Oprogramowanie otwarte. Dlaczego czasem nam nie idzie?

Tomasz Barbaszewski
Kraków

1. Wstęp

Życie to nie jest bajka! Ten popularny zwrot zwłaszcza wszystkich, którzy pragną działać aktywnie, a jedynie tacy ludzie tworzą ruch Open Source.

Największym błędem, jaki można popełnić jest niedostrzeganie własnych błędów!

Oczywiście, można szukać usprawiedliwień przed samym sobą, zwalać wszystko na pogodę, rząd, listę IPN, Microsoft oraz fizykę jądrową. Jeśli sami w to uwierzymy – to wypada się tylko położyć pod gruszą i wypić kolejne piwko (jeżeli tylko będzie za co...)

Tylko potem nie należy mieć pretensji do całego Świata, że nas nie docenił.

2. Usługi czy produkty?

Powyższy dylemat staje przed każdym, kto pragnie prowadzić działalność komercyjną z wykorzystaniem produktów OpenSource. W większości przypadków kapitałem, którym dysponujemy jest po prostu nasza wiedza – a więc wydaje się, że świadczenie kompetentnych usług dla Klientów Końcowych jest najlepszym rozwiązaniem. Jeżeli w wyszukiwarce Google wpiszemy hasła `Usługi`, `Informatyczne` oraz `LINUX`, to nawet tylko w „Polskim Internecie" otrzymamy ogromną liczbę trafień, z których większa część to strony firm komercyjnych różnej wielkości.

Jednak w przytłaczającej większości przypadków (oczywiście, są i wyjątki) oferta takich usług występuje jako uzupełnienie innej działalności (np. typowo reselerskiej), ponieważ w typowym przypadku nie pozwala na utrzymanie nawet stosunkowo niewielkiej firmy!

Cena za instalację systemu LINUX na serwerze lub stacji roboczej kształtuje się w okolicach 300–400zł netto. Jeżeli dodamy do tego obsługę WWW, DHCP, FTP i SMB to klient zapłaci nam (według cen dostępnych w Internecie) mniej więcej 1000 – 1500zł netto. Osobom prywatnym może się to wydawać bardzo atrakcyjnym sposobem zarabiania pieniędzy, jednakże jeśli zestawimy to z potrzebami finansowymi firmy, w której pracuje 5 osób kalkulacja wygląda zgoła inaczej. Jeśli taka firma posiada niewielki lokal, pokrywa koszty rozmów telefonicznych oraz dostępu do Internetu, dysponuje niezbędnym sprzętem, prowadzi niezbędną obsługę księgową – to jej miesięczne koszty stałe będą kształtować się na poziomie około 5000zł (bez wynagrodzeń pracowników). Do tego dojdą koszty osobowe – przy założeniu, że pensja jednego pracownika (z podatkami, ZUS, opieką zdrowotną itp.) będzie kosztować firmę w przybliżeniu 2000zł (pracownik otrzyma wówczas około 1250zł „na rękę") koszty te wyniosą mniej więcej 10 tys. zł miesięcznie.

Tak więc razem musimy zarobić miesięcznie ponad 15 tys. zł, czyli wykonać co najmniej 15 typowych instalacji Linuksa! Oczywiście w przypadku droższych usług specjalizowanych bilans będzie korzystniejszy, ale zlecenia na takie usługi jest o wiele trudniej zdobywać.

Zarobek w wysokości 15 tys. zł miesięcznie, jest bardzo atrakcyjny dla 1–2 osób, ale z ledwością pozwala utrzymać niewielką firmę. Jeżeli chcemy, aby nasze instalacje były wykonywane rzeczywiście profesjonalnie i spełniały indywidualne wymagania użytkownika, to czas, który przeznaczymy na jedną taką instalację wyniesie mniej więcej dwa-trzy dni. Bardzo łatwo osiągniemy więc próg wydajności i albo zacznie nam brakować czasu, albo zaczniemy się rozglądać za powiększeniem zespołu. W jednym i drugim przypadku możemy łatwo wpaść w pułapkę (mogę podać wiele przykładów), ponieważ liczba kontraktów nie rośnie zazwyczaj liniowo z liczbą pracowników, a doba ma tylko 24 godziny. Sytuację mogą ratować długoterminowe kontrakty wsparcia technicznego, ale ich realizacja wymaga także dysponowania odpowiednimi zasobami.

3. To może produkty?

Niestety i w tym przypadku nie jest zbyt różowo. Opracowanie nowego produktu wymaga czasu! Ten czas na ogół nie zostanie nam podarowany. Nawet jeśli już opracujemy nowy, rewelacyjny produkt to pozostaje jeszcze „drobnostka" - wprowadzenie go na rynek. Jeżeli nawet nam się to uda, to i tak nie możemy spać spokojnie. Im większy sukces osiągniemy, tym szybciej „obudzi się" konkurencja. Oczywiście jej zadanie jest znacznie ułatwione – wystarczy wykorzystać nasz pomysł, zatrudnić odpowiednich ludzi (nawet na zlecenie) i zaoferować niższą cenę. Możemy więc znaleźć się dość łatwo w sytuacji, w której nie zdążymy odrobić naszych nakładów początkowych i zostaniemy zmuszeni do obniżania oferowanych cen poniżej poziomu opłacalności.

Produkty mają jednak jedną niezaprzeczalną zaletę – są powtarzalne i zazwyczaj chętniej kupowane (będzie o tym więcej później). Powszechna opinia głosi, że „Linux jest darmowy", co powoduje dużą niechęć do płacenia za usługi związane z tym systemem. Trudno się też „podpierać" argumentami typu „ale moja firma to Platynowy Reseller, co zapewnia...", bo po prostu czegoś takiego nie ma (choć dostawcy komercyjnych dystrybucji Linuksa usiłują coś podobnego stworzyć).

Co innego pudełko sprzętowe zawierające specjalizowane oprogramowanie. Opory związane z jego zakupem są znacznie mniejsze – wszak kupujemy coś materialnego, na czym można choćby nakleić numer inwentarzowy, a nie coś, o czym wszyscy wiedzą, że jest dostępne za darmo. Cena za naszą usługę (opracowanie odpowiedniej wersji systemu, świadczenie gwarancji itp.) nie jest więc zbyt widoczna i tym samym staje się bardziej akceptowalna.

4. Jeśli nie chcesz mojej zguby, inwestora daj mi luby...

Zdobycie inwestora wydaje się być „złapaniem byka za rogi". Niestety, obecnie działalność usługowa nie pozwala w większości przypadków na osiągnięcie oczekiwanych zysków (patrz wyżej), zaś produkcja wymaga inwestycji długoterminowej, a do takich inwestycji większość potencjalnych inwestorów nie jest przygotowana (niekoniecznie finansowo, ale mentalnie). Nawet jeżeli początkowo spotkamy się z entuzjazmem, to po kilku miesiącach zaczną się pojawiać problemy we współpracy, ponieważ znakomita większość przedsięwzięć (nawet tak zwanych „producentów") polega na zasadzie „tanio kupić – jak najszybciej sprzedać" i trudno znaleźć zrozumienie dla inwestycji, które zwracają się po roku lub dwóch.

Nie jest to jedynie polska specyfika (choć w Polsce taka tendencja jest szczególnie silna). Znakomitym przykładem może być choćby historia Bernarda Dainesa (znanego także jako „*Father of Fast Ethernet*" oraz „*King of Gigabit Ethernet*") i sukces jego technologii w CISCO Systems, a fiasko w Alcatelu...

A więc - „nie kijem go – to pałką".

Trudno tu coś konkretnego radzić – pomóc może jedynie upór, odporność na ciosy oraz przekonanie o słuszności obranej drogi – ostatnio z „Discovery Channel" dowiedziałem się, że Alfred Nobel wysadził w powietrze po kolei 6 swoich fabryk – a jednak coś po nim zostało....

5. Licencje otwarte i związane z nimi problemy

Tekst ten piszę w lutym (na prośbę Organizatorów), nie wiem, czy do Pingwinariów ukaże się już książka Prof. Ryszarda Markiewicza z Uniwersytetu Jagiellońskiego, która jest poświęcona właśnie tym zagadnieniom (Prof. R. Markiewicz jest jednym z Autorów naszej Ustawy o Ochronie Praw Autorskich). Na pewno warto będzie się zapoznać z tym opracowaniem.

Obecnie namnożyło nam się różnych „Licencji Otwartych", i programiści często się gubią w gąszczu stosowanych w nich sformułowań. O ile jeszcze licencję GPL większość z nas zna i popiera, to inne licencje, które są niekompatybilne z GPL rodzą wiele istotnych wątpliwości. Na przykład licencja zwana MPL (*Mozilla Open License*). Pozwala ona na wykorzystywanie bez ograniczeń kodu binarnego, ale tylko w postaci skompilowanej, dostarczanej przez Mozilla.org. Jeśli chcemy wykonać kompilację kodu we własnym zakresie (nawet bez jakichkolwiek zmian w kodzie źródłowym!), to wówczas powinniśmy spełnić kilka dodatkowych warunków (`www.mozilla.org/MPL/mpl-faq.html`)

Dodatkowe (choć mniej istotne ograniczenia) nakłada także nowa licencja `XFree.org`.

Ponieważ obiecałem, że spojrzę na problem oczami programisty, a nie prawnika (entuzjastów prawa odsyłam do książki Profesora Markiewicza), to spróbuję podnieść kilka ważnych według mnie spraw.

Po pierwsze – każdy Autor ma prawo określić prawa wykorzystywania swojego dzieła – i nie należy się obrażać – można przecież nie używać oprogramowania, którego sposób licencjonowania nam nie odpowiada.

Po drugie – fakt, że oprogramowanie jest udostępniane nieodpłatnie nie oznacza wcale, że nie podlega ono warunkom licencji. Prawa osobiste do dzieła nabywa się z chwilą jego utworzenia, i tak naprawdę nie można się ich zrzec. Można natomiast przekazać prawa majątkowe lub wręcz z nich zrezygnować.

Niestety, większość polskich programistów dość lekko traktuje te sprawy. Posłużę się przykładem:

Autorem popularnego oprogramowania rdesktop (`www.rdesktop.org`), który realizuje znakomicie połączenia z serwerami MS Windows jest Matthew Chapman z Australii. Mało kto jednak wie, że urodził on się jako Kamil Maciej Konarski i to w Wadowicach i że mówi po polsku...

Matt (wraz z innymi deweloperami) udostępnia to oprogramowanie na licencji GPL, a jak wszyscy pewnie wiemy wymaga ona umieszczenia w przypadku wykorzystywania programu do dalszej odsprzedaży odpowiednich informacji o Autorze oraz warunkach licencji i gwarancji.

Przykro mi to stwierdzić, ale w większości produktów wykorzystujących program rdesktop nie znalazłem żadnych tego typu informacji! W moim głębokim przekonaniu jest to postępowanie wysoce nieetyczne. Mało tego – nawet w menu wyboru funkcji jednego z polskich produktów pojawia się napis o treści „RDP Windows". Uruchamia ono (proszę zgadywać) program rdesktop! To już jest podwójne nadużycie, bo narusza zarówno prawa Matta Chapmana, jak i firmy Microsoft!

Firmująca to rozwiązanie firma prawdopodobnie w ogóle nie zastanowiła się nad tak trywialnym problemem licencjonowania oprogramowania i prawami do używania znaku Windows.

Niestety, nic praktycznie nie można z tym zrobić, albowiem w świetle prawa zaprotestować przeciwko takim praktykom może jedynie właściciel praw autorskich lub praw do zastrzeżonego znaku handlowego.

Innym, bardzo niebezpiecznym zjawiskiem jest „zamykanie" otwartego oprogramowania w tak zwanym „oprogramowaniu autorskim". I nie chodzi mi bynajmniej o licencje biblioteczne LGPL, ale o programy, których twórcy jasno podkreślają, że udostępniają je na zasadach licencji otwartej. A przecież treść licencji GPL mówi jasno:

„Możesz rozprowadzać program (lub opartą na nim pracę) pod warunkiem udostępnienia kodu źródłowego lub złożeniu deklaracji o nieodpłatnym udostępnieniu tego kodu"

(dokładny, lecz znacznie dłuższy zapis znajduje się na `www.gnu.org.pl`). Nie wspomnę już o takim drobiazgu, że autorzy tych „programów autorskich" zazwyczaj zapominają podać autorów wykorzystanych przez siebie programów udostępnianych na zasadach GPL lub innych licencji otwartych.

6. Co wobec tego powinniśmy robić?

Przede wszystkim – powinniśmy się nawzajem szanować. Nie ma co ukrywać – nieomal wszystkie powstające obecnie większe programy powstają na zasadzie sklejania w całość poszczególnych modułów – GPL, XFree, MDL i innych licencji otwartych oraz modułów udostępnianych na zasadach komercyjnych (odpłatnie z zamkniętym kodem) i modułów stworzonych we własnym zakresie.

W starym dobrym Uniksie istniał plik `/etc/copyrights`. Wymienione w nim były prawa do poszczególnych użytych modułów. Proponuję, aby polska społeczność związana z oprogramowaniem otwartym wskrzesiła tą dobrą zasadę (ewentualnie mogą być dwa pliki: `/etc/copyrights` oraz `/etc/copylefts`). W każdym razie zasad licencjonowania programów (nawet tych darmowych) powinniśmy bezwzględnie przestrzegać, bo inaczej podcinamy gałąź, na której siedzimy. Zauważone nadużycia praw autorskich do programów objętych licencjami otwartymi należy konsekwentnie piętnować. Nie możemy zrobić wiele więcej, ale w moim przekonaniu idea Free Software Found jest bardzo cenna, bo pozwala ona na ochronę twórców Wolnego Oprogramowania przez Instytucję, która gromadzi na ten cel odpowiednie środki.

7. Własne moduły – czy i kiedy udostępniać ich kody źródłowe

W Polsce panuje na ogół duża niechęć do udostępniania kodów źródłowych oprogramowania. Jest to oczywiście zrozumiałe, ale też w wielu przypadkach bardzo niekorzystne.

Oczywiście, jak już wspomniałem powyżej Autor dysponuje prawami do swego utworu (osobistymi zawsze, zaś majątkowymi pod określonymi warunkami). Jeżeli opracowaliśmy program w ramach naszych obowiązków służbowych, to prawa majątkowe (a tym samym decyzja o sposobie udostępniania programu) należą do naszego pracodawcy. Szefowie mają zaś naturalną tendencję do zamykania oprogramowania (wszak za jego rozwój zapłacili!).

Przejęcie praw majątkowych do oprogramowania wiąże się jednak nie tylko z korzyściami, lecz również z przyjęciem sporej odpowiedzialności. Oprogramowanie zainstalowane u wielu klientów to nie tylko spory dochód, ale również obowiązek jego rozwoju, świadczenia wsparcia technicznego oraz innych dodatkowych usług. Dużo nawet sporych firm odniosło chwilowy sukces sprzedając wiele egzemplarzy oprogramowania składając bardzo daleko idące deklaracje jego wieloletniej obsługi, lecz później niestety zabrakło środków na jej realizację.

Jeżeli decydujemy się na udostępnianie naszego oprogramowania na zasadach komercyjnych musimy dokładnie skalkulować całość przedsięwzięcia! Inaczej sukces może szybko zamienić się w katastrofę. Wiara jest rzeczą piękną, ale rzeczywistość często skrzeczy i pomimo, że wierzymy w jakość naszych produktów lub usług powinniśmy brać pod uwagę możliwe dodatkowe koszty.

W tym miejscu znów posłużę się przykładem z tak zwanego życia: W jednym z ostatnich sporych przetargów oferowano wsparcie techniczne (wraz z organizacją Call Center) dla 1000 jednostek zainstalowanych na terenie całej Polski w cenach od 20 do 50 tys. zł (brutto) i to w okresie 3 lat!

Wypada więc od 20 do 50 zł na jednostkę na trzy lata, czyli od około 7 do 17 zł rocznie! Nie mam pojęcia, jaka kalkulacja doprowadziła do takiego rezultatu, ponieważ nawet pół etatu to dla firmy wydatek co najmniej w wysokości 2000 zł miesięcznie (jeśli ktoś zgodzi się przyjąć taką odpowiedzialność z 1000 zł miesięcznie), a wiec 24 tys. zł rocznie, a w ciągu trzech lat – 72 tys. zł! A gdzie koszty sprzętu, telekomunikacji, delegacji itp.?

No cóż – papier wszystko przyjmie, przetarg się wygra – a potem jak wieszczył Wieszcz:

„...Szlachta na koń siędzie, ja z synowcem na czele i jakoś to będzie."

Alternatywą jest licencja GPL. Zapewnia nam ona pełnię praw osobistych, umożliwia świadczenie gwarancji (w ramach dodatkowego kontraktu), lecz równocześnie nie tylko zwalnia nas od odpowiedzialności za oprogramowanie, lecz również otwiera spore szanse na jego dalszy rozwój. Może więc warto się zastanowić, czy nie jest to w wielu przypadkach opłacalne – nikt przecież nie zabrania sprzedawać oprogramowania objętego licencją GPL!

8. Oczekiwania rynku

Środowisko Wolnego Oprogramowania czuje się w większości głęboko niedocenione. Z jednej strony przecież dysponuje bardzo zaawansowaną technologią i jest przekonane (i słusznie) o ogromnym potencjale idei Open Source, a z drugiej bardzo trudno jest się mu przebić. Oczywiście najłatwiej zwalić wszystko na „marketing firmy Microsoft", ale może skuteczniej będzie spojrzeć sobie głęboko w oczy i zapytać – czy naprawdę jesteśmy w stanie spełnić oczekiwania naszych potencjalnych klientów?

Współczesny odbiorca przykłada przede wszystkim wagę do możliwości nieomal natychmiastowego skorzystania z zakupionego produktu. Sam komputer osobisty, który kiedyś nobilitował jego posiadacza obecnie jest zwykłym towarem rynkowym, sprzedawanym jak przysłowiowa marchewka i przeznaczonym do równie szybkiego zużycia. Twórcy oprogramowania użytkowego i rozrywkowego prześcigają się w upraszczaniu jego użytkowania, a instrukcji już od dawna nikt nie czyta.

Jeszcze około 15 lat temu wraz z zakupem systemu UNIX otrzymywało się ogromne pudło z dokumentacją techniczną w postaci bardzo wygodnych segregatorów, do których łatwo można było wprowadzać uaktualnienia lub własne dodatkowe strony. Mniej więcej 10 lat temu segregatory zamieniły się w tandetne książki zwane w USA „Paper Back", które rozpadały się po kilku miesiącach użytkowania, a potem (oczywiście dla wygody użytkowników!) książki zamieniły się w jeden krążek CD z plikami html.

Oszczędność dla producenta niewątpliwie ogromna – ale nie dam sobie wmówić, że tak jest wygodniej. Rezultat jest oczywisty – dokumentacja jest traktowana jako coś nieomal zbędnego, bo przecież wystarczy napisać setup lub install (dla bardziej wtajemniczonych make, ale już bez wskazywania Makefile) i wszystko powinno zrobić się samo.

Otrzymałem kiedyś zlecenie do uruchomienie drukarek obsługiwanych przez serwer ANNEX dla wiodącego operatora telekomunikacyjnego. Problem polegał na tym, że nastąpiła zmiana systemu operacyjnego wraz z instalacją nowych dużych maszyn uniksowych. Nikt nie mógł sobie poradzić z tym problemem. Ponieważ kiedyś blisko współpracowałem z producentem ANNEX-ów – nieistniejącą już firmą XYLOGICS podjąłem się tego zadania, które wcale nie wydawało mi się trudne. Zabrałem ze sobą dokumentację *ANNEX Unix Administration Guide* i pojechałem 300 km. Na miejscu otworzyłem pokaźną księgę na rozdziale *Printing with ANNEX*, sprawdziłem, że w nowych maszynach zastosowano system drukowania oparty na definicji SYSTEM V (poprzednie wykorzystywały BSD), wprowadziłem odpowiednie (podane szczegółowo w dokumentacji) zmiany w pliku `printer_interfaces` i wszystko zgodnie z moimi przypuszczeniami zadziałało. Przez cały czas przyglądali mi się pracownicy zarówno klienta końcowego, jak i obsługującego go „Autoryzowanego Resellera". Po pół godzinie wszystkie drukarki działały bez zarzutu. Zdziwienie było ogromne, ponieważ „intensywne próby" rozwiązania „problemu" trwały od ponad 2 tygodni, a książka identyczną z tą, którą przywiozłem tkwiła przez cały czas na półce!

Jest to znakomita ilustracja potrzeb współczesnego rynku i oczekiwań użytkowników i administratorów. Czy nam się to podoba, czy też nie nikt nie ma ochoty na takie nudne i zbędne działania jak zapoznawanie się z możliwościami sprzętu i oprogramowania. Komputery powinny być proste i już!

Wobec takich oczekiwań nasze środowisko jest prawie bezradne. Sytuacja przypomina mi początkowy okres rozwoju systemu MS Windows i odnoszenia się do niego środowiska *UNIX Gurus*. Nie używaliśmy wprawdzie określenia *lamer*, ale tak naprawdę to było tak samo. My byliśmy dumni i wiedliśmy długie dyskusje o technologii SMP oraz CC-NUMA, a tam wprowadzono polskie literki, co dla nas było całkowicie niegodnym prawdziwego guru zajęciem.

Większość produktów linuksowych (oczywiście istnieją wyjątki) jest dość surowa, a próby wprowadzenie graficznych wersji programów instalacyjnych w nowoczesnych dystrybucjach spotykało się ze zdecydowanym oporem środowiska. Tymczasem, czy nam się to podoba, czy nie ludzie wolą klikać i wybierać z góry przygotowane opcje lub uruchamiać WIZZARDS lub DRAKE niż używać linii komend. Może się nam to nie podobać, ale to klient głosuje za rozwiązaniem. Kierowcę, który przyzwyczaił się do uruchamiania samochodu rozrusznikiem nie da się przekonać do wykorzystywania korby, choć taki sposób niewątpliwie bardzo skutecznie oszczędza akumulator…

Jeżeli więc chcemy, aby nasze rozwiązania stały się powszechniejsze, to nie możemy lekceważyć oczekiwań przeciętnego klienta. W przeciwnym razie będziemy ograniczeni do własnego grona i trzeba się będzie rozstać z marzeniami o powszechnym wprowadzeniu Open Source we wszystkie dziedziny życia. Argument, że za pomocą Open Source i Linuksa można spełnić wszystkie potrzeby nikogo nie przekona, a w najlepszym przypadku spotkamy się z propozycją – no to pokaż jak? A to może być czasem dość trudne i potencjalny klient po prostu nie wytrzyma kolejnej kompilacji jądra.

9. Jak to zrobić?

Oprogramowanie Open Source to ogromny zbiór wspaniałych idei, znakomitych programów narzędziowych i niestety stosunkowo niewiele gotowych do codziennego użycia programów. Przeciętny użytkownik może wie, że istnieje OpenOffice, niewielu już wskaże na GIMP, a już wyjątki wymienią Mozillę lub Firefox lub Evolution.

Wspomniałem już o programie rdesktop Matta Chapmana. Czy ktoś z przeciętnych użytkowników będzie miał cierpliwość do wprowadzenia na przykład takiej komendy?

```
rdesktop -a24 -kpl -rsound \
   -rlptport:LPT1=/dev/lp0 \
   -xlan -z 10.1.1.77 &
```

Przecież to jedynie kilka z podstawowych opcji połączenia z maszyną MS Windows 2003 server:

- `-a24` to 24 bity koloru,
- `-kpl` włącza obsługę polskiej klawiatury,
- `-rsound` uruchamia obsługę dźwięku poprzez kanał wirtualny,
- `-rlptport:LPT1=/dev/lp0` uruchamia mapowanie drukarki,
- `-xlan` informuje protokół, że wykorzystujemy sieć lokalną,
- `-z` włącza kompresję,

po opcjach mamy adres serwera Windows oraz skierowanie programu w tło.

Bardzo to proste i oczywiste, ale proszę to przekonać przeciętnego użytkownika, ze powinien używać takiej komendy w codziennej pracy. Proszę spojrzeć na program uruchamiający klienta Microsoft RDP chociażby dla Windows XP i będzie jasne o co chodzi. Jeśli nie zrobimy tego w podobny sposób, lub wręcz tak samo, to możemy zapomnieć o powszechnym wykorzystywaniu tego programu. Co najwyżej w czasie pokazu wszyscy pokiwają z uznaniem głowami i być może nawet zdobędziemy powodzenie u Pięknych Pań (jaki on wspaniały, że to wszystko pamięta), ale klient kupi rozwiązanie Windows.

A przecież to w gruncie rzeczy jest bardzo proste. Dysponujemy środowiskiem X Window (jeszcze raz napominam – *Nie X Windows!!!*), oraz szeregiem dość prostych w wykorzystywaniu narzędzi.

Co powinien robić nasz program? W zasadzie nic skomplikowanego. Wystarczy wydać komendę rdesktop podstawiając odpowiednie opcje. Jedno z możliwych rozwiązań może wyglądać mniej więcej tak:

Nasz program przygotuje plik o nazwie na przykład `rdesktop.conf`, w którym zapisze wszystkie opcje, które umożliwią nam zgodne z oczekiwaniami uruchomienie programu `rdesktop`.

Zamiast bezpośredniego wywołania rdesktop z wieloma wprowadzanymi „z palca” opcjami będziemy uruchamiać prosty skrypt, który odczyta plik `rdesktop.conf`, podstawi ustawione w nim opcje pod odpowiednie zmienne i uruchomi całość. Oczywiście, ten skrypt nie będzie naszym interfejsem użytkownika, a jedynie współpracującym z nim programem uruchomieniowym.

Sam interfejs (oczywiście graficzny) przygotujemy wykorzystując TcL/Tk lub Fltk albo jakiekolwiek inne narzędzie pozwalające na stosowanie widgetów.

Tworzymy pierwszą formę dialogową – na przykład ustawienie głębi kolorów. Mamy trzy możliwości – 8 bitów, 16 bitów i 24 bity. Oczywiście nie każemy naszemu użytkownikowi liczyć bitów i stworzymy listę wyboru (standardowy widget) lub trzy przyciski typu Radio (znów standardowy widget) umożliwiające wybór jednej z opcji: 256, 65 tysięcy lub 16 mln kolorów. Do formy dodajemy odpowiedni guzik potwierdzający dokonanie wyboru wywołujący funkcję powodującą zapis do pliku `rdesktop.conf` odpowiednio `-a8`, `-a16` lub `-a24`.

Następnie tworzymy kolejne formy dialogowe – wybór klawiatur, mapowanie drukarki itp. Każda z nich powoduje dokonanie odpowiedniego zapisu w pliku konfiguracyjnym. Całości możemy nadać formę WIZZARD (ustaw parametr $\mapsto$ przejdź dalej, ustaw następny parametr $\mapsto$ przejdź dalej…) uruchamiając po kolei przygotowane formy dialogowe, lub stworzyć większą formę dialogową obsługującą zakładki (każda zakładka odpowiada przygotowanej uprzednio formie dialogowej).

Ponieważ oczywiście myślimy także o poważnych administratorach, którzy brzydzą się „klikaniem” plik konfiguracyjny `rdesktop.conf` może na przykład wyglądać tak:

```
# informacje licencyjne

# wpis dokonany automatycznie przez konfigurator
# ustawienie głębi koloru 24 bity
-a24
# koniec wpisu konfiguratora

# wpis dokonany automatycznie przez konfigurator
# włączenie obsługi dźwięku
-rsound
# koniec wpisu konfiguratora
...
-rcomport:COM1=/dev/ttyS0  # (ten wpis
  # został dokonany edytorem vi ,,z palca'')
...
```

Całą rzecz możemy „pociągnąć” dalej. Jeżeli dysponujemy kilkoma serwerami MS Windows możemy stworzyć pliki konfiguracyjne `rdesktop.conf.1`, `rdesktop.conf.2` i tak dalej.

Jeżeli wykorzystujemy ICEWM (nawet w wersji LIGHT), to w jego pliku konfiguracyjnym opisującym „pop-up menu” odpowiednich zapisów uruchamiających przygotowany wcześniej skrypt, którego argumentem będzie kolejny plik konfiguracyjny. Pozycje w ICEWM Menu będą więc przyporządkowane kolejnym plikom konfiguracyjnym, zaś automatyczne dokonanie zapisu w pliku opisującym menu ICEWM to wspaniałe ćwiczenie z edytora sed (jak ktoś go nie lubi, może wykorzystać inne narzędzie).

Powyższy opis jest jedynie ilustracją tego, o co w tym wszystkim chodzi. W sieci są zresztą dostępne nieodpłatne (choć znacznie prostsze od opisanego powyżej) gotowe nieodpłatne programy realizujące graficzny interfejs użytkownika dla programu rdesktop – nie szukając wiele znalazłem dwa: jeden w Pythonie, a drugi w Perlu, jednak wykorzystanie TcL (można go kompilować) lub Fltk daje znaczną oszczędność miejsca na dysku.

Zdalną administrację możemy zrealizować np. wykorzystując Webmina (najlepiej w połączeniu z SSL).

Ponieważ moje opracowanie to nie kurs tworzenia GUI ograniczę się jedynie do tego krótkiego zasygnalizowania możliwości zbudowania takiego interfejsu i tym samym udostępnienie naszych pomysłów także „pospolitym klikaczom”. Tylko w taki sposób możemy pokazać szerokiemu gronu użytkowników, że Open Source może być proste i przyjazne.

10. Problem wersji 1.0

W pewnym momencie musimy się zdecydować – mamy wersję 1.0!

Programiści reagują generalnie na zbliżanie się tej chwili w dwojaki sposób i dzielą się na perfekcjonistów i entuzjastów.

Perfekcjoniści uważają, że ich program nie jest jeszcze gotowy i w drastycznych przypadkach stan ten może trwać nawet kilka lat. Entuzjaści zaś odwrotnie – ich program jest już wspaniały i gotowy do szerokiego wprowadzenia i wszyscy powinni paść przed jego wspaniałością na kolana.

Oczywiście, oba podejścia są bardzo niebezpieczne. W odpowiednim momencie powinien wkroczyć prowadzący projekt i przeprowadzić obiektywną analizę jego stanu. Jeśli podstawowe deklarowane funkcje programu działają, to należy rozpocząć procedurę testowania. W szczególności powinniśmy zwrócić uwagę na stabilność oprogramowania oraz poprawną obsługę błędów (zawsze mnie zastanawia, jak wiele kombinacji klawiszy mogą nieświadomie wprowadzać użytkownicy).

Zasadą powinno być dzielenie pracy na dobrze zdefiniowane, w miarę możliwości oddzielne moduły. Ułatwia to nie tylko tworzenie oprogramowania, lecz również jego testowanie.

Wprowadzając wersję 1.0 nie należy zakładać, ze jest ona ostateczna. Prawdopodobnie podczas procesu testowania stwierdzimy, że wiele rzeczy można było zrobić lepiej, a co najmniej bardziej elegancko. Jeżeli jednak program realizuje przyjęte założenia i jest zgodny z dokumentacją to powinniśmy wprowadzić wersję 1.0 (co oczywiście nie wyklucza prowadzenia prac nad wersją 1.1). Inaczej nasza praca nigdy się nie zwróci i dołączymy do klubu perfekcjonistów.

Wersji 1.0 bezwzględnie nie należy wprowadzać, jeśli program wykazuje niestabilności, nie jest zgodny z dokumentacją lub przyjętymi założeniami. Oczywiście nie ma bezbłędnych programów, ale należy dokładnie sprawdzić, czy zgodna z instrukcją (a także z intuicją) obsługa oprogramowania nie doprowadza do zakłóceń w jego pracy. Testujący nie powinien się przy tym wykazywać nadmierną inicjatywą – powinien wykonać wszystkie kroki według opisu i na tym koniec. Efekt działania programu ma być zgodny z oczekiwaniami.

Żadna nawet największa firma nie bierze odpowiedzialności za wykorzystywanie nieudokumentowanych opcji oprogramowania! Warto więc napisać porządną instrukcję, bo jest ona dla nas także zabezpieczeniem przed nieuzasadnionymi pretensjami.

Jeżeli więc nasz program zachowuje się stabilnie, realizuje przewidziane funkcje oraz przygotowaliśmy odpowiednią instrukcję to mamy wersję 1.0! Pozostaje wybrać sposób licencjonowania i poddać program osądowi administratorów i użytkowników. . .

11. Siedem naszych grzechów głównych

Grzech pierwszy – brak spójnej koncepcji
Nawet jeśli mamy wspaniały pomysł, to nie wystarczy usiąść do komputera i od razu zacząć klepać kod. Wiele pomysłów nie jest do końca realizowanych z powodu braku sporządzenia spójnej koncepcji.

Grzech drugi – niedokumentowanie własnych działań
Jeżeli dziś potrafimy coś zrobić, to wcale to nie oznacza, że nie sprawi nam to kłopotu za kilka tygodni lub miesięcy! Jeżeli nie zapiszemy, jak to zrobiliśmy, to później będzie nas to kosztować bardzo dużo czasu...

Grzech trzeci – brak troski o szczegóły
Brak polskich liter w konfiguratorze lub niepoprawna obsługa klawiatury może nam nie przeszkadzać w pracy, ale czy dobrze świadczy o naszej staranności?

Grzech czwarty – która to była wersja ???
W szale dokonywania poprawek powstają różne wersje przejściowe. Ponieważ zazwyczaj dokumentowanie prac na bieżąco uważamy za stratę czasu (patrz Grzech Drugi), to możemy się bardzo łatwo znaleźć w sytuacji bez odwrotu.

Grzech piąty – zbyt wczesne samozadowolenie
Nasz program już działa – i to nawet nieźle. Nie oznacza to wcale, że ktoś oprócz nas potrafi go wykorzystać! Sprawdźmy, czy nasz kolega może go uruchomić i używać bez naszych dodatkowych wyjaśnień!

Grzech szósty – zaniedbywanie programów pomocniczych (instalator, konfigurator itp.)
Przecież wystarczy `*.tar.gz` i dostęp do plików konfiguracyjnych za pomocą `vi`! Wszyscy inni to lamerzy i niech się uczą. Można i tak, ale proszę się nie obrażać, że nasz program nie cieszy się zasłużoną i szeroką popularnością.

Grzech siódmy – słomiany zapał
Entuzjazm dla projektu maleje w miarę kontynuowania prac nad nim. Jeżeli nie wykażemy wystarczającej konsekwencji, to projekt zostanie po pewnym czasie porzucony. Przypomina to usychanie niepodlewanej roślinki – na początku wygląda jeszcze dobrze, potem tylko nieźle, a w końcu staje się zbyt późno, aby ją reanimować. Proszę sobie przypomnieć fabryki wysadzone przez Alfreda Nobla i nie zrażać się porażkami. Przegrane bitwy to nie od razu przegrana wojna „Być zwyciężonym i nie ulec – to zwycięstwo" (Marszałek J. Piłsudzki).