscene is the relationship with computing platforms. The mainstream regards platforms as neutral devices that can be used to reach a predefined result, while the demoscene regards them as a kind of raw material that has a specific essence of its own. Size categories may also split platforms into subplatforms, each of which has its own essence. The mainstream wants to hide platform-specific characteristics by encapsulating them into uniform straightjackets, while the demoscene

The bug

Our mainstream economic system is oriented towards maximal production and growth. This effectively means that participants are forced to maximize their portions of the cake in order to stay in the game. It is therefore necessary to insert useless and even harmful "tumor material" in one's own economical portion in order to avoid losing one's position. This produces an ever-growing global parasite fungus that manifests as things like black boxes, planned obsolescence and artificial creation of needs.

Using a software development metaphor, it can be said that our economic system has a fatal bug. A bug that continuously spawns new processes that allocate more and more resources without releasing them afterwards, eventually stopping the whole system from functioning. Of course, "bug" is a somewhat normative term, and many bugs can actually be reappropriated as useful features. However, resource leak bugs are very seldom useful for anything else than attacking the system from the outside.

Bugs are often regarded as necessary features by end-users who are not familiar with alternatives that lack the bug. This also applies to our society. Even if we realize the existence of the bug, we may regard it as a necessary evil because we don't know about anything else. Serious politicians rarely talk about trying to fix the bug. On the contrary, it is actually getting more common to embrace it instead. A group that calls itself "Libertarians" even builds their ethics on it. Another group called "Extropians" takes the maximization idea to the extreme by advocating an explosive expansion of humankind into outer space. In the so-called Kardashev scale, the developmental stage

2

countercultural mirror that contrasts against the trends of industrial software development and helps grasp its inherent problems.

Other subcultures have been far less useful for me in this endeavour. The mainstream of open source / free software, for example, is a copycat culture, despite its strong ideological dimension. It does not actively question the philosophies and methodologies of the growth-obsessed industry but actually embraces them when creating duplicate implementations of growth-obsessed software ideas.

Perhaps the strongest countercultural trend within the demoscene is the move of focus towards ever tighter size limitations, or as they say, "4k is the new 64k". This trend is diagonally opposite to what the growth-oriented society is doing, and forces to rethink even the deepest "best practices" of industrial software development. Encapsulation, for example, is still quite prominent in the 4k category (4klang is a monolith), but in 1k and smaller categories, finer methods are needed. When going downwards in size, paths considered dirty by the mainstream need to be embraced. Efficient exploration and taming of chaotic systems needs tools that are deeply different from what have been used before. Stephen Wolfram's ideas presented in "A New Kind of Science" can perhaps provide useful insight for this endeavour.

Another important countercultural aspect of the demoscene is the relationship with computing platforms. The mainstream regards platforms as neutral devices that can be used to reach a predefined result, while the demoscene regards them as a kind of raw material that has a specific essence of its own. Size categories may also split platforms into subplatforms, each of which has its own essence. The mainstream wants to hide platform-specific characteristics by encapsulating them into uniform straightjackets, while the demoscene

7

current for very long. If you modify it, subsequent software updates will break it. It is extremely difficult to develop a focal relationship with a modern technological system. Even hard-core technology enthusiasts tend to ignore most aspects of the systems they are interested in. When ever-complexifying computer systems grow ever deeper ingrained into our society, it becomes increasingly difficult to grasp even for those who are dedicated to understand it. Eventually even they will give up.

Chopping one's own wood may be a useful way to counteract the alienation of the classic industrial society, as oldschool factories and heating stoves still have some basics in common. In order to counteract the alienation caused by computer technology, however, we need to find new kind of focal things and practices that are more computerish. If they cannot be found, they need to be created. Crafting with low-complexity computer and electronic systems, including the creation of art based on them is my strongest candidate for such a focal practice among those practices that already exist in subcultural form.

The demoscene insight

I have been programming since my childhood, for nearly thirty years. I have been involved with the demoscene for nearly twenty years. During this time, I have grown a lot of angst towards various trends of computing.

Extreme categories of the demoscene -- namely, eight-bit democoding and extremely short programs -- have been helpful for me in managing this angst. These branches of the demoscene are a useful,

of a civilization is straightforwardly equated with how much stellar energy it can harness for production-for-its-own-sake.

How the bug manifests in computing

What happens if you give this buggy civilization a virtual world where the abundance of resources grows exponentially, as in Moore's law? Exactly: it adopts the extropian attitude, aggressively harnessing as much resources as it can. Since the computing world is virtually limitless, it can serve as an interesting laboratory example where the growth-for-its-own-sake ideology takes a rather pure and extreme form. Nearly every methodology, language and tool used in the virtual world focuses on cumulative growth while neglecting many other aspects.

To concretize, consider web applications. There is a plethora of different browser versions and hardware configurations. It is difficult for developers to take all the diversity in account, so the problem has been solved by encapsulation: monolithic libraries (such as Jquery) that provide cross-browser-compatible utility blocks for client-side scripting. Also, many websites share similar basic functionality, so it would be a waste of labor time to implement everything specifically for each application. This problem has also been solved with encapsulation: huge frameworks and engines that can be customized for specific needs. These masses of code have usually been built upon previous masses of code (such as PHP) that have been designed for the exactly same purpose. Frameworks encapsulate legacy frameworks, and eventually, most of the computing resources are wasted by the intermediate bloat. Accumulation of unnecessary code dependencies

current for very long. If you modify it, subsequent software updates will break it. It is extremely difficult to develop a focal relationship with a modern technological system. Even hard-core technology enthusiasts tend to ignore most aspects of the systems they are interested in. When ever-complexifying computer systems grow ever deeper ingrained into our society, it becomes increasingly difficult to grasp even for those who are dedicated to understand it. Eventually even they will give up.

Chopping one's own wood may be a useful way to counteract the alienation of the classic industrial society, as oldschool factories and heating stoves still have some basics in common. In order to counteract the alienation caused by computer technology, however, we need to find new kind of focal things and practices that are more computerish. If they cannot be found, they need to be created. Crafting with low-complexity computer and electronic systems, including the creation of art based on them is my strongest candidate for such a focal practice among those practices that already exist in subcultural form.

The demoscene insight

I have been programming since my childhood, for nearly thirty years. I have been involved with the demoscene for nearly twenty years. During this time, I have grown a lot of angst towards various trends of computing.

Extreme categories of the demoscene -- namely, eight-bit democoding and extremely short programs -- have been helpful for me in managing this angst. These branches of the demoscene are a useful,

6

of a civilization is straightforwardly equated with how much stellar energy it can harness for production-for-its-own-sake.

How the bug manifests in computing

What happens if you give this buggy civilization a virtual world where the abundance of resources grows exponentially, as in Moore's law? Exactly: it adopts the extropian attitude, aggressively harnessing as much resources as it can. Since the computing world is virtually limitless, it can serve as an interesting laboratory example where the growth-for-its-own-sake ideology takes a rather pure and extreme form. Nearly every methodology, language and tool used in the virtual world focuses on cumulative growth while neglecting many other aspects.

To concretize, consider web applications. There is a plethora of different browser versions and hardware configurations. It is difficult for developers to take all the diversity in account, so the problem has been solved by encapsulation: monolithic libraries (such as Jquery) that provide cross-browser-compatible utility blocks for client-side scripting. Also, many websites share similar basic functionality, so it would be a waste of labor time to implement everything specifically for each application. This problem has also been solved with encapsulation: huge frameworks and engines that can be customized for specific needs. These masses of code have usually been built upon previous masses of code (such as PHP) that have been designed for the exactly same purpose. Frameworks encapsulate legacy frameworks, and eventually, most of the computing resources are wasted by the intermediate bloat. Accumulation of unnecessary code dependencies

3

also makes software more bug-prone, and debugging becomes increasingly difficult because of the ever-growing pile of potentially buggy intermediate layers.

Software developers tend to use encapsulation as the default strategy for just about everything. It may feel like a simple, pragmatic and universal choice, but this feeling is mainly due to the tools and the philosophies they stem from. The tools make it simple to encapsulate and accumulate, and the industrial processes of software engineering emphasize these ideas. Alternatives remain underdeveloped. Mainstream tools make it far more cumbersome to do things like metacoding, static analysis and automatic code transformations, which would be far more relevant than static frameworks for problems such as cross-browser compatibility.

Tell a bunch of average software developers to design a sailship. They will do a web search for available modules. They will pick a wind power module and an electric engine module, which will be attached to some kind of a floating module. When someone mentions aero- or hydrodynamics, the group will respond by saying that elementary physics is a far too specialized area, and it is cheaper and more straightforward to just combine pre-existing modules and pray that the combination will work sufficiently well.

Result: alienation

The way of building complex systems from more-or-less black boxes is also the way how our industrial society is constructed. Computing just takes it more extreme. Modularity in computing therefore relates very

well to the technology criticism of philosophers such as Albert Borgmann.

In his 1984 book, Borgmann uses the term "service interface", which even sounds like software development terminology. Service interfaces often involve money. People who have a paid job, for example, can be regarded as modules that try to fulfill a set of requirements in order to remain acceptable pieces of the system. When using the money, they can be regarded as modules that consume services produced by other modules. What happens beyond the interface is considered irrelevant, and this irrelevance is a major source of alienation. Compare someone who grows and chops their own wood for heating to someone who works in forest industry and buys burnwood with the paycheck. In the former case, it is easier to get genuinely interested by all the aspects of forests and wood because they directly affect one's life. In the latter case, fulfilling the unit requirements is enough.

The way of perceiving the world as modules or devices operated via service interfaces is called "device paradigm" in Borgmann's work. This is contrasted against "focal things and practices" which tend to have a wider, non-encapsulated significance to one's life. Heating one's house with self-chopped wood is focal. Also arts and crafts have a lot of examples of focality. Borgmann urges a restoration of focal things and practices in order to counteract the alienating effects of the device paradigm.

It is increasingly difficult for computer users to avoid technological alienation. Systems become increasingly complex and genuine interest towards their inner workings may be discouraging. If you learn something from it, the information probably won't stay

also makes software more bug-prone, and debugging becomes increasingly difficult because of the ever-growing pile of potentially buggy intermediate layers.

Software developers tend to use encapsulation as the default strategy for just about everything. It may feel like a simple, pragmatic and universal choice, but this feeling is mainly due to the tools and the philosophies they stem from. The tools make it simple to encapsulate and accumulate, and the industrial processes of software engineering emphasize these ideas. Alternatives remain underdeveloped. Mainstream tools make it far more cumbersome to do things like metacoding, static analysis and automatic code transformations, which would be far more relevant than static frameworks for problems such as cross-browser compatibility.

Tell a bunch of average software developers to design a sailship. They will do a web search for available modules. They will pick a wind power module and an electric engine module, which will be attached to some kind of a floating module. When someone mentions aero- or hydrodynamics, the group will respond by saying that elementary physics is a far too specialized area, and it is cheaper and more straightforward to just combine pre-existing modules and pray that the combination will work sufficiently well.

Result: alienation

The way of building complex systems from more-or-less black boxes is also the way how our industrial society is constructed. Computing just takes it more extreme. Modularity in computing therefore relates very

4

well to the technology criticism of philosophers such as Albert Borgmann.

In his 1984 book, Borgmann uses the term "service interface", which even sounds like software development terminology. Service interfaces often involve money. People who have a paid job, for example, can be regarded as modules that try to fulfill a set of requirements in order to remain acceptable pieces of the system. When using the money, they can be regarded as modules that consume services produced by other modules. What happens beyond the interface is considered irrelevant, and this irrelevance is a major source of alienation. Compare someone who grows and chops their own wood for heating to someone who works in forest industry and buys burnwood with the paycheck. In the former case, it is easier to get genuinely interested by all the aspects of forests and wood because they directly affect one's life. In the latter case, fulfilling the unit requirements is enough.

The way of perceiving the world as modules or devices operated via service interfaces is called "device paradigm" in Borgmann's work. This is contrasted against "focal things and practices" which tend to have a wider, non-encapsulated significance to one's life. Heating one's house with self-chopped wood is focal. Also arts and crafts have a lot of examples of focality. Borgmann urges a restoration of focal things and practices in order to counteract the alienating effects of the device paradigm.

It is increasingly difficult for computer users to avoid technological alienation. Systems become increasingly complex and genuine interest towards their inner workings may be discouraging. If you learn something from it, the information probably won't stay

5