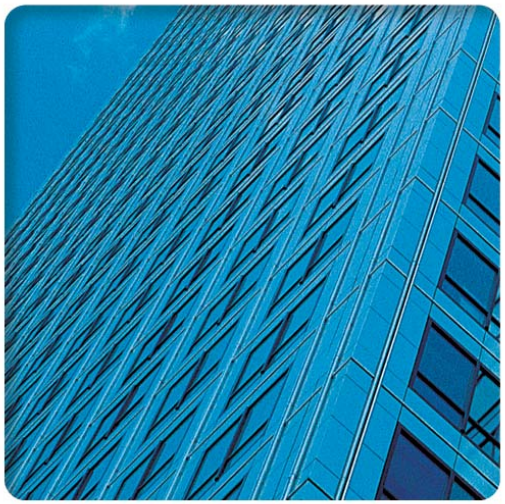


Good API Design
And why it matters

(a shameless rip-off of Bloch's OT2004 Keynote:-)

John Pagonis
Symbian Developer Network

…from little seeds

We are satisfied by doing real work, Software is like a plant that grows:

You can't predict its exact shape or how big it will grow; you can control its growth only to a limited degree.

There are no rules for this kind of thing – it's never done before.

-- Charlie Anderson, Architect, Borland Quattro Pro for Windows

Based on true stories

- This presentation is based on true stories
- The names have been changed to protect the guilty
- A lot of it has been re-used (lifted :-) from Joshua Bloch's OT2004 keynote. I take responsibility for any additions or inaccuracies :-)
- I work for the Symbian Developer Network, where our job is to:
  - … *Help developers, develop on* Symbian OS
  - … We do it for free, therefore people talk to us
  - … Their truth is many times brutal

Why is API Design Important?

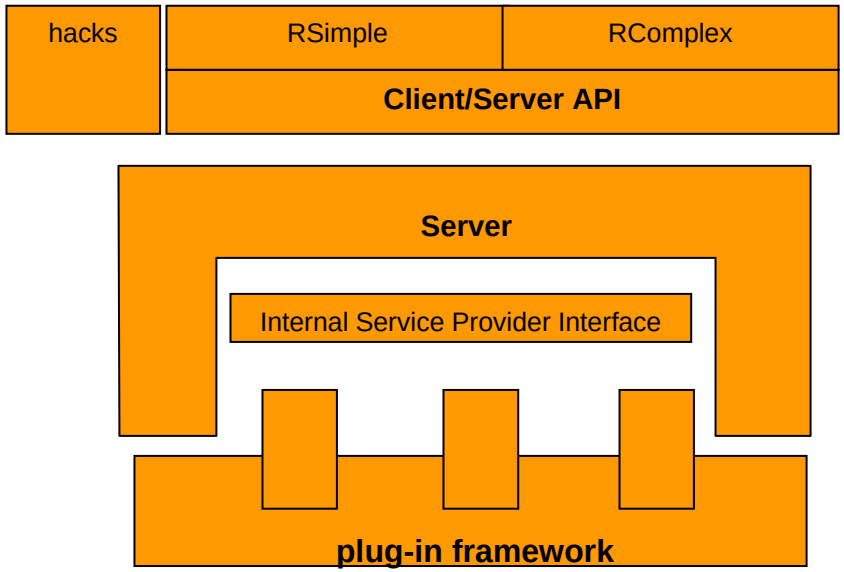
- APIs can be among a company's greatest assets
  - … Customers invest heavily: learning, writing, buying
  - … Cost to stop using an API can be prohibitive
  - … Successful public (and partner) APIs capture customers
- Can also be among a company's greatest liabilities
  - … Bad APIs result in unending streams of support
  - … Bad APIs send developers away from a platform
  - … Bad APIs make developers build _bad_ code
  - … Bad APIs are not fun!!!
- Public APIs are FOREVER, one chance to get it right

Why is API Design Vital to Symbian

- APIs and SPIs are part of what we offer, while others implement their plug-ins
- We offer a PLATFORM, people need to be able to (re-)use it! ……..Hellloooooooooo
- Increasingly ‘’all we do’’ is to define APIs
- We are also brokers and mediators for competitors who bring their ‘’functionality’’ to a common base
- If anything, the APIs on which others build their products on, is our PRODUCT.

- Gates and Ballmer were right ...and I hate it when Bill is right

How many times have you seen this?



We define so many APIs for others to implement, or is it just me?

Why is API Design Important to You?

- If you program you are an API designer
  - … Good code is modular – each module has an API
- Useful modules tend to get reused
  - … Once a module has users, you can't change API at will
  - … Good reusable modules are corporate assets
- Thinking in terms of APIs improves quality
  - … Because if you think of usage you think of testing
  - … Testing is good! Reuse is good!
  - … If it is difficult to test then most likely it is difficult to (re-)use
  - … Therefore you won't and you will not find the bugs!

Characteristics of a Good API

- Easy to learn (modulo domain specific expertise:-)
- Easy to use, even without documentation
- Hard to misuse – Very important!
- Easy to read and maintain code that uses it
- Sufficiently powerful to satisfy requirements
- Easy to extend – think BC :-)
- Appropriate to audience

The Process of API Design

- Gather requirements – with a healthy degree of scepticism from its potential users
- Start with a short spec! - primitives
- Write to your API early and often
- Writing to SPI is even more Important
- Maintain realistic expectations

Gather Requirements

- With a healthy degree of scepticism
- Not from a committee but from the real potential users!
- Your job is to extract true requirements from usage scenarios - reach out and find them
- Source the requirements by listening and observing what people do and try to do! – hint: newsgroups
- Do not listen to everyone, but ask for reviews
  - … If it is small you can extend it later
  - … But if it is ugly and complicated you'll need to support it

Start with a short spec – 1 page is ideal

- At this stage, agility trumps completeness
- Bounce spec off as many people as possible
  - Listen to their input and take it seriously
  - Remember ‘’Egoless Programming’’ (Weinberg)
- If you keep the spec short, it’s easy to modify
- Flesh it out as you gain confidence
  - This necessarily involves **CODING** !!!
  - ‘’Architect also implements’’ organisational pattern (Coplien)
- Start by writing down ‘’primitives’’ in text!
  - Responsibilities
  - Clients
  - Providers

Write to your API Early and Often

- Start **before** you've implemented the API
  - … Saves you throwing away many implementations
- Start **before** you've even specified it properly
  - … Saves you from writing specs that you will throw away
  - … Plan to throw one away anyway (Brookes)
- Continue writing to API as you flesh it out
  - … Prevents nasty surprises at release time
  - … Your usage code will live as examples and unit tests
- ''Engage QA'' organisational pattern (Coplien)
  - … Test engineers are the best to get in for review first !

Psisoft/Symbian used to do this because we used to write apps!! Therefore we used to refine our APIs and eat our own dog food early and often.

Writing to SPI is Even More Important

- Service Provider Interface
  - … Interface supporting multiple implement(ors/ations)
  - … Example: all those interfaces to CSYs, TSYs, FSYs, PRTs, PSYs, NIF/NAFs..guffs etc
- Write multiple implementations before release
  - … If you write one, it probably won't really support another
  - … If you write two, it will support more with difficulty
  - … If you write three, it will work fine
- Will Tracz calls this ''The Rule of Threes''

(Confessions of a Used Program Salesman, 1995)

Maintain Realistic Expectations

- Many API designs are over-constrained
  - … You won't be able to please everyone
  - … Aim to displease everyone equally
- Many API designs are over-engineered
  - … And not used :-(
- Expect to make mistakes
  - … A few years of real-world use will flush them out :-)
  - … Expect to evolve your APIs
  - … Don't change, extend and deprecate!

General Principles

- An API should do one thing and do it well
- Your APIs should be as small as possible, but no smaller
- Implementation(s) should not Impact the API
- Minimise accessibility of everything
- Names matter – every API is a little language
- Documentation matters
- Consider performance consequences of API design decisions
- APIs must coexist peacefully within the platform

An API Should Do One Thing and Do It Well

- Functionality should be easy to explain
  - … If it is hard to name, that's a bad sign
  - … Good names drive development
  - … Be amenable to splitting and merging modules
  - … Forget about the ''in case someone may..'' cases and concentrate on known use cases
  - … If you have conflicting use cases then possibly you need two modules

Your APIs Should Be As Small As Possible But No Smaller

- An API has to satisfy its requirements
- But, when in doubt, leave it out
  - … Functionality, classes, methods, parameters, etc
  - … You can always add, but you can never remove (something like ‘’Hotel California’’ kind of paradigm :-)
- Conceptual weight is more important than bulk
- Look for a good power-to-weight ratio

Implementation(s) Should Not Impact the API

- Implementation details
  - … Confuse users
  - … Inhibit freedom to change the implementation
  - … Usually ruin BC,SC and even DC
- Be aware of what is an implementation detail
  - … Do not over-specify the behaviour of methods
  - … Do not return big ugly complicated structs!
  - … Use the ‘’Null Object’’ pattern (Bruce Anderson) don’t return NULL if it isn’t natural (i.e. User::Alloc)
- Don’t let implementation details ‘’leak’’ into APIs
  - … On-disk and on-the-wire formats

Minimise Accessibility of Everything

- Make classes and members as private as possible
- Do not make them private though, because you are afraid someone ‘’may do something bad’’ with them!!! We have Platform Security for that!
- Maximise information hiding, so that you can change implementations easily, later
- Hide the private details in C++ headers by using idioms that help BC (that’s another seminar :-)

Names Matter – Every API is a Little Language

- Names should be largely self-explanatory
  - … Avoid cryptic abbreviations
  - … Use specific names for classes, but generic names for base classes
- Be consistent; same word should mean the same thing
  - … Throughout an API
  - … Across APIs on the platform
- Use correct parts of speech and your code will read like prose!

Documentation Matters

- Document every public class, interface, method, parameter and exception
  - Class: what an instance represents, intended use, intended clients, derivation intentions
  - Method: contract between method and its client, pre-conditions, post-conditions, side effects
  - Parameters: indicate ownership, units, show type
- For every internal class and method that needs it
  - Do not document how it does something (unless it is something really esoteric or smart), we have the source
  - Document the ‘’whys’’, the ‘’whos’’ and the ‘’intentions’’, the source and naming should tell the story
  - Good engineers go to the source (Coplien), documentation should show usage

Consider Performance Consequences of API Design Decisions

- Bad decisions can limit performance
  - … Re-use sessions, don’t create them on each call
  - … If possible pre-alloc memory so that you don’t need to trap
  - … Let resources find you (Taligent), don’t create them temporarily
  - … Don’t return temporarily (copy) constructed objects
- Effects of API design decisions on performance are real and permanent

APIs must Coexist Peacefully Within the Platform

- Do what is customary
  - … Obey standard naming convention
  - … Obey standard platform paradigms and idioms
  - … Mimic patterns in core APIs and language
- Take advantage of API-friendly features
  - … Use const descriptor references, pass around typed objects not plain TInts
- Know and avoid API traps and pitfalls
  - … e.g, `TInt DoSomethingL(TSomething& aType, CStuff* aObj)`
  - … `(notable exception is OfferKeyEventL here :-)`

…from little seeds

We are satisfied by doing real work, Software is like a plant that grows:

You can't predict its exact shape or how big it will grow; you can control its growth only to a limited degree.

There are no rules for this kind of thing – it's never done before.

-- Charlie Anderson, Architect, Borland Quattro Pro for Windows

References

- ‘’Taligent’s Guide to Designing Programs’’, Taligent Inc
- ‘’The Psychology of Computer Programming’’, Gerald M. Weinberg
- ‘’The Mythical Man-Month’’, Frederick P. Brookes Jr
- ‘’Organisational Patterns of Agile Software Development’’, James O. Coplien, Neil B. Harrison