I’m ready to wallow now

Thom Holwerda
OSNews.com

Table of contents

Introduction

After the release of the iPhone sent shockwaves across the mobile phone industry, virtually every established player was shell-shocked, and it took them years to get back on their feet - if at all. We've seen a few casualties along the way, but probably none hit me harder on a personal level than the demise of Palm.

As a long-time Palm user, I was very excited when the company unveiled the first Pre and its brand new webOS operating system, all the way back in January 2009. At that point, Windows Mobile still looked like this, Android was still at API level 1, and BlackBerry and Nokia still had no clue what they were doing. Sadly, webOS never made it to The Netherlands, and then, as we all know, Palm was swallowed whole by HP who had absolutely no clue what to do with the knowledge, heritage, and brand they had just bought.

Four years down the line, and here I am, trying to summarise my thoughts on a company that defined the mobile industry.

In this article, I'm going to celebrate the Palm that I loved - the Palm OS Palm. As someone who can still write the Graffiti II alphabet, this is a trip down into the history of the company and the operating system that, in my view, built the foundation of the popular mobile operating systems we use today. The Android and iOS violence might cast its dark shadow upon the rich past of mobile computing, but I will have none of that. Palm is the one that shaped this industry more than any other, and they deserve the credit.

Let's start in the 19th century.

You read that right.

History of handwriting recognition

The history of Palm itself most certainly doesn't extend as far back as the 19th century, as most of you will know. The company was founded in 1992 by Jeff Hawkins, joined by Donna Dubinsky and Ed Colligan, and those of you with a proper sense of history will probably know that, with a bit of effort, you could stretch Palm's history a bit further back to the late 1980s. At that time, Hawkins worked at GRiD, where he created the GRiDPad, one of the first tablet computers and the Palm Pilot's direct predecessor.

To understand Palm's history, you have to understand Hawkins' history. To understand Hawkins' history, you have to look at the technology that was at the very core of the Palm Pilot: handwriting recognition. This technology most certainly wasn't new when Hawkins started working on it, and as early as 1888, scientists and inventors were already working on the subject - in one way or another.

Before we move on, it's important to make a few distinctions in order to make clear what I mean by "handwriting recognition". First and foremost, there's the distinction between *printed* character recognition and *handwritten* character recognition, which I'm assuming is obvious. What's less obvious, perhaps, is that handwritten character recognition further breaks down in online and offline handwritten character recognition. The latter refers to - simply put - scanning handwritten characters and recognising them as such; this is used extensively by postal services to scan handwritten addresses.

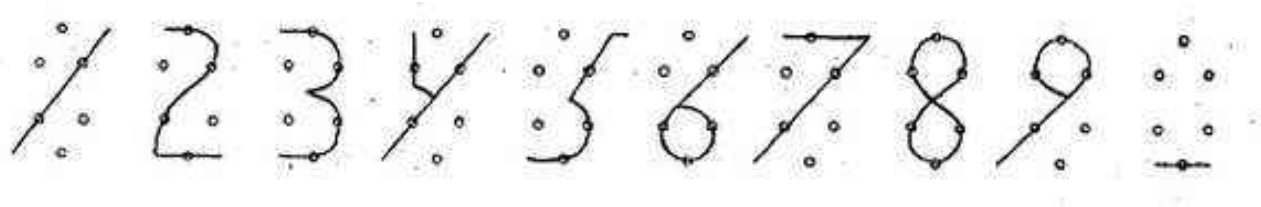
With online handwritten character recognition, characters are recognised as they are written. You could do this in a variety of ways, but the one most of us are familiar with is using a stylus on a resistive touchscreen. However, it can also be done on a capacitive touchscreen, a graphics tablet, or possibly even a camera (I don't know of any examples, but it seems possible). This is the kind of handwriting recognition this article refers to.

First steps

Having said that, the history of handwriting recognition starts in the late 19th century. There were systems which may look like they employ handwriting recognition, the most prominent of which is probably the telautograph. This mechanical device was invented and patented by Elisha Gray - yes, that one - in 1888, and converted handwriting or drawings into electrical impulses using potentiometers, which were then sent to a receiving station, which recreated the handwriting or drawing using servomechanisms and a pen. As ingenious as this system is, it isn't handwriting recognition, because nothing's actually being recognised.

In 1914, a system was invented that is considered to be the first instance of handwritten character recognition. Hyman Eli Goldberg invented and patented his 'Controller', a device that converted handwritten numerical characters into electrical data which would in turn instruct a machine in real-time.

It's quite ingenious. I'm no expert in reading patent applications, and the older-style English and technical writing don't help, but the way I understand it, it's simple and clever at the same time. Characters are written using an electrically conductive ink. A 'contactor', consisting of six groups of five 'terminals' (so, six digits can be written) is then applied to the written ink. The electrically conductive ink of a character will connect the five terminals in a specific way, which creates circuits; in which way these terminals are connected by the ink depends on the shape of the character, thus creating various different circuits (see the below image). These different currents then give different instructions to the machine that's being controlled.



Neither of these systems employed a computer, so we're still a way off from handwriting recognition as we know it today. In addition, there were more systems - more and less advanced than what I've already described - but I'm not going to describe them all; the point I'm trying to make is that the idea of trying to control a machine using handwriting is an old idea indeed, with implementations dating back to the 19th century.

Now let's jump ahead to the late '50s and early '60s, and bring computing into the mix.

Stylator

Before we actually do so, we should consider what is needed to operate a computer using handwriting. It seems simple enough, but consider what computers looked like during those days, and it becomes obvious that a lot had to be done before we arrived at handwriting recognition on a computer. An input device was needed, a display to show the results, a powerful computer, and the intricate software to glue it all together.

The input device was the first part to come. In 1957, Tom Dimond unveiled his Stylator invention, in a detailed article titled "Devices for reading handwritten characters". Stylater is a contraction of *stylus* and *interpreter* or *translator*, which should be a clear indication of what we're looking at: a graphic tablet with a stylus.

Stylator's basic concept isn't all that different from Goldberg's Controller. However, it improves upon it in several crucial ways, the most important of which is that instead of connecting terminal dots with conductive ink to create circuits, you're using a stylus to draw across a plastic surface with copper conductors embedded in it. The conductors are laid out in such a way that with just three lines consisting of seven conductors, all

numerical characters can be recognised. The illustration below from Dimond's article is pretty self-explanatory.

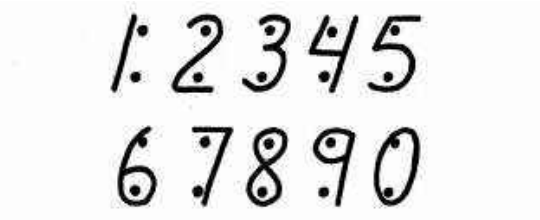


Fig. 4—Numerals with dot constraint.

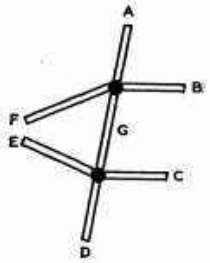


Fig. 5—Set of bipolar coordinates for character recognition.

As you can see, writing numerals 'around' the two dots will ensure the characters can be recognised. When the stylus crosses one of the conductors, the conductor is energised and the combination of energised conductors corresponds to a numeral. This system allows for a far greater degree of variation in handwriting styles than the Controller did, as you can see below with the numeral '3'.

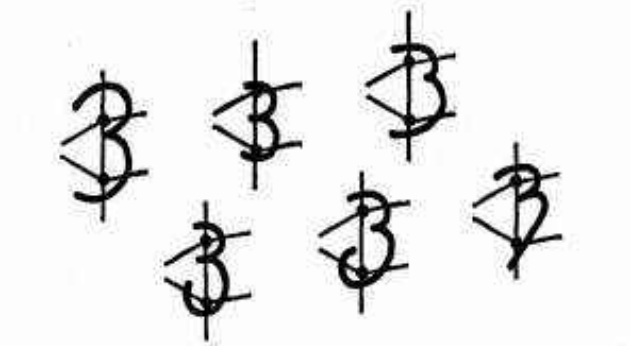


Fig. 6—Range of variation permissible.

The two-dot system can be expanded to four dots to accommodate for all letters in the alphabet, but as you can see in the examples below, it does require a certain amount of arbitrariness in how to write the letters.

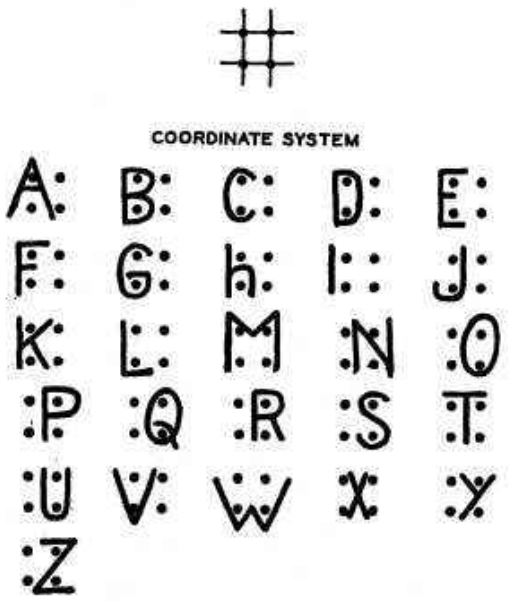


Fig. 8—Four-dot letter restraint—method 1.

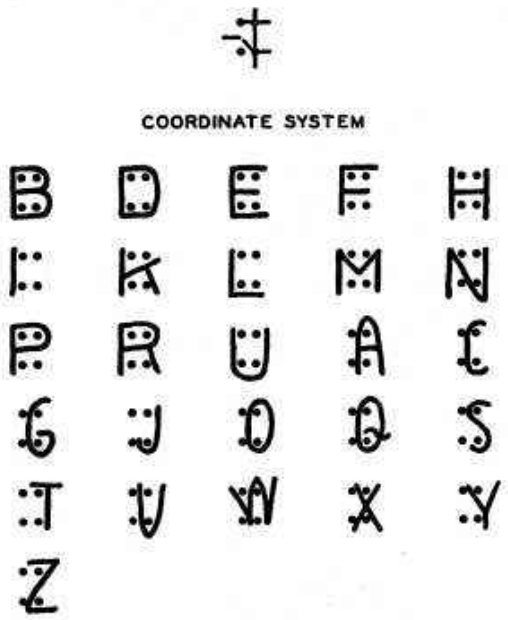


Fig. 9—Four-dot letter constraint—method 2.

Alternatively, Dimond suggests, you can employ the *sequence* in which the conductors are energised to expand the two-dot system to also allow for recognising letters. It's also important to note that the Stylator tablet required you to manually clear the character recognition buffer by tapping the stylus on a separate area because Stylator has no way of knowing when a character is completed.

Dimond lists a number of possible uses for Stylator. "Several uses have been suggested for the Stylator. It is a competitor for key sets in many applications. It has been successfully used to control a teletypewriter. It is attractive in this application because it is inexpensive and does not require a long period for learning to use a keyboard," Dimond writes, "If the criterial areas are used to control the frequency of an oscillator, an inexpensive sending device is obtained which may be connected to a telephone set to send information to remote machines."

There are several key takeaways from Dimond's Stylator project, the most important of which is that it touches upon a crucial aspect of the implementation of handwriting recognition: do you create a system that tries to recognise handwriting, no matter whose handwriting it is - or, alternatively, do you ask that users learn a specific handwriting that is easier for the system to recognise? This would prove to be a question critical to Palm's success (but it'll be a while before we get to that!).

In the case of the former, you're going to need very, very clever software and a very sensitive writing surface. In the case of the latter, you're going to need very simple letters and numerals with as few strokes as possible to make it easy to learn, *but* the recognition software can focus on just that specific handwriting, greatly reducing its complexity. Stylator clearly opted for the latter due to hardware constraints.

The Stylator, while a huge leap forward over earlier systems, was still quite limited in what it could do. To really make handwriting recognition a valid input method, we

need more. Let's make another leap forward, and arrive at a system consisting of a graphics tablet, CRT display, recognition software, and a user interface - essentially a Palm Pilot the size of a room.

The holy GRAIL

Over the course of the 1960s, the RAND Corporation worked on something called the GRAIL Project, short for the Graphical Input Language Project. The description of the project is straightforward: "A man, using a RAND Tablet/Stylus and a CRT display, may specify and edit a computer program via flowcharts and then execute it. The system provides relevant feedback on the CRT display." The entire project is detailed in a three-part final report, and was sponsored by the Advanced Research Projects Agency (ARPA or DARPA, it's been renamed quite a few times) of the US Department of Defense.

The GRAIL Project was part of a larger interest in the industry at the time into human-machine interaction. GRAIL is an experiment into using a tablet and stylus to create computer programs using flowcharts - and in doing so, includes online handwriting recognition, a graphical user interface with things like resize handles, buttons, several system-wide gestures, real-time editing capabilities, and much more.

Let's start with the RAND Tablet/Stylus. I think some of you may have heard of this one before, especially since it was often quoted in articles about the history of tablets published after the arrival and ensuing success of Apple's iPad. The RAND tablet is a massive improvement over the Stylator, and would be used in several other projects at RAND - including GRAIL - even though it was originally a separate research project, also funded by DARPA. As was the case with many other RAND projects at the time, a detailed report on it was written, titled "The RAND Tablet: a man-machine graphical communication device". The summary neatly details the device:

> The Memorandum describes a low-cost, two-dimensional graphic input tablet and stylus developed at The RAND Corporation for conducting research on man-machine graphical communications. The tablet is a printer-circuit screen complete with printed-circuit capacitive-coupled encoders with only 40 external connections. The writing surface is a 10"×10" area with a resolution of 100 lines per inch in both x and y. Thus, it is capable of digitizing >10^6 discrete locations with excellent linearity, allowing the user to "write" in a natural manner. The system does not require a computer-controlled scanning system to locate and track the stylus. Several institutions have recently installed copies of the tablet in research environments. It has been in use at RAND since September 1963.

As I already mentioned, during those times a lot of research went into improving the way humans interacted with computers. After coming to the conclusion that the then-current interaction models were suboptimal for both computer and user, scientists at RAND and elsewhere wanted to unlock the full potential of both user and computer. A number of these projects were "concerned with the design of 'two-dimensional' or

‘graphical’ man-computer links” (in other words, the first shoots of the graphical user interface).

From the very beginning, RAND focussed on exploring the possibilities of using “man’s existent dexterity with a free, pen-like instrument on a horizontal surface”. This focus led to the eventual creation of the RAND Tablet, which was, as we already saw in the description in the summary above, quite advanced. The technical workings are slightly beyond my comfort zone (I’m no engineer or programmer), but I believe I grasp the general gist.

The tablet consists of a sheet of Mylar with printed circuits on each of its two sides; the top circuit contains lines for the x position, while the bottom circuit contains lines for the y position. These lines are pulsed with negative and positive pulses, which are picked up by a stylus with a high input impedance. Each x and y position consists of a specific sequence of negative and positive pulses; negative pulses are zeros and positive pulse are ones, which, when combined, lead to a Gray-pulse code for each x,y position. These can then be fed into a computer where further magic happens.

This is just a basic description of how the system works, greatly simplified and based on a very simple, 8×8-line version of the RAND Tablet used in the article for explanatory purposes. There’s a lot more interesting things going on deeper in the system (such as ignoring accidental movements), and if you want to know more technical details I highly recommend reading the article - it’s quite readable.

The tablet itself was not a goal per se; it was a means to an end, with the end being to make it easier for humans to interact with computers. With this in mind, the RAND tablet would return to the forefront several years later, when RAND unveiled the GRAIL Project. At OSNews and other places, you’ve probably heard a lot about Douglas Engelbart’s NLS, the revolutionary work done at Xerox PARC, and the first commercially successful graphical user interfaces developed at Apple (the Macintosh), Commodore (AmigaOS), and Digital Research (GEM). Yet, I’ve never seen or heard anything about GRAIL, and to be honest, that’s a shame - because it’s bloody amazing.

I will provide a summary on what the GRAIL Project entails, but for those of you interested in the nitty-gritty, I advise you to read all three in-depth articles on the project (a total of 126 pages, so grab a coffee) and simply skip my summary:

1. The GRAIL Project: an experiment in man-machine communications
2. The GRAIL language and operations
3. The GRAIL system implementation

The goal of the GRAIL Project was to develop a ‘common working surface’ for both human and computer - a CRT display. They concluded that the flexibility of the output (the CRT display) should be matched by the flexibility of the input, so that direct and natural expression on a two-dimensional surface was possible, and that’s - obviously - where the RAND Tablet comes back into play. The project had four design objectives:

1. to use only the CRT and the tablet to interpret stylus movement in real-time
2. to make the operations apparent
3. to make the system responsive
4. to make it complete as a problemsolving aid

This led them to the creation of a graphical programming language which uses flowcharts as a means for the user to instruct the computer to solve problems. The flowcharts were drawn by hand on the tablet, and would appear on the screen in real-time. Much like Ivan Sutherland's Sketchpad, the user could draw a 'messy' shape (say, a rectangle), and the computer would replace it with a normalised variant. He could then manipulate these shapes (resize, move, alter) and connect them to create a flowchart. He could also write on the tablet, and have it appear on the screen - and much like the rectangle, the computer would recognise the handwritten characters, and turn them into normalised characters.

To facilitate the interactions, a dot on the display represented the position of the stylus on the tablet, and real-time 'ink' was drawn on the display whenever the stylus was pressed onto the tablet. The tablet surface corresponds 1:1 with the display surface. These three elements combined allowed the user to remain focussed on the display at all times - clearly an intermediary step towards modern high-end graphics tablets which combine pressure sensitive digitisers and styluses with displays.

The system also contained several elements which would return in later user interfaces, such as buttons and resize handles, and would even correct the user if he drew something 'unacceptable' (e.g., drawing a flow from one symbol to another if such a flow was not allowed).

Thanks to the wonder of the internet and YouTube, we can see GRAIL in action - and narrated by Alan Kay. Kay even states in the video that one of the window controls of the Mac was "literally" taken from GRAIL.

The GRAIL Project also introduced several gestures that would survive and be used for decades to come. The caret gesture was used to insert text, a scrub gesture to delete something, and so on. These gestures would later return in systems using the notebook UI paradigm, such as PenPoint OS and Newton OS.

The biggest challenge for the GRAIL Project engineers was to ensure everything happened in real-time, and that the system was responsive enough to ensure that the user felt directly in control over the work he was doing. Any significant delay would have a strong detrimental effect on the user experience (still a challenge today for touch-based devices). The researchers note that the computational costs for providing such accurate user feedback are incredibly high, and as such, that they had to implement several specialised techniques to get there.

For those that wish to know: the GRAIL Project ran on an IBM System/360 Model 40-G with two 2311 harddisks as secondary storage and a Burroughs Corp. CRT display, and the basic operating system was built from scratch specifically for GRAIL. Despite the

custom nature of the project and the fact that the System/360 was available to them on an exclusive basis, the researchers note that the system became overloaded under peak demands, illustrating that the project was perhaps a bit too far ahead of its time. At the same time, they also note that areas were being investigated to distribute the processor's load in a more evenly manner.

While those of you interested in more details and the actual workings at lower levels can dive into the three articles linked to earlier, I want to focus on one particular aspect of the GRAIL Project: its handwriting recognition. I was surprised to find just how advanced the recognition system was - it moved beyond 'merely' recognising handwritten characters, and allowed for a variety of gestures for text editing, as well as automatic syntax analysis to ensure the strings were valid (this is a programming environment, after all).

To get a grip on how the recognition system works, we have to step away from the GRAIL Project and look at a different research project at RAND. The GRAIL articles treat handwriting recognition rather matter-of-factly, referring to this other project, titled "Real-time recognition of handprinted text", by Gabriel F. Groner, from 1966, as the source of their technology.

The RAND Tablet had already been developed, and now the task RAND faced was to make it possible for characters handwritten 'on' the tablet to be recognised by a computer so they could be used for human-machine interaction. The researchers eventually ended up with a system that could recognise the upper-case Latin alphabet, numerals, and a collection of symbols. In addition, the scrubbing gesture (for deletion) mentioned earlier was also recognised.

There were a small number of conventions the user had to adhere to in order for the character recognition software to work properly. The letter O had to be slashed to distinguish it from the numeral 0, the letter I needed serifs to distinguish it from the numeral 1, and Z had to be crossed so the system wouldn't confuse it with the numeral 2. In addition, characters had to be written separately (so no connected script), and cursive elements like curls had to be avoided.

Already recognised text could also be edited. Any character already on the screen could be replaced simply by overwriting it (remember, the tablet and display corresponded 1:1). In addition, characters could be removed by scrubbing them out.

So, let's get to the meat of the matter. How does the actual recognition work? The basic goal of a handwriting recognition system is fairly straightforward. You need the features which are most useful for telling one character apart from the other; features which remain fairly consistent even among variations of the same character, but differ among the various characters. In other words, you want those unique features of a character which are always the same, no matter who writes the character.

First, you need to get the actual data. As soon as the stylus is pressed on the tablet's surface, a switch in the stylus is activated, which signals the recognition system that a stroke has been initiated. During a stroke, the recognition system is notified of the

position of the stylus every 4 ms (each position is accurate to within about 0.127 mm). When the stylus is lifted off the surface of the tablet, the recognition system is notified that the stroke has ended.

The set of data points received by the recognition system is then smoothed (to reduce noise) and thinned (to remove unnecessary data points). The exact workings of smoothing and thinning are defined by a set of formulas, for which I refer you to the article (I don't understand them anyway). In any case, the goal is to reduce the amount of processing required by reducing the number of individual data points.

The character (now represented by data points) is then analysed for features like curvature, corners, size, and several position features. The entire character is divided up into a 4×4 grid, and the features are located within any of the grid's 16 areas. With this information in hand, the recognition system's decision making scheme makes the call which symbol - if any - has just been written. The recognition system can handle characters consisting of multiple strokes, and it's smart enough so that the user no longer needs to inform the system when a character has been completed.

To give you an idea of how much variation is allowed, look at the below set of strokes. Each and every one of them is recognised as a 3.

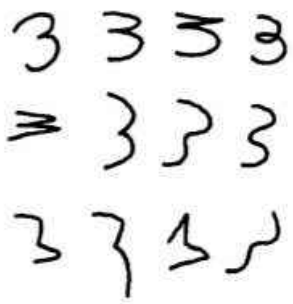


FIG. 4 - Some Tracks Recognized as the Symbol "3".

The accuracy of the system proved to be very high. The researchers asked people with zero experience with the system to sit down and use it, and they found that the average accuracy rating was 87% (I doubt I can even hit that with modern touch keyboards). People with previous experience with the system hit an average of 88%, and those that helped design the system - and, in fact, on whose handwriting the system was based - hit 93%. The researchers found that several characters proved especially problematic, such as [vs (, or the asterisk.

The team concludes as follows:

> The recognition program responds quickly and is efficient in storage. When the time-delay normally used to separate symbols is set to zero, the lifting of the pen and the display of a recognised symbol are apparently simultaneous. The recognition program - including the data analysis and decision-making routines,

> and data storage; but not display or editing routines - requires about twenty-four hundred 32-bit words of memory.

The system proved to be capable enough to be used in the GRAIL Project, as you could see in the video (although it was most likely refined and expanded by that point). It's incredibly impressive to see what was possible given the limited resources they had to deal with, but if there's one thing I've learnt over the years pouring over this kind of stuff, it's that limited resources are a programmer's best friend.

So, GRAIL was the whole nine yards - a tablet and pen operating an entire system using handwriting, shape, and gesture recognition. What's the next step? Well, as fascinating and impressive as the GRAIL Project is, it's 'only' a research project, not an actual commercial product. In other words, the next step is to see who first brought this idea to market.

And here, we run into a bit of trouble.

Going to market

We run into trouble because I can't seem to find a lot of information about what is supposedly the first product to bring all this to market. Jean Renard Ward, a specialist and veteran in the field of pen computing, character recognition, and similar, has created a comprehensive bibliography concerning these topics, and put it online for us to peruse through.

In it, he notes that Applicon Incorporated, a company which developed, among other things, computer aided design and manufacturing systems, developed the first commercial gesture recognition system. He references one of the company's manuals, which, as far as I can tell, is not available online. He wonders if Applicon, perhaps, uses the Ledeen character recogniser.

At first, I couldn't find a whole lot of information on Applicon (you'd be surprised how little you can do out of the Dutch countryside). There's a Wikipedia page for the company, but it lacks verification and citations, so I couldn't judge the validity of the claims made. Wikipedia claims that Applicon's products ran on PDP-11 machines from DEC, and that, much like the GRAIL Project, they used a tablet mapped to the display for input, including gesture and character recognition. However, without proper citations, it was impossible to verify.

And then, during one last ditch attempt to find something more tangible, I struck gold. David E. Weinberg has written a detailed history of CAD, titled "The Engineering Design Revolution", which is freely available online (a 650-page treasure trove of awesome stuff). Chapter 7 deals entirely with Applicon and the company's history, and also includes a fairly detailed description of the products it shipped.

Applicon's early systems, built and sold in the early 1970s, were repackaged PDP-11/34 machines from DEC, which Applicon combined with its own Graphics 32 processor and called the AGS/895 Central Processing Facility. The software was

written in assembly, and used a custom operating system (until DEC's RSX-11M came out in 1987). The unique selling point here was the means by which the user interacted with the system. As described by Weinberg:

> The key characteristic of Applicon's AGS software was its pattern recognition command entry or what the company called Tablet Symbol Recognition. As an example, if the user wanted to zoom in on a specific area of a drawing, he would simply draw a circle around the area of interest with his tablet stylus and the system would regenerate the image displaying just the area of interest. A horizontal dimension was inserted by entering a dot followed by a dash while a vertical dimension line was a dot followed by a short vertical line. The underlying software was command driven and these tablet patterns simply initiated a sequence of commands. The system came with a number of predefined tablet patterns but users could create patterns to represent any specialized sequence of operations desired.

While it doesn't specifically state that it employed handwritten character recognition, it can be inferred from the description - a letter or numeral is simply a pattern to which we arbitrarily ascribe meaning, after all. Applicon supposedly used the Ledeen gesture recogniser, which, in some literature, is actually called the Ledeen character recogniser, and is capable of handwriting recognition. I fully understand if some of you think I'm inferring too much - so I'd be very happy if someone who actually has experience with Applicon's products to step forward and correct me.

From the 1970s and onward, multiple commercial products using handwriting recognition, styluses and tablets would enter the marketplace. Take the Pencept Penpad, for instance, a product on which Jean Renard Ward actually worked. He's got a fascinating video and several images on his website demonstrating how it worked - basically a more compact version of the systems we just discussed like GRAIL and the AGS/895 Central Processing Facility.

As a sidenote, on that same page, you'll also find more exotic approaches to handwriting recognition, like the Casio PF-8000-s calculator, which used a grid of rubberised buttons instead of a digitiser. Even though it has no real bearing on this article, I find the concept quite fascinating, so I wanted to mention it anyway. There's a video of it on YouTube showing it in action.

These relatively early attempts at bringing handwriting recognition and pen input to the attention of the greater public would later catch the attention of the behemoths of computing. GO Corporation, technically a start-up but with massive amounts of funding, developed PenPoint OS, which Microsoft perceived as such a threat that after several interactions between the two companies, Redmond decided it had to enter the pen computing market as well - and so, Windows for Pen Computing was born, which brought pen computing to Windows 3.x. Apple, of course, also followed this trend with the Newton.

All these products - PenPoint OS, Windows for Pen Computing, the Newton - have one thing in common: they were commercial failures. Fascinating pieces of technology,

sure, but nobody bought and/or wanted them. It wasn't until the release of the original Palm Pilot that pen computing and handwriting recognition really took off.

Driving the point home

If you've come this far, you've already read approximately 6000 words in the form of a concise history of handwriting recognition. While this may seem strange for an article that is supposed to be about Palm, I did this to illustrate a point, a point I have repeatedly tried to make in the past - namely, that products are not invented in a bubble. Now that patents rule the industry and companies and their products have become objects of worship, there's a growing trend among those companies and their followers to claim ownership over ideas, products, and concepts. This trend is toxic, detrimental to the industry, and hinders the progress of technology.

I've just written 6000 words on the history of handwriting recognition, dating back to the 19th century, to drive the point home that the Palm Pilot, while a revolutionary product that has defined the mobile computing industry and continues to form its basis until today, was not something that just sprung up out of thin air. It's the culmination of over a hundred years of work in engineering and computing, and that fancy smartphone in your pocket is no different.

With that off my chest, let's finally talk about Palm.

Palm's hardware

Note: *I'm not going to detail the various different names under which Palm has operated over the years, such as PalmSource, PalmOne, and so on. For the sake of clarity, I'm just referring to all of them as 'Palm'.*

In order to understand Palm, its origins, its history, and its philosophy, you really have to take a close look at just four products: the GRiDPad, the Zoomer, the Pilot 1000, and lastly, the Palm V. Not entirely coincidentally, these four products were all the brainchildren of Jeff Hawkins - founder of Palm.

Let me just throw all the cards on the table so hard they slice it in two: there is no single person who has had a greater and more everlasting impact on the mobile computing industry than Jeff Hawkins. His achievements include the first consumer tablet computer, one of the first PDAs, the first *successful* PDA, and the first successful smartphone. Not entirely coincidentally, as you read through this chapter, you'll see that his philosophy on product design bears a striking resemblance to that of another industry heavyweight.

Much of the information in this chapter (specifically that related to Hawkins) comes from the following sources:

- Jeff Hawkins Oral History - a fascinating in-depth interview with Jeff Hawkins from 2002, covering his entire career up until that point.
- Hawkins video interview by Michael Chui - a recent interview, from November 2012.
- Short video interview with Hawkins.
- A talk by Hawkins on how Palm approached product design.

All of these are very interesting and quite entertaining. Definitely worth a watch and read.

GRiDPad

Jeff Hawkins already had a rather diverse education and employment history before he joined GRiD. After growing up with an adventurous inventor as a father, he earned a B.S. in electrical engineering at Cornell University. With that degree in hand, he was employed by Intel in 1979, where he took on several different roles over the years - among others, he worked on single-board computers and training and teaching others about microprocessor design (including the ambitious but ill-fated Intel 432).

He left Intel behind in 1982 to go work at GRiD, a startup at the time. GRiD had *literally* just invented the modern laptop, and Hawkins was responsible for putting together the training material for this device. In other words, from relatively early on in his career, he was involved in mobile computing. He also developed GRiDTask, a high-level RAD programming language.

However, his real passion was neuroscience. Before he joined GRiD, Hawkins had already tried to get into MIT with a research proposal concerning a big problem in neuroscience - namely, that despite everything we know about the brain, we don't really have any idea how it actually works. Hawkins wanted to work on this problem, but MIT basically told him it was a waste of time. He didn't drop the idea, though, and during his time at GRiD he kept working on the problem in the evenings.

After trying Stanford, which turned him down because they didn't have any activity in the field he wanted to study, Hawkins ended up at Berkeley, where he was allowed to do an independent study. He took two years off at GRiD, but during those years, he didn't make the progress he wanted - he started to doubt himself, so he decided to go back to work at GRiD "for a few years", and maybe he could pick it up later. This was 1988.

It's during this second stint at GRiD that Hawkins first gave the world a taste of what was to come. During his two years away from GRiD, and as part of his study into neuroscience, he also came into contact with neural networks, a field which spawned a number of companies trying to apply neural networks commercially. One of these companies was Nestor, which had developed a handwriting recognition system based on neural networks. They asked a million dollars for the system - something which surprised Hawkins.

> I thought, "What? A million dollars for this thing that I don't think is even very useful? I can do that." So I went home that night and created my own handwriting mechanism software. I said, "If they can sell it for a million dollars, I can do it better." So I did it without using neural networks, using some of the math I was doing for the brain stuff.

He wrote it in one evening, or so the legend goes. This is when the idea for the GRiDPad was born; he took his handwriting recognition engine to GRiD, and told them he wanted to build a tablet, with a stylus and everything. He would be responsible for the entire project - hardware, software, the whole thing. GRiD was a bit nervous about it, according to Hawkins, but they agreed.

And so, in 1989, the GRiDPad saw the light of day. It was the first tablet computer with pen input in a self-contained unit (the Linus Write-Top was released a few years earlier, and was very similar, but consisted of two separate units connected via a cable), and used Hawkins' handwriting recognition system, then dubbed PalmPrint.

Hardware-wise, it weighed a hefty 2 kilograms, and ran on a Intel 80C86 at 10Mhz. It sported 1MB of RAM, two RAM card expansion slots, and a serial port. The display was a 10" LCD with 32 grayscales, 80×24 text or 640×400 pixels, and the stylus was attached to the device. It ran MS-DOS 3.3, and was aimed squarely at vertical markets. The United States military and Chrysler supposedly bought quite a few of them, and GRiD claimed that 10000 were sold in 1990 - which is not bad for one of the first products in its category, and cost $2370 a pop.

Regular consumers couldn't actually buy a GRiDPad - only businesses who needed dozens or more of them were eligible to buy them. Still, those that did use GRiDPads were fascinated by the device, and some of them told Hawkins - why don't you make a smaller version of this? Something I could carry in my pocket, which holds all my personal stuff, and is a lot cheaper?

Click.

Zoomer

Hawkins knew what to do. GRiD wanted him to do it, but he was apprehensive about doing the product at GRiD, because it was an enterprise company, and in his mind, he envisioned the as-of-yet unnamed product as something for ordinary consumers. Hawkins mentioned his product idea to a lot of people, and some of them suggested he start his own company. He was scared about this at first, but after investors started *coming to him*, basically begging he'd go through with it, he did so. Tandy had acquired GRiD in 1988, and also decided to invest in Hawkins' new company.

And so, in January 1992, Palm was born.

Still, it wasn't until March 1996 that the Palm Pilot 1000 - the first Palm Pilot - was launched. What did Palm do in between? Well, Palm contributed to a product that looked a lot like a Palm Pilot in many ways, but was actually a complete and utter disaster. It was a design-by-committee hodgepodge of parts from different companies, neither of which seemed to have a very good idea of what Hawkins was trying to achieve.

This product was the Zoomer. Try to keep up, but the Zoomer consisted of Casio hardware, the GEOS operating system, Palm's PIM applications and handwriting recognition, and Tandy marketing. According to Hawkins, they couldn't agree on anything, and nothing got done. Worse yet, in comes Apple, out of nowhere, with the Newton. Apple hyped the Newton and the PDA concept into the stratosphere, and they managed to ship before the Zoomer could. "Everything is falling apart! No matter what happened, we would have failed. The Zoomer was a terrible product," Hawkins reminisces about those days.

The Zoomer was slow, clunky, and unpleasant to use. There's a great video on YouTube that shows wait cursors, the clunky handwriting recognition (with a dialog box!), slow redraw, and so on. To be honest, I didn't think it was as bad as Hawkins describes it in later, more current interviews, but it's still clear this is miles behind what we came to know from Palm OS.

Hardware-wise, the Zoomer ran on a NEC V20 processor - which was a bit of an oddball to begin with. It sported 1 MB of RAM, a 4MB ROM, a miniature RS-323 port, infrared receiver, and a PCMCIA slot (type 2). It had a 320×256 monochrome LCD, and had a promised battery life of 100 hours on 3 AA batteries. It cost a rather whopping $700.

The Zoomer became a market failure, but in a surprising turn of events, Apple's involvement in the PDA market was a blessing in disguise for Palm, because the Newton was just as terrible as the Zoomer. Apple hyped the Newton to an insane degree, and Sculley claimed it would reinvent personal computing - this meant Apple got all the press and attention, but when the Newton turned out to be a dud, *it also got all the flack*.

Palm remained largely under the radar, which gave the company the chance to realign its goals. Nobody wanted to invest in the PDA category anymore, no company wanted to devote resources to it, which turned out to be another blessing in disguise: it meant Palm had to do everything on its own - software *and* hardware.

While the Zoomer was a failure, it taught Palm and Hawkins a number of very valuable lessons, and because of those lessons, it was still a very important product in Palm's - and therefore the mobile industry's - history. First and foremost, Hawkins realised that in order for a consumer product to work, you had to do both the software and the hardware (as Alan Kay already noted).

Secondly, the failure of the Zoomer prompted Palm to do something no other company in the industry had yet done: instead of guessing what consumers wanted, why don't we just ask them? After talking to consumers, Palm came to a rather crucial realisation. Hawkins summed the survey responses up as follows:

> I don't want a complex thing. I was just trying to figure out a way to get my life organized. Today I use paper, and I'm always trying to coordinate with my assistant. And we can never keep calendars. We're always printing things out. I'm just trying to get my life organized. All I really care about is calendars and address book and trying to coordinate with my secretary. All this other stuff? I don't need all this other stuff. Just solve my basic organizational problem.

Palm realised that they didn't need to revolutionise computing. They weren't competing with computers, *but with paper*. They also learned that synchronisation should be a core part of the product - keeping things organised - and that it should be simple and straightforward.

With the blessing of Palm's board, and the realisations from the survey in hand, Hawkins set to work. In one evening, he created a wooden model of what would become the Palm Pilot and its cradle.

Palm Pilot

After the failure of the Newton and the Zoomer, the PDA market - which had never gotten off the ground to begin with - was pretty much dead. GO had gone bust, and EO and General Magic were about to follow. Palm had no PDA products, and Apple was peddling the Newton which nobody bought (it was one of the first product lines Steve Jobs axed when he returned to the company). It was a failed market category, and nobody was interested in investing in it.

So, Palm had Hawkins' model for the Pilot, and they knew pretty well what they wanted to build. Hawkins defined four core criteria the Palm Pilot had to fulfil:

- It had to be pocketable. You had to be able to carry it around without being bothered by it.
- It had to be the ideal price point: $299.
- Synchronisation with the desktop had to be built-in (as opposed to an aftermarket add-on as it had been treated as up until then).
- It had to be fast, without any wait cursors. Remember: it had to compete with paper, and paper is instant.

And Palm only had $3 million to do it, which is a laughably low budget for a complicated product like this. This lack of funds initially forced them to make some interesting decisions, like opting for a small, local, two-person low-budget design shop for the industrial design. Luckily, though, US Robotics was fascinated by the idea for the Palm Pilot, and decided to flat-out buy Palm, which provided Hawkins and his team with the required funds (3Com then bought US Robotics in 1997).

And so, with US Robotics' funds, Palm managed to complete the device's development, and meet the four targets it had set for the device. On 29 January, 1996, a press release was sent into the world, titled "U.S. Robotics launches breakthrough pocket-size connected organizer for PC users", detailing the breakthrough new device.

> The Palm computing division of U.S. Robotics today announced Pilot, a line of handheld electronic connected organizers designed to work as companion products to a desktop or laptop computer.
>
> Created by U.S. Robotics to be the first connected organizer, the Pilot family of products is designed to meet the needs of PC users who want to manage their activities both remotely and on their desktops.

The Pilot 1000 - the first model - used a Motorola Dragonball processor, a low-power processor based on the design of the 68000 family of processors, with a speed of 16 Mhz. It had 128 KB RAM built-in (512 KB for the Pilot 5000 model) and 512 KB of ROM for the operating system and applications, but this could be upgraded to a maximum of 12MB of RAM and 4MB of ROM via slots behind a plastic cover on the back of the device (in fact, the Pilot 1000/5000 could be upgraded to Palm OS 2.0 this way).

The display was a 160×160 grayscale LCD (no backlight), with the characteristic Graffiti input panel underneath. The input panel was flanked on both sides by permanent tappable shortcuts to home, calculator, search, as well a button to open the current application's drop-down menubar. Underneath the display sit two scroll buttons (up/down), buttons to access the most important PIM applications, and a power button. The casing was plastic, and the device weighed 160 g.

It came with a cradle, which was connected to the PC via a serial cable. Place the Pilot in the cradle, press the HotSync button, and the synchronisation software would do its

thing. It was compatible with all major PIM suites at the time, and more support was added as time went on and the product proved to be a hit. The desktop-side synchronisation software initially only supported Windows 95 and Windows 3.11 through Win32s, while version 2.0 added support for Windows NT. Mac OS 9 and Mac OS X support was added in later versions. Linux was never officially supported, but third party solutions existed.

The Palm Pilot 1000 and 5000 ran Palm OS 1.0, but we'll get to the operating system later.

After the Pilot 1000 and 5000, Palm launched the upgraded versions, the PalmPilot Personal and Professional, in 1997. A year later, in 1998, they followed up with the Palm III. The Palm Pilot was a success - by 1 December 1999, Palm had sold over 5 million PDAs - and this attracted others to the PDA market. Most notably: Microsoft.

Hawkins had one last trick up his sleeve to deal with the Redmond behemoth.

Palm V

The success of Palm's products got the attention of Microsoft, and the company pretty much announced it was going to crush Palm. According to Hawkins, Microsoft had a sales conference, where, at some point, a big target appeared on the projecter screen, with the Palm logo dead in the centre of it: "we are going to crush and kill these guys", was the central message. Hawkins recalls that he got condolence letters after that, stating things like "Sorry Jeff. Too bad."

Remember that in the late '90s, Microsoft was at the very pinnacle of its power, and pretty much had the entire computer industry on a leash. BeOS a possible threat? Microsoft forced OEMs into not selling it preinstalled. Microsoft wanted to enter the PDA market? Let's tell our OEMs to make a bunch of PDAs running our Windows PocketPC operating system, and we'll surely crush Palm and own the market before Hawkins can say "what the...".

Even though everyone assumed Microsoft would win, Hawkins devised a rather crazy-sounding and, at the time, controversial idea. This idea was based on how Hawkins viewed Microsoft: according to him, Microsoft simply didn't understand why Palm was successful. Sure, they could throw lots of money around, and wield unimaginable amounts of influence, but at the of the day, *they don't build the entire product*. Microsoft can only do the software, but for the hardware, they have to rely on others - and those others "just aren't very creative".

Microsoft was stuffing Windows PocketPC full of features. Palm OS was eclipsed by all the things you could do with it, and all the functionality that was packed into these PocketPCs. There was increasing pressure within Palm to try and match Microsoft feature-for-feature, but Hawkins realised that as a small company, Palm would never be able to do so. It would be a battle lost before it even began.

> And I said, “No, we’re not going to add any features. Nothing. We’re going to make a beautiful product. They can’t make a beautiful product because they don’t make products. They just make software.” And if I tried to catch up to them on features, the reviews are going to be: “Palm tries to catch Microsoft features. Doesn’t do it.” If I don’t do any new software and I just make a beautiful piece of hardware, the reviews are going to be: “Palm does beautiful piece of hardware. Microsoft hardware looks ugly!” We had this big battle internally where I basically said: “We’re not going to chase after Microsoft on features.”

No new features.

A decade later, Apple’s Bertrand Serlet stood on a stage during WWDC 2009, to announce Mac OS X 10.6. He proudly proclaimed that it had “no new features”. According to Serlet, a move that was “unprecedented” in the PC industry. The audience clapped enthusiastically. The press ate it up.

But I digress.

In any case, this was the premise behind the Palm V: no new features, focussing entirely on the industrial design instead. Palm worked together with industrial design firm IDEO, and out of this collaboration the iconic and revolutionary Palm V was born. The Palm V was a huge leap forward compared to the Pilots that had come before, and it introduced an iconic shape that many will, to this very day, associate with Palm (the below photo of a Palm Vx, the refreshed version, was kindly provided by Kurt Roesener).

The Palm V was built out of anodised aluminium, and was only half as thick as its predecessors - a mere 10 mm. It was also smaller and lighter, but retained the same

screen dimensions. Several other aspects were revolutionary too - it was the first Palm powered by a lithium ion battery instead of AAA batteries. The battery was non-removable, a 'feature' which sent ripples across the smartphone industry when Apple did the same for the iPhone eight years later. For aesthetic reasons, the device had no screws - the case was glued in place.

There's an interesting story to this, actually. The manufacturer was very apprehensive about using only glue, and was afraid that the glue would melt if users, say, left the device on the dashboard of a car in the sun. The manufacturer wanted to delay the release of the Palm V by four months to redesign the product to use screws instead. Hawkins had to personally go down to the manufacturer and convince them the glue would work - Hawkins' father was a prolific inventor, and Hawkins grew up with loads of different materials all around him in the house, including loads of different types of glues. In other words, Hawkins knew his way around product materials, including glue.

The Pilots that had come before were strictly utilitarian, focussed on businessmen and women instead of general consumers. The Palm V changed all this. Its shape would define the company's products for years to come. It had smooth curved sides with a slightly wider bottom than top section, making it all not only look distinct and beautiful, but also very comfortable to hold. Whether you looked at other PDAs, smartphones, or mobile phones of its era - there was nothing else like it. Everybody else was building plastic monstrosities.

I personally never owned a Palm V (sadly), but two PDAs in my collection still sport the same general design - a Tungsten E2 and a Palm T|X.

The Palm V was a smashing success. For the first time, a mobile computing device was designed to be beautiful, and "it turned out to be very successful. We turned it into a personal artefact, or a personal piece of jewellery or something and [Microsoft] couldn't compete with that," according to Hawkins. Palm was riding high at this point,

and the Palm V succeeded in keeping Microsoft at bay. Of all the PDAs sold between January 1999 an July 2002 (in the US), 68% were Palm devices, while Handspring and Sony (both Palm OS licensees), brought Palm OS' PDA market share up to 90%. The remaining 10% was Microsoft, through Casio, Compaq, and HP (source).

In other words, Microsoft had failed to "crush and kill" the company.

Defining Palm

The four products discussed in detail - the GRiDPad, Zoomer, Pilot, and Palm V - have all played a crucial role in defining Palm's success and brand. The GRiDPad made Hawkins realise what people really wanted out of a mobile computer. The Zoomer taught him how *not* to do it. The Pilot proved there was a market for a simple, easy-to-use, mobile computer. Finally, the Palm V made it clear that a focus on beautiful hardware formed the final piece of the puzzle in appealing to a wide audience.

During its heydays, Palm enjoyed the kind of dominance Android currently enjoys. However, unlike most Android device makers, Palm enjoyed Apple-like margins - in 2000, almost 40%. Palm proved that there was a huge, profitable market for easy-to-use, beautifully designed mobile computing devices - long before iOS and Android came along.

With the hardware handled, let's move to software. Let's talk Palm OS.

The Palm operating system

How do you describe an operating system whose ideas, concepts, and approaches have pretty much defined the entire mobile computing industry? This question has been running through my head ever since I decided to write this article. A device's operating system and the user interface it employs are what you interact with on a daily basis; if the operating system and you, as a user, click, you form somewhat of an emotional bond with it over the years.

This hasn't happened on many occasions with me. Regular OSNews readers know that I have such a connection with what I consider the best operating system ever made, BeOS. QNX, back when it was possible to run it as a desktop operating system with its quirky PhotonUI, also has a special place in my heart. And that's about it.

So, what about Palm OS? You might think, after an article like this with more than 21000 words or so, that I have somewhat of a special connection with it. In reality, though, not really. I respect it for what it represents and for how it has defined the industry to this very day - but as a former Palm OS user, my memories of it aren't always fond, especially not when you take off the rose-coloured glasses of nostalgia. To further complicate matters, back in the PDA heydays, I kind of preferred the dark side.

This is not going to be a complete, thorough, 100% description of Palm OS and its architecture - I'm going to cover a number of lower-level aspects that I find fascinating, after which I'll move on to the more user-visible parts of the operating system. Let's start with the kernel Palm OS used.

The kernel

I had always assumed that Palm OS used a homegrown kernel, but as it turns out, that's only partially true. From version 1.x to version 4.x, Palm OS used the AMX 68000 kernel, one of the many kernels available for the AMX Real Time Operating System from Kadak. The AMX RTOS uses many different kernels depending on the target platform, so you have kernels for 8086 real mode, PowerPC, MIPS, ColdFire, and so on. Since Palm devices used 68000-based Motorola Dragonball processors, they opted for the AMX 68000 kernel.

However, Palm didn't use this kernel as-is. They only used the specific parts that they really needed, and ditched everything else. AMX 68000 is a preemptive multitasking kernel, but since there's no user-facing multitasking on Palm OS, there's little to no need to expose the more complex features of the kernel that make this possible to developers. In addition, Palm wanted to prevent the kernel from becoming too complex, since they wanted to retain portability (this turned out to be a very smart decision) for everything that ran atop the kernel.

"The palmOne, Inc. Palm OS does not provide access to the underlying AMX 68000 Multitasking Kernel," Kadak explains on its website, "Instead, it provides a preconfigured collection of AMX tasks and AMX compatible device drivers which

manage the Palm device. The Palm OS then provides an application API which offers the range of services needed by most developers to extend the functionality of the Palm device into a particular application area."

Jim Schram, while working at PalmSource in 2002, provided some more details on a Palm developer mailing list. "The Palm OS uses AMX tasks, timers, semaphores, mailboxes, and signal groups. Nothing too complex," he stated, "We wanted to keep it simple to enable easier porting to other kernels (which we've done in Palm OS 5) and because at the time AMX was chosen, there was no need for more sophisticated features."

Up until the Palm OS 4.x series of releases, Palm continued to rely on the AMX 68000 kernel. However, it became clear right around the start of the new millennium that the Dragonball architecture was no longer competitive. As such, Palm switched to the ARM architecture with the release of the Palm Tungsten T in November 2002, which opened the platform up to a whole slew of new possibilities, such as improved multimedia support, higher screen resolutions, and so on.

Switching to a new architecture meant choosing a new kernel. While Palm could've opted for the AMX 4-ARM kernel, the company decided to 'go custom' and create its own kernel. Therefore, the first version of Palm OS 5 includes an entirely new and custom kernel, written by Palm, affectionately called 'MCK' after its author. Sadly, I was unable to find out who these initials (?) refer to.

For its initial version, Palm decided to - again - keep things simple, and implement more or less the same functionality for MCK as the old AMX 68000 kernel provided, meaning that the same third party developer restrictions still applied. Still, Palm did have plans to expand the functionality - just not for its first release. "Support for additional background threads, timers, etc. is being added in a future version of the Palm OS. However, in v5 we thought it better not to bite off more than we could chew," according to Schram.

It's clear that Palm did indeed have at least some ideas about and plans for structural improvements to the lower levels of the operating system - just how serious these ideas and how far along the plans were will most likely remain a mystery forever. The ill-fated Palm OS 6 Cobalt (which I'll get back to later in the article) may provide some answers to this particular question.

The switch from Dragonball to ARM didn't just necessitate a different and/or new kernel; Palm OS also needed a method to ensure that the huge existing collection of applications written for Palm OS 4 and earlier would still run on Palm OS 5 and its different architecture. Palm's solution to this problem was the Palm Application Compatibility Environment ("Professional Palm OS Programming" is a great book on Palm OS' internals), or PACE.

For any call to a Palm OS function made by an application compiled for 68K, PACE executes the ARM equivalent of said function. This normally creates a lot of overhead, but the ARM processors were so much faster than the Dragonball ones that 68K

applications actually ran *faster* using PACE than running natively on a Dragonball device. It was also possible to create a native ARM code module, which could be linked into a 68K application - this was especially important for applications that did a lot of number crunching.

PACE also took care of a more exotic problem arising from switching between the two architectures - endianness. Since I don't want to butcher this, I'll just refer to the description found in "Professional Palm OS Programming".

> In addition to handing 68K operating system calls off to their native ARM counterparts, the PACE converts 68K data structures into their ARM equivalents. Unlike the big-endian 68K architecture, ARM stores multibyte integers in little-endian format. ARM hardware also requires that 4-byte integers be aligned on 4-byte boundaries; such integers need to be aligned only on an even boundary in the 68K architecture. These differences in data storage mean that Palm OS structures, such as those used to contain information about a form, differ significantly between 68K and ARM.

The end result of PACE was that the transition from 68K to ARM was relatively transparent for users. I can't recall ever encountering any issues running 68K applications on ARM Palm devices, but then again, that doesn't mean they didn't exist. The only complication I can recall is that Palm OS didn't provide support for fat binaries, and as such, you had to choose between a 68K and an ARM version when downloading an application.

There is no file

I'd like to cover how Palm OS handles files, because it has a rather peculiar way of handling them, as unlike most other operating systems, it doesn't actually make use of a file system. We're going to get quite technical here, and to be honest, this goes a bit over my head, but I do believe I grasp the general gist of what's going on.

Very crudely put, a conventional file system does not edit information stored in a file 'in place'. Instead, the file (or the relevant part of it) is loaded into a memory buffer, where it is read and/or altered, after which it's written back to disk, including the changes. However, Palm OS uses RAM for both dynamic and storage purposes, and (especially earlier in its existence) had limited quantities of either. As such, Palm OS' engineers took a different approach.

Instead of relying on the concept of files, Palm OS uses 'records' and 'databases'. Records are chunks of memory, which are stored in databases. A database, then, is a list of memory chunks and a bunch of metadata about the database as a whole, the database header. The header contains a number of fields such as `name` and `version`, but the most important of these fields are `type` and `creator`. The former tells the system if it's an application or a data database, while the latter makes it possible for the system to associate various databases with one another.

It vaguely reminds me of application bundles in systems like RISC OS, NEXTSTEP/ GNUstep/Mac OS X, and others.

Records that need to be edited are not loaded into RAM first; instead, they're edited in place by Palm OS' memory manager and data manager APIs. In other words, chunks do not have to be loaded into dynamic RAM, meaning you need less of it, while also eliminating the overhead associated with moving data between dynamic RAM and storage. Because of this, application developers are not allowed to directly interact with records, because they might screw things up; they must use the memory manager and data manager APIs, because if they edited them directly in a manual fashion, they might damage them, which is particularly problematic due to them being edited in place.

Other than RAM and performance improvements, this approach also has a few other benefits, the most important of which is that databases' records can be intermingled with one another (i.e., it doesn't matter where a record is stored). This means that adding, removing, or deleting records does not require other records to be moved around, further reducing overhead.

When you transfer Palm OS databases to a regular file system, they get the file extension .prc or .pdb. That's why when you download a Palm OS application from the web, it often comes with the .prc extension. "The PRC file, then, is simply the flat file representation of a Pilot resource database," Theodore Ts'o explains, "When the PRC file is loaded into the Pilot, it is converted into a resource database using the Palm OS routine `dmCreateDatabaseFromImage()`."

Single-tasking (but not quite)

Conventional wisdom: Palm OS cannot multitask. You can only run one application at a time, you can't have applications or services running in the background, and opening an application will kill the current one. This conventional wisdom is true, but not entirely. There's a few details about how the Palm OS handles running applications that are quite fascinating, and which, at times, reminds me of Android's Application Components and activities.

Theoretically, the situation on Palm OS is quite straightforward. There can be only one application running at any given time. An application is closed when another application is launched. Interestingly enough, this includes the actual application launcher itself - so, unlike, say, iOS or Android, where respectively Springboard or your-launcher-of-choice continue to run even if another application is running, the Palm OS launcher is actually terminated every time another application is opened.

Technically speaking, the home button on a Palm OS device doesn't take you *back* to the launcher, it actually *launches* it. For some reason, I find this quite fascinating. Alternative launchers also existed for Palm OS.

On a more technical level:

> Palm OS Garnet and earlier versions of Palm OS have a preemptive multitasking kernel. However, the User Interface Application Shell (UIAS), the part of the operating system responsible for managing applications that display a user interface, runs only one application at a time. Normally, the only task running is the UIAS, which calls application code as a subroutine. The UIAS doesn't gain control again until the currently running application quits, at which point the UIAS immediately calls the next application as a subroutine.

This all seems fairly straightforward. However, as with just about anything in life, nothing is as simple as it seems. Palm OS actually contains a number of mechanisms which allow for some rudimentary forms of something that could be classified as multitasking. It's not quite the kind of multitasking you can do on your octocore Windows 7 box with 32GB of RAM, but the capabilities still somewhat question the conventional wisdom about Palm OS.

I'm going to discuss three of them: launch codes, notifications, and helper notifications.

The first of these mechanisms in Palm OS is the concept of launch codes. As their name implies, launch codes launch applications. The most basic launch code is `sysAppLaunchCmdNormalLaunch`, which instructs the application to do a full, regular launch and display its user interface. Tapping an application's icon in the launcher (or pressing one of the hardware buttons) generates this launch code.

Launch codes can also be used to instruct the application how to behave. There are quite a number of standard launch codes the operating system provides, and, when necessary, application developers can define their own launch codes as well. For example, `sysAppLaunchCmdGoToURL` launches an application and tells it to pass a URL, while `sysAppLaunchCmdFind` finds a text string.

Furthermore, launch codes can be accompanied by two types of information (as per the Palm OS Programmer's Companion):

- A parameter block, a pointer to a structure that contains several parameters. These parameters contain information necessary to handle the associated launch code.
- Launch flags indicate how the application should behave. For example, a flag could be used to specify whether the application should display UI or not.

The cool thing about launch codes is that they can also be generated by applications to launch each other without actually switching applications from the user's perspective. I.e., if application Abc uses a launch code to launch application Xyz, application Abc's UI remains visible to the user. From a technical standpoint, application Xyz is now the current application (and thus has access to storage), but from the user's standpoint, Abc is still the current application because its UI is still visible (one a side note, there's also `SysUIAppSwitch` - which does exactly as the name implies. It will close your application and launch another one, including its interface).

There are countless ways this system can be useful. For instance, a mail client can use launch codes to instruct the address book application to find a specific phone number and return the results - without actually opening the address book. Many launch codes are sent to all applications at once, but only relevant applications will respond. This means that when you are searching for a phone number using a launch code sent to all applications, the music player won't do anything because that obviously wouldn't make sense.

The second mechanism I want to discuss in light of pseudo-multitasking are notifications. You are forgiven for thinking I'm talking about these things, but I'm not. Notifications, in Palm OS parlance, are sent by an application or the system itself to an application, instructing it to do something (or not).

The difference between launch codes and notifications it that while launch codes can only be sent to and received by applications, notifications can be received by any code resource, like shared libraries and system extensions. In addition, an application or resource will only receive notifications it has specifically registered to. In other words, they're more efficient than launch codes, which are often sent to all applications.

Furthermore, notification clients (the receiving ends) can add to the notification upon receiving them. For instance, the Palm OS Programmer's Companion notes the antenna on the Palm VII (a truly intriguing device in and of itself); a client could set the `handled` flag for the `sysNotifyAntennaRaisedEvent` notification to `true` to inform other applications.

Every client can also add its own information to a notification, so it can actually collect data on applications. A potential use for this - inspired by modern mobile operating systems like Android and Windows Phone - is the creation of a share menu. You could send out a notification asking which applications can share a photo, and every relevant application would add itself to the notification, through which the share menu could then be populated.

As a side note, Palm OS does actually have 'notification-notifications', i.e., what you most likely think of first when you hear 'notifications'. There are three related frameworks in Palm OS - the Attention, Alarm, and Notification Managers - that deal with these, and Palm OS provides support for dialogs, beeps, sounds, LED blinking, a subtle indicator in the top-left, and vibration.

The third and final mechanism related to pseudo-multitasking are helper notifications. These are used when one application wants to request a service from another application. The Palm OS Programmer's Companion gives the example of the Address Book application requesting that the Dial application dial the phone number the user has tapped. Palm OS also included other services through helper notifications, such as emailing, texting, and faxing. Remember that in the pre-smartphone/pre-wifi era, many of these services also required bringing up BlueTooth and making a connection to a separate phone. Crucial, of course, is that application developers can define their own services to use with the helper notification framework - and surprise, they can.

Of course, applications could query what other applications supported. It could ask an application for a list of services that it provides, request that the application make sure it can perform the service, and, finally, the actual request to perform the service. The benefit of this system over, say, launch codes is that launch codes require that the application developer knows the actual launch codes and the ID of the application it wants to launch, whereas with helper notifications you're using a standardised framework that doesn't require any application-specific knowledge.

None of these mechanisms constitute true multitasking by any stretch of the imagination. In essence, all of them are examples of inter-application communication which can perform some of the tasks that would otherwise be done through traditional multitasking (on, say, a desktop computer). All this is not done because Palm OS is inherently incapable of multitasking.

On a technical level, Palm OS is capable of multitasking just fine; in fact, during syncing, the Sync application starts a second task (the first one being the UIAS) to handle communication with the desktop computer (as far as I can tell, this is the only official case where multiple tasks are running - and even then, this second task is running at a lower priority to maintain responsiveness for the UI).

Even on the user interface level, there's provisions for multitasking: tapping/pressing and holding the home button opens a menu of recently used applications, ordered by last used. This way, you can quickly switch applications, and thanks to Palm's incredibly quick application loading (generally, zero lag), it *feels* like multitasking. To further add to the illusion of multitasking, Palm OS applications can save state.

Combine all these things, and Palm OS offers a surprising amount of options to developers with which they can create a pretty convincing illusion of true multitasking. Add to this the fact that developers get to define their own launch codes, notifications, and helper notifications, and you can envision some pretty advanced stuff. In fact, the screenshot application I used for this article uses helper notifications to 'run' in the background.

Not bad for a single-tasking operating system.

Graffiti

That long-winded first part of this article wasn't for nothing. It's impossible to talk about Palm OS without addressing what arguably is its most recognisable feature: Graffiti, its handwriting recognition system.

Before the Pilot, the Zoomer already came loaded with Palm's handwriting recognition, then dubbed PalmPr, which recognised a user's natural handwriting, and you could write anywhere on the screen. However, recognising a user's natural handwriting is hard, and quite prone to errors. So, when working on developing the Palm Pilot, Hawkins had an epiphany - instead of trying to get the computer to recognise a user's natural handwriting, why don't we create a simple, easy-to-learn alphabet that's much easier to recognise (remember the key takeaway from the Stylator)? Sure, users had to learn a handwritten alphabet, but hey, users need to learn to touch type, too, right?

And so, Hawkins developed the Graffiti alphabet. Graffiti was simple, and, quite ingenious in that it consisted entirely of single strokes, which made it much easier to write with the stylus. Since I can't resist writing this one down: it was a stroke of genius.

During that one faithful evening when Hawkins mocked up the wooden model for the Palm Pilot, he also came up with the idea of having a dedicated writing area. Up until then, PDA-like devices had you write anywhere on the screen, from left to right. This posed obvious problems, problems that I can personally attest to when playing with whole-word write-anywhere input methods: you run out of space. The displays and digitisers are usually not sensitive enough for small handwriting, so you tend to write larger to ensure proper recognition. The consequence is that more often than not, you have to cut off mid-word because you just reached the bezel.

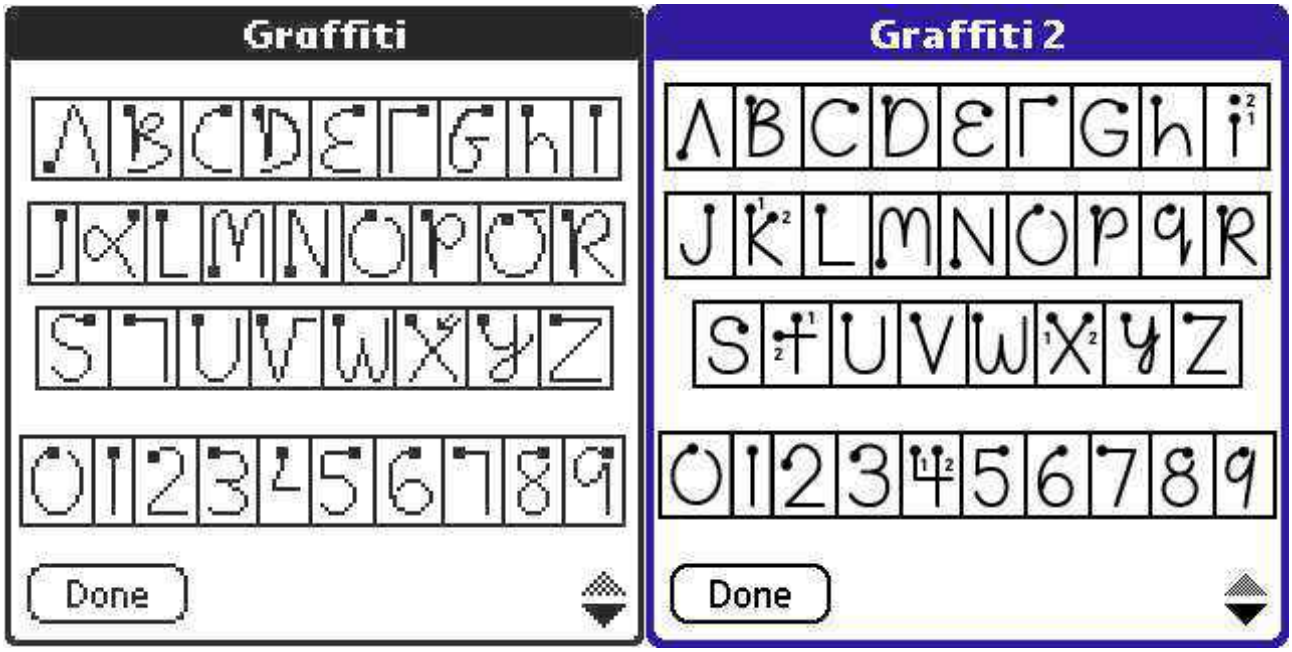
Hawkins devised the concept of writing everything down per letter, one atop the other. The single-stroke Graffiti alphabet made it possible for the system to understand that whenever the stylus was lifted from the screen, a character had been completed. The writing area was further refined by dividing it up into two areas: one for lower case, one for numerals. The area on the dividing line between the two areas was used for upper case.

Graffiti was a success even before the first Palm Pilot was released. Palm also released Graffiti for the Newton, and it was the default handwriting recognition system on GEOS-based portable devices like those from HP or devices running Magic Cap. Combined with selling synchronisation software to HP, this allowed Palm to get some income in between the failure of the Zoomer and the arrival of the Palm Pilot.

Sadly, the '90s computing industry wasn't free of patent lawsuits. Xerox claimed it had a patent on single-stroke alphabets, and successfully took Palm to court (in 1997) over Graffiti (even though the single-stroke alphabet in the patent bears little resemblance (page 5) to Graffiti, but alas). While the case went back and fourth a few times, Palm had to eventually pay Xerox $22.5 million in retroactive licensing fees in 2004.

In response to the then-ongoing lawsuit, Palm had to alter Graffiti to circumvent Xerox' patent. So, in 2003, Palm unveiled Graffiti 2, which, for the most part, they didn't actually develop themselves. They licensed Communication Intelligence Corporation's Jot handwriting recognition system, improved it, and labelled it Graffiti 2. It shipped in Palm OS 5.2 for ARM devices and Palm OS 4.1.2 for 68K devices.

The biggest change was that, in order to circumvent the single-stroke nature of the original Graffiti, a few characters now consisted of two strokes. This made the characters look more like their regular, written counterparts, but at the same time made them more error-prone to write. To this day, I often fail the dual stroke characters.



ACCESS, Palm OS' current owner, actually provides Graffiti for Android. Weirdly enough, this is Graffiti 1 - I'm guessing the licensing fees paid to Xerox covered non-Palm OS implementations (or the original patent has expired).

The user experience

Now that I've covered some of the lower level aspects of Palm OS that I found interesting, I want to talk a little bit more about the user experience. Unlike the technical details that I've covered so far, this is a more esoteric subject matter, and relies more on personal experiences, interpretations, and expectations, and is, probably, coloured by nostalgia. Just like with the lower levels of the operating system, I'm not going to cover everything, focussing instead on a few things that stand out to me.

First and foremost, the most defining characteristic of Palm OS - at least, in my view - is the sheer speed of the user experience. I've already offered a few glimpses into just how far the Palm OS engineers were willing to go for speed, and of course into how vital this was for Hawkins. He wanted the Palm Pilot to compete with paper's instant nature, and they managed to do just that.

Despite the archaic hardware Palm OS ran on - compared to today's massive smartphone beasts - almost everything you did was almost always instant. It often feels as if Palm OS responds even *before* you tap. Of course this isn't the case, but that's how it feels. I know of only one operating system that is as responsive as Palm OS is, and that operating system ironically ended up in the same place as Palm OS did.

My 'primary' Palm OS test and play device is a Palm T|X - the last true Palm PDA, which has all the bells and whistles, and runs the latest and greatest release of the operating system, Palm OS 5.4.9. It's running an Intel XScale PXA 270 at 312 Mhz (don't be fooled - that's ARM) with 32MB of dynamic RAM, and everything you do in Palm OS is instant. Even though this is technically the latest and greatest PDA Palm ever built, its processing power pales into insignificance compared to the quad-core powerhouses we lug around today. The fact that our modern smartphone operating systems actually *need* all this power and still aren't as responsive as Palm OS is telling.

That's all fine and great, but what happens if we take a step back in time and specifications and leave ARM behind? I've got a Sony CLIÉ PEG-SL10 running Palm OS 4.1, which uses the old 68K architecture, a Motorola DragonBall VZ 33 MHz to be precise, with a mere 8 MB of combined dynamic and storage RAM. On this device, too, Palm OS is instant. In fact, the operating system responds so well that the *display* becomes a bottleneck; the PEG-SL10 is equipped with a 320×320 monochrome display, but it's not TFT - it's STN. STN has a slower response time than TFT, and it shows; it just can't keep up with Palm OS.

As a Palm OS user, you expected stuff to be fast. You could spot a non-native application from a mile away, and not only because it looked different and didn't follow Palm OS conventions - non-native applications didn't adhere to the as-fast-as-paper mantra, and thus, were slow and far less responsive. I honestly believe that my almost militant desire for consistency between applications on an operating system stems partly from my days as a Palm OS user (and partly because of BeOS, of course), because something that looked out of place was probably going to be slow and cumbersome.

Palm OS wasn't only fast according to today's standards; it was also fast compared to its contemporary competition. Windows PocketPC (Windows Mobile) was Palm OS' primary competition through much of its lifespan, but when it comes to sheer speed and responsiveness, PocketPC was a laughable joke compared to Palm OS. As I've already hinted, I was a heavy PocketPC user, and I actually happen to have an iPaq with almost identical hardware as the Palm T|X (same processor), but it's a frustrating mess of loading times and spinning wait cursors.

It wasn't all unicorns and sunshine, though. Later in its life, you could see that Palm started to experience more and more problems with trying to extend the functionality of the operating system. Wifi, for instance, always felt like a bolted-on hack; connecting to the network was incredibly slow, and in order to preserve battery, it disconnected and turned itself off every time the device went to sleep. Very cumbersome.

The browser situation on Palm OS was a complete disaster, too. The browsers available were generally quite slow, and even when they were current they were terrible at rendering websites. Basically anything that required a direct internet connection through wifi was slow - whether it be loading a web page or checking for and downloading new emails.

It's easy to explain why: when Palm OS was originally conceived, the web and wifi were in its infancy, so they were never a core consideration during most of the

development of the operating system. When it became clear that both wifi and mobile browsing became important for mobile devices, Palm had already spun off Palm OS development into a separate company, which turned out to be a complication for the further development of Palm OS (we'll get back to post-5.4 Palm OS later).

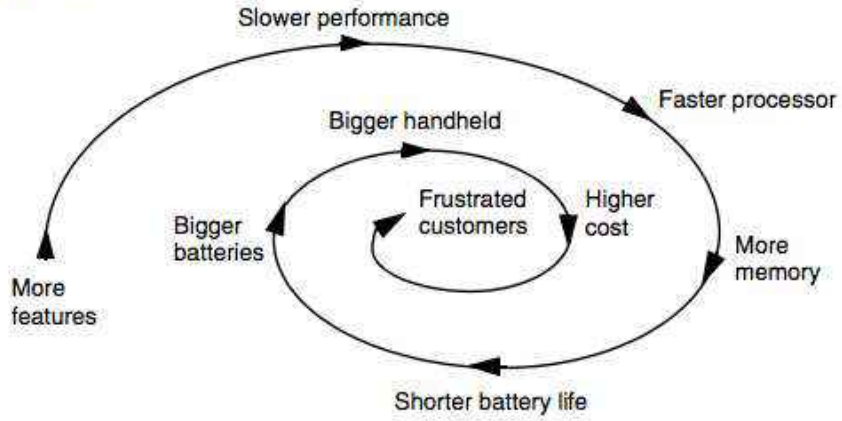
Zen of Palm

The last aspect of Palm OS that I'd like to cover in detail is what Palm itself referred to as "Zen of Palm". Zen of Palm is a core part of the operating system and its applications, and is one of the prime reasons why the platform became so successful in the first place. It defined every nook and cranny of the operating system, from its focus on speed to the design of applications themselves. Let's take a look at what this means.

One of the design goals for the original Palm Pilot, and all Palms that came after it, was that instead of competing with a regular computer, Hawkins realised it had to compete with paper (like a paper diary, address book, and so on). Other than meaning that the system had to be fast, it also meant it had to be simple, straightforward, and task-focussed. All this combined was governed by a set of design philosophies called 'Zen of Palm'.

The most important aspect of Zen of Palm is that when it comes to handheld computing devices, more features usually isn't better. I could spend a lot of words on trying to explain why, but just look at the 'spiral of doom' below.

Figure 1.1 More can be less



In the world of regular computers, more features generally means better. Related to that, faster hardware generally also means better. These two forces strengthen each other: if you pile on more features, you're going to need faster hardware. If you create faster hardware, developers will pile on more features. Lather, rinse, repeat. Our current desktops may be sporting quad cores with 3700 kajiggers, but do they feel any faster than the machines you had a few years ago?

Palm realised that in order to make a handheld device that people wanted to use, it needed to break with the convention of the time that more equals better. So, Palm

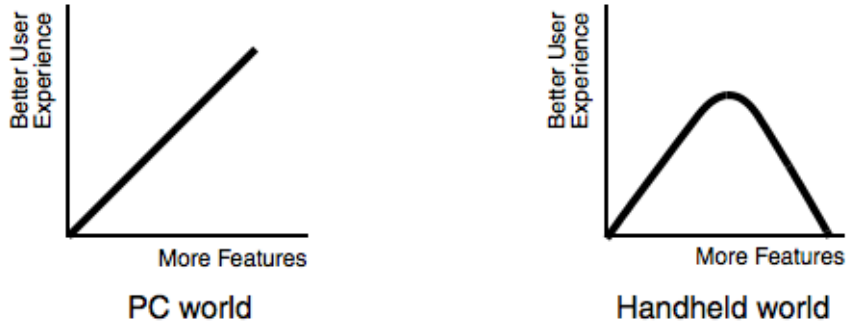
made two very bold decisions. First, it would only implement a very small set of features that it believed customers needed. Second, and related to this, Palm actively deemphasised the importance of specifications like processor speed and amount of RAM.

By not trying to keep up with the features and specifications race, Palm could focus on what it believed to be the 'essence of handhelds', distilled down into five core tenets:

- **Handhelds excel at perceived speed.** The obvious one: a handheld must be fast and responsive. Users don't care about the technical specifications, but only about if they can quickly get it out, find what they're looking for, put it away, and continue with whatever they were doing. Your handheld can have a quad core processor with 2 GB of RAM, but if it's slow and unresponsive, the user doesn't benefit from the fast specifications.
- **Too many features frustrate customers.** Speed suffers from having too many features in more ways than just overtaxing the hardware. Too many features also distract from the core features the user is most likely going to need because they get in the way.
- **A handheld must be free to roam about.** A handheld user shouldn't have to worry about battery life. Adding in lots of hardware and software features will only tax the battery more, causing the user to constantly worry about battery life, and if he'll make it through the day. Modern smartphones have piled on the features and specifications, but battery technology hasn't improved all that much. We all know the end result.
- **Handhelds must be wearable.** A handheld must be small and light. Loads of features and hefty specifications will require a bigger battery, making the device less pleasurable to wear. There's no such thing as a universal 'correct size', though. Take the Galaxy Note devices - they tend to be more popular with women than with men. The reason is simple: many women carry a purse to put the Note into. Men don't.
- **Handhelds are about the user.** This last one is a philosophical combination of the other four. Simply put, adding too many features degrades the user experience of a handheld device.

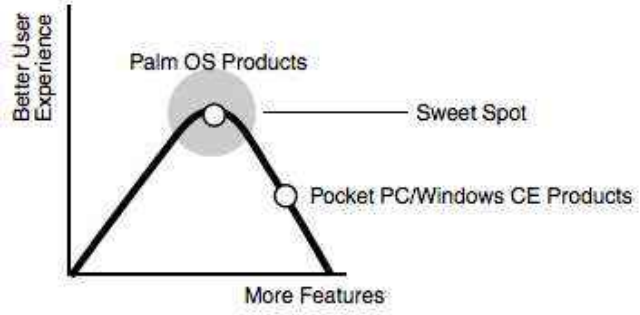
In the form of two simple graphs:

Figure 1.2 Feature list vs. user experience



Palm even provided a version comparing Palm OS to Pocket PC.

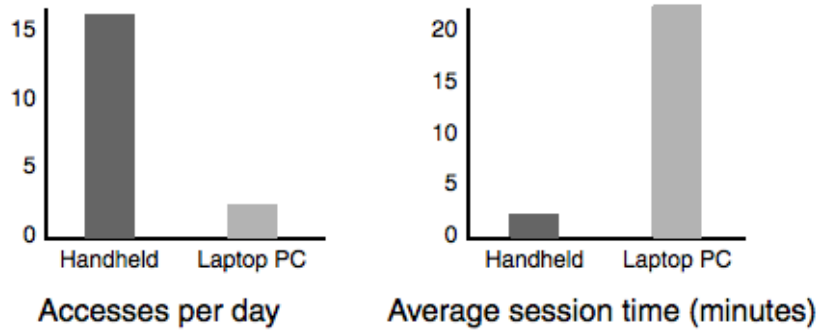
Figure 1.4 Different views on new features



When Microsoft put that Palm logo in the centre of a crosshair and vowed to kill and crush the company, they thought they could do so the only way they knew how. Microsoft was a PC software company, so they figured they could crush Palm through more and more features, and better and better specifications, user experience be damned. This tactic worked for Redmond in the PC business, so why wouldn't it work for the handheld market?

Palm knew why: it wouldn't work because usage patterns on handheld device are exactly the opposite of those on PCs. Handhelds are used very frequently, but in short bursts to quickly perform small tasks. PCs, on the other hand, are used less frequently, but the user sessions are much longer, and the tasks tend to be more complex and demanding. Palm OS was designed for the handheld usage pattern; Windows Mobile was not.

Figure 1.3 Opposite usage patterns



Source: Palm, Inc. user surveys.

Because of all the boatloads of features Microsoft haphazardly shoved into Windows Mobile, the user experience was slow, complex, and frustrating. OEMs tried to solve this by faster hardware, but this significantly harmed battery life. In response, Pocket PCs got bigger batteries, making them heavier and more cumbersome. Windows Mobile had entered the spiral of doom, and there was no way back. The end result was that Palm OS dominated the handheld market.

Palm urged application developers to adopt the Zen of Palm approach to application design. I'm not going to detail all the advice Palm gave out, but there are a few nuggets that stand out and that I'd like to highlight. I urge you - especially if you are a mobile application developer - to read the whole document, since it's just as relevant today as it was ten years ago.

The most crucial question is probably *how* you go about limiting the amount of features your application has. First, Palm urges developers not to think in terms of trying to find ways to squeeze a desktop application's feature set onto a mobile device. Second, developers also shouldn't start with a desktop application's feature set and then try to remove features that might not make sense on a handheld. Instead, you should start from zero, and then add only those features that improve the user experience. "You don't pare down from 100 percent," Palm notes, "You start at zero and work your way up. Start with nothing, and add only essentials - one by one."

This leads to a second question - how do you determine what these essential features are? According to Palm, you identify the problems, find the simplest solutions to each problem, and get rid of everything else. In doing so, it's often best to limit yourself to using only proven and mature technologies, and not use technologies just because they're cool - only use them when they address a legitimate problem.

While you can ask your customers what they're looking for, Palm does note that they "do not always understand the limitations and consequences of the technologies they ask for. They don't realize that getting exactly what they ask for can be quite unpleasant." If you can't implement a feature in a satisfactory fashion, *don't implement it*. Or, try to find an alternate solution using mature technologies. In other words, don't think about what users say they want; think about ways to solve their underlying problems.

Palm also cites the well-known 80/20 rule - focus on what users do 80% of the time, and ignore the other 20%. This will, of course, harm fringe use cases, but it does allow you to focus on the things that matter to most of your users. “Pressing the Date Book button takes you to the current date,” Palm notes, “Why? Because 80 percent of the time, users want to see what they have scheduled for today.”

In keeping with the application-centric speed-focused design of Palm OS, it’s also advised to break your application up into smaller, more focussed applications wherever possible. Several Palm OS devices shipped with DataViz’ Documents To Go, which was a monolithic mobile office suite with a text processing, spreadsheet and presentation application all rolled into one. I personally never understood why DataViz didn’t just provide three separate, smaller, and faster applications, using the faux-multitasking mechanisms cited earlier to allow them to interact. Documents To Go was a clear case of not applying Zen of Palm.

Once you do have your feature set nailed down, your application’s UI should keep the amount of steps needed to perform common tasks to an absolute minimum, and lesser-used and power user-oriented features should be hidden in preferences and the application’s menu bar. Palm also urges developers to adhere to the user interface guidelines (hallelujah!) so that users can apply the knowledge gained from one application to the next. Lastly, you’re supposed to ‘delight’ the user, but Palm doesn’t really detail what this entails.

Palm’s Graffiti is actually a perfect example that encompasses almost all of the aspects of Zen of Palm. While the market was asking for full natural handwriting recognition (i.e., recognising any user’s own handwriting), this required massive amounts of processing power and RAM, and thus, a bigger battery - so the devices became bigger but were still slow as molasses, and generally, the recognition still sucked (I’m looking at you, Newton).

Palm understood that instead of “how to get natural handwriting recognition to work”, the real problem was “how to input text on a handheld”. To solve this problem, you really didn’t need natural handwriting recognition at all - a simple, single-stroke alphabet that was easy to learn was a far better solution, since it required far less processing power and RAM, which in turn meant better battery life and smaller devices. Since the recognition system only had to work with a small set of possible variations, it was also a lot faster and more accurate.

Despite the superior input method, Palm still focussed on tapping with the stylus as the main input method to use the device. The Newton, by contrast, included very cool features like recognising written-down times, dates and keywords to trigger the creation of appointments or to bring up contacts, but combined with the crappy handwriting recognition of the Newton, this was far less optimal than just tapping away with the stylus, and required more processing power to boot. By focussing on the underlying problem instead of the coolness factor, Palm arrived at a superior, if less flashy, solution.

I've only scratched the surface of the whole idea behind Zen of Palm - there's a lot more in there, and if you are a mobile application developer, you owe it to yourself to give it a read. It may be geared towards the dead Palm OS platform, but most - if not all - of its concepts are perfectly applicable on other platforms like Android, Windows Phone, and iOS.

Speaking of iOS, you have undoubtedly noticed that many of the ideas in Zen of Palm seem to have found their way onto the iPhone. I don't think this is a coincidence: I am convinced that the original designers of the iPhone and iOS were very much aware of Zen of Palm, and of the Palm OS in general. Reading through the iOS human interface guidelines, you'll encounter many of the same ideas, concepts, and approaches to application and operating system design.

Let me make it clear: *this is not a bad thing*. In fact, it's quite the opposite - Zen of Palm even specifically explains that studying your competitors' products is a good thing to do, because not only can you learn from what they do wrong, but also from what they do right. Palm OS/Pilot has a lot in common with iOS/iPhone not only in operation, approaches and design philosophies, but also in their respective roles in establishing a new category of devices and shaking up the entire industry.

I would have been more surprised if Apple's engineers *hadn't* taken a lot of cues from Palm OS, the Pilot, and Palm's philosophies. Palm was, after all, one of the most successful handheld companies of all time. Palm didn't create a market; they perfected it. The iPhone didn't create a market; it perfected it. The similarities are legion.

That being said, the differences are clear, too. Whereas Palm OS strived for user interface consistency, iOS never really cared about that. In addition, iOS' (in my view, childish and condescending) analog design (yes I will hammer that term into common parlance instead of "skeuomorphic") would never have passed Zen of Palm; Palm would have considered it useless fluff that would only worsen the user experience (Palm OS sits on the other end of the design spectrum: almost strictly digital).

Palm OS' legacy

So, what is Palm OS' legacy? What mark did it leave? How did it influence the industry?

Palm OS showed the industry what a mobile operating system for the average consumer should look like, how it should work, and what it should - and more importantly, *should not* be capable of. Consumers didn't want MS-DOS with a stylus input overlay. Consumers didn't want the confusing notebook metaphor GO and Apple used. Consumers didn't want a desktop operating system's interface shoehorned into a small screen. Consumers didn't want to have to deal with managing multitasking and the associated complexity. Consumers didn't want slow and bulky natural handwriting recognition.

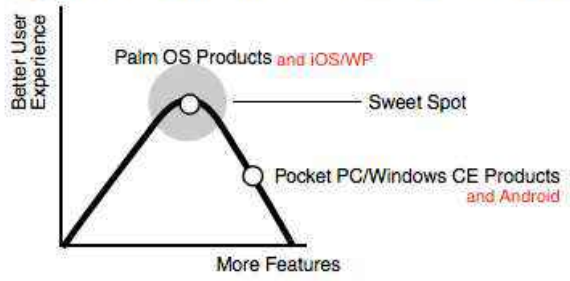
They wanted a minimalistic, single-tasking operating system that allowed them to focus on a single task, and do so fast, without having to wait for programs to load or go through endless confusing dialogs and setup screens. Users wanted an operating

system with a graphical user interface that was designed specifically with its primary input method in mind. They wanted an operating system that didn't require all the manual fiddling that the desktop operating systems of Palm OS' day required. They wanted an operating system that didn't drain the battery in a few hours. Users wanted an application-centric device.

Add all of these together, and 15 years ago, you got Palm OS.

Despite the improvements made to mobile hardware, today's mobile platforms still struggle with the same kind of trade-offs as Palm originally had to contend with, and they still make different choices. Remember that curve I showed you earlier, comparing Palm OS with Windows Mobile? I edited it a little to make it current.

Figure 1.4 Different views on new features



It's clear that iOS and Windows Phone are closer to Palm OS in stressing user experience over features, while Android is much closer to Windows Mobile in stressing features and versatility over user experience. Note, though, that this is not necessarily a bad thing - it's just a trade-off. While Android is often a bit messier and less polished, its large feature set and incredible versatility more than make up for it. And sure, iOS and Windows Phone don't have all the features and versatility of Android, but this does make them feel more polished. You decide for yourself what matters to you - and so far, people overwhelmingly choose flexibility and features.

Which is interesting, because they didn't use to, because Palm OS dominated the market. I already hinted at my personal preference for Windows Mobile when it comes to PDAs, and this may seem weird considering everything I've written so far. However, I preferred Windows Mobile exactly because of its versatility and many features; I took the suboptimal user interface and battery life for granted, because I mostly used them at home anyway (also, I cheated - I used Palm OS while on the go, and Windows Mobile at home).

I'm sad that Palm OS is dead. I've done a lot of studying and investigating for this article, and I've encountered so many cool things that have made me realise that Palm OS was far less outdated and inflexible than I originally thought. Did Palm OS really have to die? Was it really at the end of its wits at version 5.4.9? Or was there life in the old girl yet?

We'll get to that in a minute. First, I want to discuss a subject I have yet to touch upon: Palm OS and licensing, and what non-Palm hardware makers contributed to the platform.

Miscellaneous

I didn't really know where to put the final two sections of this article, so I decided to create a small miscellaneous chapter specifically for them. In this chapter, I'll first detail the importance of Palm OS licensing, after which I'll dive into the future of Palm OS, as seen from 2004 (which means Cobalt).

Licensing Palm OS

While often overlooked, you can't really write a detailed article about Palm and the Palm OS without diving into the many companies that licensed the Palm OS for use on their own devices. Sure, Palm itself commanded the vast majority of the Palm OS market, but the licensees played a crucial role in the evolving ecosystem: they became test beds for new features, technologies, and ideas.

From very early on in its existence, Palm intended to license Palm OS to third parties. Instead of scavenging around trying to scrounge up a licensee or two, like so many modern platforms have to do, Palm scored big right at the very beginning: five months after the original Palm Pilot started shipping, they scored IBM. The industry giant was going to sell rebranded Pilots, which eventually arrived on the market about a year later in September 1997 as the IBM Workpad 10u. IBM continued to rebadge Palm OS devices until 2001.

Vertical markets were also interested in Palm OS. And so, in March 1998, Symbol launched the SPT1500, a Palm III-like device with a larger forehead to accommodate a SE 900 Pico Scan Engine (apparently a revolutionary small barcode scanner). You could also get a version of the SPT1500 with wifi built-in (no joke - it had spectrum24). As far as I can tell, this makes it the first Palm device with wifi built-in.

From 1999 and onwards, there's an explosion in Palm OS licensees, many of which would play crucial roles in the on-going development of the platform and its supporting ecosystem. HandEra (formerly TRG) launched the TRGpro in October 1999, first to introduce a Compact Flash slot. In 2001, HandEra would launch the HandEra 330 - not only was this one of the first (not *the* first, though, but we'll get to that one in a second) devices with a 'high' resolution display (320×240 instead of the usual 160×160), it also introduced a feature that would become standard on all higher resolution Palm devices: a virtual Graffiti input area.

October 1999 also saw the introduction of the first ever Palm OS device with mobile phone technology built-in, making it one of the very first smartphones on the market: the Qualcomm pdQ. Hardware-wise it was identical to the Palm IIIe, but came with CDMA technology so you could call and even browse the web and send and receive email. Weirdly enough, the phone and Palm OS features were separated; the phone functionality ran its own firmware, and communication between that firmware and the Palm OS was limited. Later versions, after Qualcomm was acquired by Kyocera, improved this integration substantially.

Other than loads of Palm OS devices from smaller, lesser-known companies (my favourites: the AlphaSmart Dana and the Tapwave Zodiac), several big names would become Palm OS licensees: Nokia (never released a device as far as I know, but there's this), Samsung, Acer, Fossil (for smartwatches - currently all the rage again), Garmin, and Lenovo. As interesting as these were, they were small-time compared to the two big ones: Sony and Handspring.

Sony became a Palm OS licensee in 1999, and unlike some of the other licensees, it really tried to push the envelope for the Palm OS platform with its line of CLIÉ devices. While the first CLIÉ - the PEG-S300 - didn't exactly set the world on fire in 2000, it did have a unique design aesthetic that broke the mold a bit, and it had a more expensive version with a colour screen, too. This wasn't the first Palm OS device with a colour screen, but it was among *the* first.

After getting their feet wet with those first few Palm OS devices, Sony really started to drag the Palm OS platform forward, kicking and screaming. Released in March 2001, the CLIÉ N700C was the first Palm OS device to sport a "high" resolution display, and it did so in a way that didn't break application compatibility: it simply doubled all the pixels from the standard Palm OS resolution of 160×160 to 320×320 (all that is old is new again). It wasn't until November 2002 that Palm itself released a device with a 320×320 resolution, the Palm Tungsten T.

Even before Palm adopted the 320×320 resolution for its higher-end devices, Sony had already stepped up the game with what I think is the most iconic of all CLIÉ devices: the CLIÉ NR70. The NR70 introduced the characteristic flip-and-twist design with a full QWERTY keyboard; you could flip the screen open like on a clamshell phone, and you could then twist the display around and close it back up to arrive at a more conventional form factor. It sported a resolution of 320×480, although applications had to be designed to support it (Palm wouldn't adopt this resolution until two years later (!) with the slider Tungsten T3).

The NR70 had a more expensive brother: the NR70V. The V-model added yet another first for the Palm OS platform: an integrated digital camera with 0.1 megapixel. Sony also released the somewhat odd-looking PEGA-MSC1, a digital camera in the form of a Memory Stick which could shoot pictures at a maximum resolution of 320×240. You could use this with select CLIÉs with Memory Stick slots. Just to illustrate how far ahead of the curve Sony was: only two years later did Palm release its first device with a camera (the Zire 71).

I've always wanted a CLIÉ back when they were still current, but they were quite hard to come by where I lived. A few years ago I bought two CLIÉs from one of our French readers, one of which is the PEG-NX80V, one of the later flip-and-twist models. Its camera no longer works and the display has odd discolouring issues (I intend to open it up to see if I can fix it), but the whole form factor is quite pleasant. The keyboard is rubbish (impossible to properly type on), but twisting and turning the display is very satisfying. It's sad the mobile world has pretty much converged on boring slabs - I'd love Sony to release a modern, dual-screen interpretation of the flip-and-twist design.

Other than these specific firsts that Sony brought to the Palm OS ecosystem, a big focus of the company was multimedia - imaging, video, ATRAC and MP3 playback with remote control-equipped headphones (my NX80V has a stylus tip built into the remote!), and so on. This wasn't really a focus for Palm, but Sony showed them how it was done. In fact, Sony was so far ahead by this point that not Palm, but Sony was the first to launch a Palm OS 5 device (and, thus the first ARM device). Heck, Sony even went so far as to develop its own unique ARM processor for CLIÉ devices.

Before announcing the end of the CLIÉ line for European and North-American markets in June 2004, Sony had one last crazy trick up its sleeve for the world: the CLIÉ PEG-UX50. The UX50 (and its cheaper sibling, the UX40) was a really tiny Palm OS laptop with a twisting display. Pretty it was not, but it more than made up for it with its sheer awesomeness. I'd love to have one of these.

Japan got to enjoy the CLIÉ line for a year longer still, but in early 2005 the curtain closed in Japan, too. However, Sony just couldn't resist going nuts one more time, and in late 2004, it released a... Palm OS tablet: the PEG-VZ90. This crazy-looking contraption sported an OLED display, and has so much 'want' written all over it that it was only available in Japan. I'm fairly certain when I say none of us have ever seen one of these exotic beasts in real life (yet another reason to visit Japan).

And with that, the CLÉ line met its end. Sony had made significant contributions to the Palm OS ecosystem, and its focus on improving the multimedia aspects of the platform benefitted all Palm OS users. Their sales never even came close to that of Palm itself, but they left a definitive mark on the Palm industry.

Handspring is an entirely different animal than Sony or any of the other Palm OS licensees. The company has a very unique history in that it was founded by none other than Jeff Hawkins himself. In 1998, Hawkins and Donna Dubinsky had become unhappy with where 3Com - then owner of Palm - was taking the platform, since they realised that the PDA business wasn't going to stay around forever. "The handheld

computing business is going to go away," Hawkins said at the time, "It's going to become part of the cell phone business." Hawkins was always one step ahead of everyone else.

So, Hawkins and Dubinsky left Palm, and founded a new company called Handspring. They licensed the Palm OS right away, and one year later, September 1999, it released its first two PDAs: the Visor and Visor Deluxe. Handspring's first distinctive feature was the Springboard expansion slot; a proprietary slot that could house just about anything from memory cards to cameras, GPS modules, cellular modules, MP3 players, wifi, BlueTooth, barcode scanners, games, and much, much more. The importance of the Springboard slot's ability to house mobile phone technology is evidenced by the fact that all Visor PDAs were equipped with a microphone specifically for phone expansions.

The craziest accessory every to be made for Springboard? I doubt you want to know, so skip the next quote box if you don't.

> Raynet Technologies unveiled today the world's first body massager Springboard module for the Handspring Visor. Designed by Singapore Corporation, Raynet Technologies Pte Ltd, the Raycom Personal Massager converts the Handspring Visor handheld into a massaging device, while maintaining other computing operations.

It's even been reviewed. I didn't bother to read it, but here you go.

Handspring continued to build and sell the Visor line until it was discontinued in 2002, because the company had something far more special up its sleeve: the Handspring Treo. The Treo 180 was the first of the line, and came in a version with a QWERTY keyboard and a version with a Graffiti input area (the Treo 180g). Handspring released several more models before it was acquired... By Palm.

Following the Merger, Palm would continue to release several new and updated Treo models. They were quite popular - in fact, in the US, Palm enjoyed a smartphone market share in late 2005 of 31%, second only to RIM's 55%. As far as I can tell, the Palm Centro, released in late 2007, is the last Palm OS device Palm ever made - before switching to webOS. It ran Palm OS 5.4.9.

Handspring innovated both in and outside the Palm OS ecosystem with the versatile expansion slot and, more importantly, with the Treo. Thanks to the Treo and its success, Palm was given a crucial stay on the way to its demise, which gave it just enough time to develop the next chapter in the company's history - but that's a story that has already been told.

Where to go from here (if here is 2004): Palm OS 6 'Cobalt'

Before I let you go, there's one last chapter to this story before we wrap things up. This won't be a happy chapter, and it won't be about remarkable aspects of the Palm OS you didn't know about, nor will it be about cool technology from the '50s. With this last

chapter, I won't be able to strike the string of nostalgia either - because we're going to talk about something that has never been released.

For those of you who've seen Fucking Åmål (and for those of you who haven't: shame on you), remember that gut-wrenching scene that dragged on - intentionally - for what seemed like bloody hours where nobody showed up for Agnes' party? Where director Lukas Moodysson forced you to watch the naïve hopefulness of Agnes' mother as a stark contrast to Agnes' depressed anger?

Well, Cobalt is like Agnes. Nobody showed up for Cobalt's party either.

So far, I've been largely successful in ignoring the ownership situation of Palm OS, but for this last chapter I can no longer get around it. I'm not going into too many details, so here's the Cliff's Notes: Palm thought it was a great idea to create a wholly owned subsidiary dedicated to the development of Palm OS, called PalmSource, in 2002. They thought they had an even greater idea when they spun PalmSource off as an independent company in 2003. PalmSource was then acquired by ACCESS in September 2005.

Before being acquired by ACCESS, PalmSource worked on the future of Palm OS: Palm OS 6, better known as Cobalt. PalmSource took this matter quite seriously, and went for an almost ground-up rewrite; they stated around 80% of the code was rewritten. Cobalt was meant to bring the Palm OS into the 21st century, and included many features that had long eluded the platform. Sadly, since not a single licensee adopted Cobalt, information on it is patchy, at best.

First and foremost, just as Jim Schram had already predicted way back in 2002 when talking about the then-new MCK kernel in Palm OS 5, Cobalt finally exposed the system's inherent ability to multitask to application developers. It was a fully ARM-native 32bit operating system with multitasking and multithreading, so multiple applications and services could run at the same time. It had a background processing model designed to "reduce most memory problems commonly associated with multitasking in mobile devices".

Related to this, Cobalt also introduced protected memory, so that a crashing application would no longer be able to drag down the entire system. In addition, the maximum RAM and ROM limits were both raised to 256 MB, with the ability to raise this even higher in the future. Loads of security features (like industry-standard encryption stuff for enterprises) were added as well as core parts of the operating system, and it had a brand-new communications framework based on STREAMS, which supported multiple concurrent communication sessions.

A particularly interesting aspect of Cobalt was its new, extensible multimedia framework. As most of you will know all too well, Palm bought Be, Inc. and the BeOS in 2001 (yes, OSNews dates back that far!), and since BeOS is software, it ended up at PalmSource after it was spun off. Cobalt's multimedia framework and its "rich graphics and multimedia features" came from BeOS, and may even have been designed and

written by former Be engineers (I can't find confirmation that actual Be engineers worked on it, though).

Coded by our Be superheroes or not, the BeOS-derived framework brought Palm OS up to speed with the rest of the industry, and provided it with, among other things, rich graphics support with "paths, rotation, gradient fills, anti-aliasing, outline fonts and transparency", and supported screen sizes of up to 32000×32000 pixels (!). It also supported various audio and video formats, and developers and licensees (ha!) could easily add support for more.

Cobalt included an emulator so it could run applications written for Palm OS 5 and earlier. However, only "well-behaved" applications would run without modifications, while more complex ones would need modifications. Supposedly, Astraware's David Oakley 'ported' Bejeweled to Cobalt in less than half a day. Of course, developers could also write applications specifically for Cobalt, which were called Palm OS Protein applications. Interestingly enough, these could apparently be compiled for both ARM and... x86. I have no idea if ACCESS had plans for an x86 Palm OS.

On 10 February 2004, Cobalt was released unto the world - however, nobody, not even Palm, licensed it, preferring to ship Palm OS 5.4.9 instead. So, PalmSource went back to work and released Cobalt 6.1 in September 2004, which added an "enhanced" user interface (it looks exactly the same as before, just with slightly differently-designed buttons), and integrated a few things that were optional before (like BlueTooth and wifi). Sadly, nobody wanted Cobalt 6.1 either.

Since Cobalt never made it to market, there's not a whole lot to add here - I can't refer to reviews of devices, I can't talk about personal experiences, nothing. All we have is a preview by Brighthand's Ed Hardy, who notes, among other things, that yes, the BeOS-inspired multimedia framework could display a movie playing on all sides of a spinning cube. Which is very, very useful in real life computing scenarios both mobile and desktop, and don't you dare tell me otherwise.

It's worth noting that there is actually a freely available Palm OS Cobalt emulator that was provided by ACCESS. It's still available, and runs on Windows 8 just fine. It runs the full Cobalt operating system in a window, and you can change several settings like resolution, memory, and so on. It's pretty fast, but looks dated (despite the updated UI), and it misses several key applications (like the browser or an email client). Still, because you're controlling it with a mouse and there's little you can actually do with it, it can't possibly form the basis of something resembling a review.

And this is where Cobalt's story ended - or so I thought. I always thought that not a single Cobalt device was ever made - but during the long process of planning and writing this article, I stumbled upon an article by Bob Russell at mobileread, which confirms that Cobalt devices certainly did exist. The report is corroborated by David Beers, a Palm OS developer who actually handled one of the prototype devices.

A company called Oswin Technology, based in Singapore, designed and built two devices running Cobalt in 2005 that were offered for sale at the last PalmSource

DevCon before ACCESS acquired PalmSource. Russell even provides three full-page scans of the advertisements for the devices, which list all the specifications and two beautiful photographs of the devices. One was a candybar phone, the other a slider (?), but otherwise had nearly identical specifications. They sported a FreeScale i.MX21 processor, and 64MB RAM/128 MB ROM.

They aren't exactly beautiful or anything, but they are - as far as I know - the only complete Cobalt devices ever made. I wonder how many were sold at the time; I guess it depends on their price, and on just how many people were at that specific PalmSource DevCon. I wouldn't peg the number at much higher than a few dozen, making this one of the most exotic Palm OS devices ever made - perhaps the most exotic. Interestingly enough, it seems that one of the two devices did make its way to market as the Zircon Axia A108 - running Windows CE.

The elephant in the room right now is of course why, exactly, Cobalt failed to attract licensees. I think the answer to this question doesn't really have anything to do with Cobalt itself, but more with the storm that was brewing in Cupertino. By this point, all traditional Palm OS licensees had left the platform behind, except for Palm itself (and even they defiled their heritage by selling several Windows Mobile Treos), meaning that unless PalmSource could attract new licensees, it would have to rely on Palm.

However, I don't think it's far-fetched to believe that because several of Palm's employees had ties to Apple, they were aware - to a certain degree - of the work Apple was doing on the iPhone, and that Cobalt would not be competitive enough. In other words, no matter how much of an improvement Cobalt was compared to Palm OS 5.x, Palm didn't believe it was good enough to take on Apple's upcoming iPhone.

And from what I've seen of Cobalt, they were right.

While Cobalt's story ends here, ACCESS didn't stop trying to squeeze more blood from the dry, withered stone that is Palm OS. Aside from Palm OS 6.x, PalmSource/ACCESS had also developed Palm OS 5.5, which was designed to run in a dedicated virtual machine so other possible smartphone platforms could run Palm OS applications (it made its way to Nokia's internet tablets). Like Cobalt, nobody was really interested in this thing either.

However, this product, later renamed to the Garnet VM, was to play a crucial role in ACCESS' next attempt at gaining traction in the smartphone market: the ACCESS Linux Platform. As its name implies, ALP was a Linux environment based around Gtk+ and GNOME technologies, which could also run mobile Java applications. On top of that, Garnet VM was built-in to provide compatibility with applications for Palm OS 5.x and earlier. Its user interface was essentially a slightly more modern-looking Gtk+ recreation of the Palm OS graphical user interface (although the last version, from 2009, doesn't have much in common with Palm OS any more).

ACCESS failed to find any licensees for ALP. Its website stands as a 404-ridden tomb of broken dreams and unfulfilled aspirations.

As a sort-of conclusion to the Palm OS and Cobalt chapters, here's a short overview of the major Palm OS releases.

- Palm OS 1.0: 1996
- Palm OS 2.0: 1997
- Palm OS 3.0: 1998
- Palm OS 3.5: 2000 (a point release, but a major one)
- Palm OS 4.0: 2001
- Palm OS 5.0: 2002
- Palm OS 6.0: 2004
- Palm OS 6.1: 2004 (September - the final Palm OS release)

And that's all she wrote.

I'm ready to wallow now

And here we are. After almost 22000 words covering so many different aspects and such a large time period, it's virtually impossible to come up with a satisfying conclusion that would touch upon all the topics discussed. As such, I'm not even going to try; I tried to give every chapter its own conclusion. Instead, my final, closing words will be of a personal nature, and will probably not sit well with everyone.

Palm showed the world exactly what consumers wanted out of a mobile device. It did this not by trying to put a desktop computer into a smaller device, or by piling on the features and specifications, but by actually trying to understand the problems users were facing. The company's relentless focus on speed, efficiency, and price, combined with the concept of Zen of Palm, led it to a mobile platform that serves as the archetype for all the popular mobile platforms we have today.

And yet, in 2013, Palm no longer exists. While conventional wisdom has it that Palm died when webOS failed to catch on, it is my view that Palm was already long dead before webOS ever even arrived on the scene. The company doomed itself the moment it decided to spin off its software division, tearing apart the hardware and software integration that Jeff Hawkins knew was crucial to creating a compelling handheld product, after having experienced first-hand the disaster that was the Zoomer. That faithful day in the company boardroom, when they all agreed to spin off the Palm OS, Palm sealed its fate.

From that decision onwards, Palm continued its long slide into irrelevance. If it hadn't been for the Handspring acquisition and that company's forward-looking Treo, Palm would have died earlier. The Treo gave the company a stay, but the bad decisions just kept on coming. Without control over their own operating system, Palm could no longer add the features it needed in a properly integrated way, nor could it modernise the platform in a way that made sense for them. The situation got so dire that the company had to sink to extraordinary lows when it decided to sell devices with Windows Mobile. The Windows Mobile Treos were a slap in the face of the company's own heritage, its fans, and the ideals it had stood for since its very inception.

So far, I've barely touched upon webOS, and this may have surprised some. The reason for that is simple: webOS failed to impress me. I didn't get to try webOS when it was current, since the platform never made its way to The Netherlands. So, it wasn't until last year that I finally got my hands on a Palm Pre 2, which one of our readers loaned to me.

I was incredibly excited. I expected a quick and snappy modern Linux-based operating system that had Zen of Palm written all over it, running on the kind of well-built, quality hardware that I had come to expect from Palm thanks to my Tungsten E2 and T|X. Instead, I got something that had absolutely nothing to do with everything I loved about Palm. WebOS turned out to be a battery-sucking, unbearably slow monstrosity of an operating system running on low-quality hardware.

Sure, people will talk about how I should have used a Pre 3 or TouchPad, without realising that's *exactly* why webOS is not a true Palm product: Palm never hid itself behind specifications. Palm never had to say "sure, this sucks, but just you wait until our *next* device, when our hardware has finally caught up with our features!" That's not how Palm is supposed to develop products. That's not Zen of Palm. I don't care if the Pre 3's or 4's or 5's hardware specifications finally made webOS bearable - as a Palm user, I never had to care. Why should I start now?

WebOS plummeted Palm in the spiral of doom.

The entire operating system reeks of limited resources and shortcuts, so many shortcuts. Instead of a custom, specifically adapted kernel, designed for speed, they took the Linux kernel without really having the manpower to really adapt it to their needs. Instead of creating a proper API and UI layer specifically designed for handheld devices, true to the Palm way, time constraints forced the company to settle for WebKit, using HTML and JavaScript to create applications. And so on.

We all know why they had to take these shortcuts: they spun off their operating systems division, and all the knowledge, expertise, and manpower that came with it. All this was now wrapped up in a company they no longer controlled. Thanks to a deal with ACCESS, Palm only had the code to Palm OS 5.x to work with - not the much more advanced and more modern Cobalt.

In my completely and utterly speculative view, Palm would have been better off if they had used Cobalt as a base for their next-generation mobile operating system. Cobalt, as it existed at the time, was not ready to take on iOS or the then-nascent Android operating system, but had Palm combined the lightweight, Cobalt core, capable of all the things we ask of modern systems today, with the innovative multitasking-oriented card user interface Matias Duarte had designed, they'd not only have had a fast and modern operating system, they'd have had it far, far earlier.

My original plans for this article - made before the Pre 2 arrived on my doorstep - were much, much different from what you're about to finish reading. The Palm OS and Pilot parts were always going to be in there, but the main body of the article would be made up of an intricate look at webOS, a celebration of a fantastically innovative company's last great hurrah, its final encore before the curtains closed.

After actually using the webOS and testing it for weeks on end - insofar I could, as the battery would only last for four-to-five hours before running out - it dawned on me I just couldn't do it. This product doesn't deserve a celebratory eulogy. You can't write an article about how great an operating system is - and I've been around the block when it comes to operating systems - just because it has a cool card UI. A cool UI doesn't hide the fact that it's slow and unresponsive. A cool UI doesn't hide the fact that the underlying system is unoptimised. A cool UI doesn't hide the fact that it sucks battery like a there's no tomorrow. A cool UI doesn't hide the fact that the hardware was of appallingly low quality.

A cool UI doesn't hide the fact that the operating system has absolutely nothing to do with what Palm is supposed to stand for.

So, I threw all my original plans in the bin, and started from scratch. I was going to write a celebration of Palm, its innovations, its unique approach to hardware and software design, and its defining role in the mobile computing industry. After weeks and weeks of work and months of planning, I hope I succeeded in doing just that. And, I hope you enjoyed it.

Palm, thanks for the fantastic products.

Me, I'm ready to wallow now.