I-TECHNOLOGY INC

Eventz Omnibus

October 2019
V-1.0

Self publishing

Document Revision History

Version	Date	Description of Change	Person Responsible
1.0	Oct 31, 2019	Initial Release	Steve Jackson

Table of Contents

Introduction to the Eventz Omnibus

This document is an Omnibus collection of topics that describe how the Eventz software development methodology works. The reader will probably be drawn to some but not all of these topics. After the Introduction, the chapters are standalone and can be read in any order.

Eventz is an open source Event-Driven methodology for the development of modular transaction processing software. It is a Disruptive Technology, because it is a radically new approach to software. Its hallmark is simplicity, and we really need that in software!

Eventz is a powerful methodology because it guides the business stakeholders to build solid computing solutions, always following the self-same process. The basic concept is that transactional computing systems can be created solely by:

1. **Data** in the form of immutable tuple Event Records (ER) composed of fields describing all relevant details to journalize each Event.
2. **Logic** in the form of Functions (Microservices) which create, display or report on these Event Records.

So we, with stakeholders, work to identify the activities , actions an events in the business space, and to create the necessary fields to describe each event fully as an event record (ER). The immutable Archive of these ERs is the sole form of data persistence used in Eventz. And the Functions (implemented with microservices) that create and publish the ERs are the sole form of program logic.

Microservice architectures have great promise for developing and deploying solutions that are loosely coupled and highly coherent. They also can be daunting to implement and deploy. Eventz is a methodology that provides an infrastructure that makes this process highly accessible.

Eventz has just a few components:

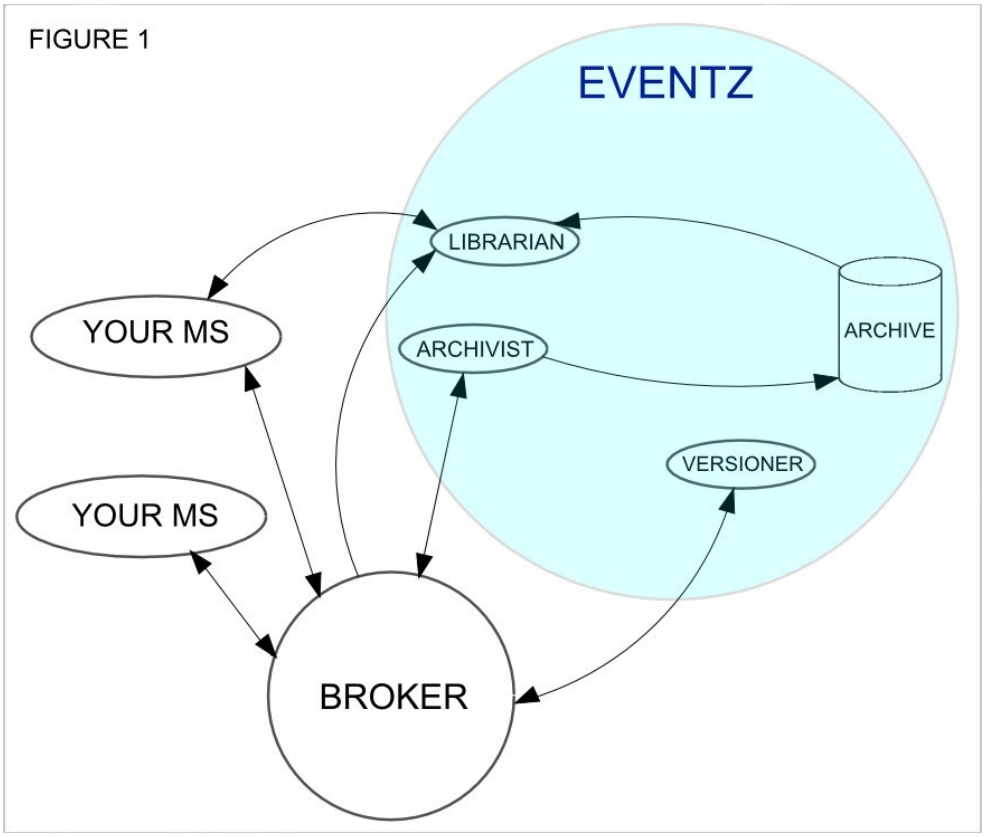
- EventzAPI is a python library that provides objects and methods that

facilitate AMQP messaging with the microservices.

• A RabbitMQ server that implements a publish-and-subscribe broker.

• Services in Docker containers that archive and allow querying an indelible archive of published ERs.

Figure 1 below shows an overview of the components and how they interact.



Every application has multiple activities, which are the policies and processes of the business. These activities generate Events. It is these Event Records (ER) that are published and subscribed to by the Function microservices. Pub-Sub of ERs constitutes the sole means of communication between microservices.

One or more Archivist services subscribe to all events and save the ERs in (Append-only, WORM, Indelible, Unassailable, Permanent and Eternal) data Archives. Librarian services can query an Archive and return a result set of the

selected ERs. This provides the sole data persistence to the Functions.
For faster response, each Function microservice has the option of hosting a local Archive containing just the subset of ERs that it subscribes to.
ERs are published and stored in a globally pre-defined format. The ER begins with a record type identifier, followed by appropriate signature metadata, and then the payload of data generated by the Event. The message is an ASCII encoded tuple, which can be encrypted if necessary.
A range of record types (900000000.00 to 999999999.99) is reserved for system messages, which provide a way to report errors, ping services etc. System messages are handled by the API.

This Omnibus is organized into Chapters. You can use the ***control-click*** feature of the Table of Contents dotted lines to jump to the discussion topics of greatest interest.
Some Conclusions:

> Can computing be so simple as Eventz proposes? Many readers will dismiss the Eventz scheme as simplistic and severely limited in its reach. We agree that Eventz is simple, but we have probed to find its transactional application ceiling using our reference applications, and we have yet to find it! Just blue sky.
>
> Our first reference application was a full-featured MRP system for electronics manufacturing, complete with purchasing and production scheduling support.
>
> Our second reference application was a pay reconciliation program for Phoenix payroll.
> https://fixphoenix.ca/

What are a few of the unique benefits of the Eventz methodology?

1. It is truly Agile, as intended by the 2001 Agile committee members, easily and without contortions. Engineers will love Eventz.
2. You can easily find the staff required, despite the serious shortage of technical talent.
3. Disaster Recovery is easily achieved and verified.
4. Tight security can be engineered into your subnet.
5. It breaks down the barriers between business and programming, like no other methodology claims to do.
6. It has no onerous Requirements Specification process up front. Eventz is EZ Event-Driven.
7. It separates Data from Information – a truly delightful separation of concerns.
8. Report functionality can be deferred to a quiet afternoon down the road, and does not need to be baked into the initial system design.
9. Late-arriving change orders are not disruptive to the core design.
10. It is simple, with a short learning curve! Not Rocket Science at all.

Contact Information

Further details are available at Eventzapi.com and info@i-technologyinc.com

Chapter 1 Theory

1.1 Eventz Infrastructure Schema

The Eventz paradigm envisions stakeholders identifying activities that generate event records. These activities are modeled with microservices that communicate with each other by publishing and subscribing to ERs. Data persistence is provided by one or more immutable archive(s) created and maintained by Archivist service(s) and data sourced through Librarian service(s).

Event records begin with a record type which is a floating point number unique to that record type. The record type is assigned by the developer as part of the initial specification of the ER. If record definitions are modified after distribution they are given a new version number. The version number is the fractional part of the floating point number.

Following the record type is a number of fields which constitute record metadata. The metadata will identify the originating application, it’s location and the device it’s running on, its user as well as the time of publication. In addition there are 5 metadata fields for use by the application developer. The Librarian is constrained to provide results based on indexes to the metadata fields only.Security

The Rabbit MQ server supports message encryption and authentication[1].
The Organization is responsible for providing and controlling certificates. The indelible nature of the archive and the metadata in the records facilitates forensic audit in the case of a breach. Restricting interactions to AMQP messaging minimizes the threat of attack propagation.

RabbitMQ Server can be anywhere on the Internet

Each Function must know the IP address of its RMQ server in order to establish p & s connections. This is provided typically with yaml files that are local and

1 https://www.rabbitmq.com/ssl.html

provide the information required to each service.

Archives can be anywhere

To minimize latency, it is desirable to have an Archive on the client's premises. There can and should be several other Archives as backups. The accuracy of the Archive is mission critical! There must be no missing ERs, and all Archives must be certifiably identical by content (the order of the ERs can be different). Each archive must be accessible to an Archivist and Librarian running on the same machine.

Message Latency

Where latency is a problem, it will be desirable to install all the parties (RMQ, Broker, Archive/Librarian, Functions) on the same LAN.

1.2 Publish and Subscribe in Eventz

Eventz uses RabbitMQ for the pub.sub messaging. RabbitMQ is well featured and very mature.

Kafka is an alternative to RabbitMQ. This paper examines the different applications for each: ***Understanding When to use RabbitMQ or Apache Kafka*** https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka

Publish ***y*** and Subscribe to ***x***

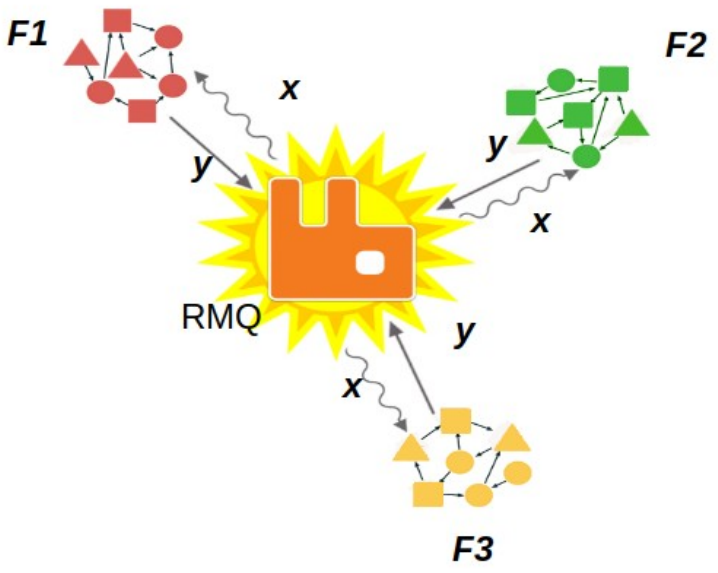
$$y = F(x)$$

Where:

Fx is a Function or Microservice

x is the set of event records subscribed to by the Function

y is the set of event records published by the Function



1.3 Delightful Separation of Concerns in Eventz

Computing is complicated!

Eventz brings a simplification, because we simply record events as new Event Records. Once this event data is published and in the Archive, any Function can use it for any purpose.

So the acquisition of the Data is separated from the use of the Data.

This is very helpful in avoiding complexity!

1.4 Citizen Developers for Eventz

"Citizen Developers[2]" can remedy the serious shortage of programmer talent. After the microservice Functions are well specified in terms of the Event Records and the logic that is needed, the work of creating the Functions can be subcontracted out. Only Functions that pass specified tests will be accepted and paid for. Upwork.com[3] and Guru.com[4] developers can be hired to accelerate progress.

The ability to outsource or subcontract the programming of the Functions can speed up an Eventz project by using external resources, in parallel. This resolves Brooks' Law: "Additional programmers will only make a late project later."

Python is attractive for citizen developers because it has a low entry threshold. The programming skills required to write Eventz Functions are low – always basic calculations involving the fields of multiple ERs. The difference between an elegant Function and a pedestrian Function may be negligible. When a Function publishes the desired ERs for all test conditions, it is finished. The EventzAPI assists greatly with the heavy lifting.

2 https://www.techopedia.com/definition/30968/citizen-developer

3 https://www.upwork.com

4 https://www.guru.com

1.5 Eventz for Internet of Things

Eventz supports and embraces the Internet of Things. Functions can be created for the purpose of formatting IoT messages into Event Records to be published to the Archive. Then any other Function in the system can use this data to create new ERs. For example, IoT temperature readings may be published as ERs, and then used to control the HVAC system zones based on time of day and occupancy, etc.

Eventz ERs can be stored today for some known (or as yet unknown) purpose in the future. We can save data that may be of interest in the computing space, since the marginal cost is negligible.

Python is supported on many platforms and many OS’s.

1.6 Eventz Eventual Consistency

Introduction

Under the Eventz paradigm, Event Records (ER) are recorded in an archive by an Archiver Function. The Archiver subscribes to all ERs published to an exchange. The Archiver appends these ERs on a first come first serve basis to the archive. The archive is a write once store such that the data in it is indelible. The archive is read by a Librarian Function that responds to queries. Applications also subscribe to these records and in addition, may query the archive to get one or more instances of the record type. The application may keep this information in a local store which may be a list in memory or a local database. The Eventz paradigm claims “eventual consistency” for the data that the microservices operate on.

“Eventual consistency is a consistency model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.” [5]

Record Issues

The following concerns arise when relying on data in remote or local stores.

Order

Every record published and archived is prepended with metadata that includes the publication date and time. A result set can be sorted on this date to preserve order. This implies that all Function microservices operate from the same time base (UTC) and local clocks are synchronized across the enterprise. A better approach would be to have the publication time be changed to archived time with the archiver inserting its time in the field.

5 https://en.wikipedia.org/wiki/Eventual_consistency,
Vogels, W.(2009). "Eventually Consistent". Communications of the ACM. 52: 40. doi:10.1145/1435417.1435432.

Sequence

Records are provided with a unique record identifier (UUID). When a record is to be modified a new record is written with a link to the original record, and the new record is tagged as an update. Similarly when a record is deleted a new record is written, linked to the last update or the original if there is no update, and tagged as a delete. Through the links we can determine the last state of the record as well as, if we need it, its history. When an application queries the archive it will obtain a result set with all states of the record. A utility in the API can reconcile the result set to contain only the latest state for each record.

Accuracy

An issue arises when more than one microservice acquires a record in the same state and each operate on that record. When one publishes its update followed by the other we will have two update records referencing the original. When this occurs it is necessary to merge the two updates. When fields conflict the latest one should be accepted. This holds up except for fields that are the result of calculations, like running balance fields. These types of fields should be avoided. Instead the delta value should be stored Running balances should be calculated when needed. This may entail accumulating all the transactions and performing the calculation from first record. If a closing balance can be stored at a convenient milestone (Year end, Month end etc.) then the calculation can start from that time. Another approach is to store the new balance in the transaction record while implementing the discipline that no transaction record will be updated or deleted. To modify a transaction value, new transaction records need to be published, using current dates, with the adjustment. This approach is used by many accounting systems

One strategy is to delay a few seconds to ensure that any potential conflicting ERs have arrived.

Embrace Reality of eventual consistency. Jonas Boner

https://www.youtube.com/watch?v=1hwuWmMNT4c

1.7 Eventz Security and Attack Surfaces

Multiple Archives for safety. You get to hold your data!

The Immutable Tuples that are Event Records (ER) can be stored in multiple Archives, ideally far apart on different continents. This data remains the property of the client, and can be put to any use that may arise in the future. The Archives are WORM (write once read many) files and they are kept safe against ransomware attacks, etc. They can be encrypted as desired. Each ER describes an event that is “of interest” in the transaction space.

1.8 Eventz is Functional Programming

Functional Programming is often seen as an Academic pursuit, rather than a practical coding method. However, Eventz uses function **F** in the defining relationship:

$$\mathbf{y_n = F_x(Y, e_n)}$$

where:

- $\mathbf{e_n}$ is some Event, like a new message, a key-press, or a timeout, etc.
- $\mathbf{y_n}$ is an Event Record, an immutable tuple that captures the event, e_n.
- $\mathbf{Y = \{y_0, y_1, \ldots, y_{n-1}\}}$ is the set of every tuple ever published, the Archive
- $\mathbf{F_x}$ is the Function (program) that turns event e into its Event Record. The arguments are the Archive Y and the new event.

Here there are no assignment statements, and no side effects.

This equation means that the only data persistence in Eventz is the Archive of Event Records. And it means that the only logic is the collection of Functions F_x . So instead of the usual central database, Eventz uses only the immutable ERs in its Archive. This greatly simplifies Disaster Recovery, since the data is immutable (write once, read many).

Because all the system logic is housed in **Functions**, which use Arguments that are previous Event Records, plus details of the current Event, the programmer has a clear and straightforward assignment. The problem has been precisely specified, so that the programming solution is evident.

It seems that Eventz will work in every transactional application space.

"Excel is the world's most popular functional language". Simon Peyton-Jones

In functional programming, programs are executed by evaluating expressions, in contrast with imperative programming where programs are composed of statements which change global state when executed. Functional programming

typically avoids using mutable state.
https://wiki.haskell.org/Functional_programming

“OO makes code understandable by encapsulating moving parts.
Functional Programming makes code understandable by minimizing moving parts”. *Michael Feathers Nov 3, 2010*

1.9 Eventz is Agile!

The Eventz programming method uses Event-Driven computing with Python. There are some major benefits in this. The basic concept is that computing systems can be created solely by:

1. **Data** in the form of Event Records (which are immutable tuples) which capture every relevant field of information to describe some event.
2. **Logic** in the form of Microservices (or Functions) which create, display or report on these Event Records.

Introduction

The **Agile Manifesto** was published in 2001 with the goal of improving the software development process. Today Scrum Agile is the most popular variant, but it is not working very well for many companies. Many of the 17 signers of the Manifesto are not supporters of Scrum Agile today. See Appendix 1 for their comments. For example, Andy Hunt wrote “The Failure of Agile” in 2015. https://toolshed.com/2015/05/the-failure-of-agile.html

Stefan Wolpers lists five common complaints of software engineers in ***Why Engineers Despise Agile***:

- Control
- Manipulation
- Monitoring
- Technology
- Teamwork

“Decisions are still business-driven, made by people without an understanding of technology. That includes in most cases the product owner as well as the middle management, or business analysts.” We will revisit this quote below.

https://www.business2community.com/product-management/engineers-despise-agile-01490586

Many people assume that the only Agile process is Scrum, but the list of processes has grown much longer, including:

- Lean
- Kanban
- Feature Driven Development (FDD)
- Extreme Programming (XP)
- Crystal
- Dynamic Systems Development Methodology (DSDM)

In fact, Scrum predated Agile, since Ken Schwaber and Jeff Sutherland had been using Scrum for ten years prior to the 2001 meeting at Snowbird where they and others signed the Agile Manifesto.
https://www.scrum.org/about

Some reasons why Sprints fail:

- Poorly written user stories – The user stories were incomplete and lacked substance in the acceptance criteria.
- No Spikes for complex stories due to time-crunch – Team was over-optimistic in estimation and pulled a story in Sprint without doing any research in the area that was unknown to them.
- Having a team of mediocre people or lack of cross-function members in the team
- Communication barriers – especially with the remote/distributed teams and bureaucracy
- Development manager acting as Scrum Master

http://www.agilebuddha.com/agile/5-whys-sprint-failed-team-did-not-deliver-committed-work/

Where have objects gone wrong?

- Most programmers program in classes
- Classes are static; we sell use cases
- We no longer can understand what our programs do!
- The GUI comes later or is separated in a client/server architecture – so users don't understand what our programs do, either
- --- And use cases aren't designed, so users can't understand them from the interface, either (nor can the coders understand them from the code.)

OO has lost its roots

- There is little reasoning and few methods that focus on "networks of cooperating objects" (DCI is an exception)
- Microservices could be kind of right except everyone is looking at the wrong end
- Focus on unit testing and object integrity has replaced "operational models" and the human focus
- Patterns, which could have opened this up, have become pre-packaged technical solutions

Where has Agile gone wrong?

- People think it means "fast"
- Social design with CRC cards has been replaced by individual or pairwise testing
- Very little about the user mental model
- Processes and tools (e.g., Jira, testing) over individuals and interactions: it's about programming
- Certification over customer collaboration (how many UX people here?)

https://www.youtube.com/watch?v=ZrBQmIDdls4

`goto 2017 James Coplien`

The phrase “serverless” doesn’t mean servers are no longer involved. It simply means that developers no longer have to think that much about them.
http://readwrite.com/2012/10/15/why-the-future-of-software-and-apps-is-serverless/

https://www.youtube.com/watch?v=Y6B3Eqlj9Fw

Bilgin Ibryam: finally, a good explanation of the poorly named serverless concept by @jeffhollan: Functions is a programming model, but serverless is a billing model
https://twitter.com/bibryam/status/1007284710136000513

“Developers turn caffeine into abstractions” - Brian Marick

Stefan Wolpers (https://age-of-product.com/author/stefan/) has published a long list of Agile Anti-patterns, and in Appendix 2 we show how ***Events Agile*** potentially resolves them all!
This could breathe new life into Agile.
Events Agile is a novel software methodology that is well adapted for transaction processing systems. The basic concept is that software applications can be entirely event-driven, so user stories are captured in event tuple records which “journalize” each event.
User Story:

As a/An	I want to…	So that…
Account holder	deposit or withdraw cash at a teller window	I can manage my cash

Event Tuple:

```
{
122                          # Event tuple type
Account #,                   # Bank account #
branch #,                    # Branch #
date-time,                   # When this transaction started
date-time,                   # When this transaction ended
amount of transaction,       # Amount of dollar credit or debit
media type,                  # USD cash, CAD cash, etc.
teller #,                    # Teller #
ID type,                     # How did I identify myself
…                            # etc., etc.
}
```

A similar event conducted at an ATM might be a different Event Record type, say 129, and a few fields would be different.

Thus each user story is expressed by an immutable Event Record tuple composed of all the data fields that the stakeholders believe will be of interest, for all present and future purposes. As the application is designed and evolves over time, there will be more and more ERs with lots of different fields. These ERs are the “Data” in ***Events Agile***, and the Archive of all ERs ever published is the sole data persistence for the application. There will be several identical copies of the Archive, located far afield, so the application is secure against data loss. The Archives are “write once, read many” or WORM. Again the ERs are immutable, and errors can be corrected by publishing adjusting ERs that link back to the original.

The Archive of ERs defines the data layer. Now where is the system Logic located? Well, the logic is in the Functions (aka microservices) which format and publish the ERs. Hence the ***Events Agile*** systems are very simple to design and build.

A recent Quora online poll posed the question, **“Is it normal to forget how your own code works?”** There were hundreds of responses from programmers of all levels of experience, and nearly all answered, “YES”.
https://www.quora.com/Is-it-normal-to-forget-how-your-own-code-works

If programmers do not know what they did in the past, do they know what they are doing now? This seems like a Major Problem!

By contrast, the programming required for the ***Events Agile*** Functions normally amounts to simple sorting and arithmetic applied to the fields of some Event Tuples. This can be accomplished with very basic coding skills in Python, say. And yes, you can remember how your Function code works, since it is, “logic that fits in my head”.

Who defines the Event Records and their data fields? It is the business stakeholders and subject matter experts. These will often be non-technical staff rather than the Developer team. These same folks can define the sorting and calculation that is required in their Functions, which create and publish the ERs. Here we see a major change at work, because the stakeholders are intimately involved in the software project, without being software engineers!
Events Agile addresses the divide between business and programming. This was a goal of the 2001 Agile Manifesto, and it has defied resolution ever since. We claim that ***Events Agile*** is ***Functional Agile***.
Let us revisit the Engineers’ complaint above, “Decisions are still business-driven, made by people without an understanding of technology.” But now this is a good thing!

Here are the four pillars of the 2001 Agile Manifesto. We demonstrate how Eventz Agile fits the bill:

Individuals and interactions over processes and tools
Stakeholders work together to define the Event Records and their fields. These Events are simply the activities of the regular operation of the business. They are known to all staffers.

Working software over comprehensive documentation
The Functions are working software. They are not accepted and paid for until they pass the operational tests. This means publishing the correct ER from a given event and test Archive.

Customer collaboration over contract negotiation
Customers are the stakeholders. They are calling the shots!

Responding to change over following a plan
The Eventz SDLC diagram shows how change orders affect ER data and logic Functions. The response to change is very fast! Changes to Reports can often be done directly from the Archive of Event Tuples, by an end user.
Behavior of the system over the Structure of the system.
Greg Young Events first
https://www.youtube.com/watch?v=esm-1QXtA2Q
Embrace Reality of eventual consistency. Jonas Boner
https://www.youtube.com/watch?v=1hwuWmMNT4c

We have seen that ***Events Agile*** resolves several long-standing computing system issues, by departing from the normal techniques.

1. Eventz does not persist data in databases. Instead, it uses a WORM Archive of immutable tuples called Event Records (ER)

and this Archive is the only form of data persistence. (This is data Representation as opposed to the normal data Encapsulation.)

2. The Event Records (ERs) journalize the “events” that occur in the transactional space. So ***Events Agile*** is event-driven, or Reactive programming.

3. System logic and data are strictly separated. Logic is housed in Functions (aka Microservices) and Data is held in the immutable ER event records. ***Events Agile*** is a “Functional Programming” method, so we prefer the name “Function” over “Microservice”.

4. The only messaging path available to the Functions is to “publish and subscribe” the ER records. So the messaging and the data persistence are the selfsame records. This is novel!

5. The Functions (aka Microservices) are technology-agnostic. All that matters is the ER tuples that enter and exit each Function.

6. ***Events Agile*** systems can be deployed in many ways: Edge, Cloud, LAN, WAN – you can choose freely according to your situation.

7. ***Events Agile*** is Simple, with a short learning curve. It is more like Python than C++.

8. Because ***Events Agile*** does not use any databases or web servers, it can be highly secure against exploits, exposing only a

tiny attack surface.

The *Events Agile* Software Development Life Cycle

The ***Events Agile*** Software Development Life Cycle is shown below. It is guided by the business stakeholders rather than by the programmers.
The steps are:

1. List each of the Event Records (ERs) and the data fields that are required for each ER.
2. List each of the Functions that are required, and describe what each must calculate.
3. Write and test the Functions.
4. Perform the pre-production testing.
5. Go to production.
6. Respond to any change orders while in production

Steps 1 and 2 are best done by the business stakeholders, while steps 3 – 6 are normally done by developer team. Often step 3 can be partially done by “citizen developers” who do are self-taught programmers.
The ERs are like the rows of a spreadsheet where the data fields are **spreadsheet cells**. The Functions are like **spreadsheet formulas**.

Notice that the skills required to design the Eventz system are only familiar spreadsheet skills!

In ***Events Agile*** projects the stakeholders do most of the work of designing the computing solution! When the Functions arrive at the In-Box of the Developer

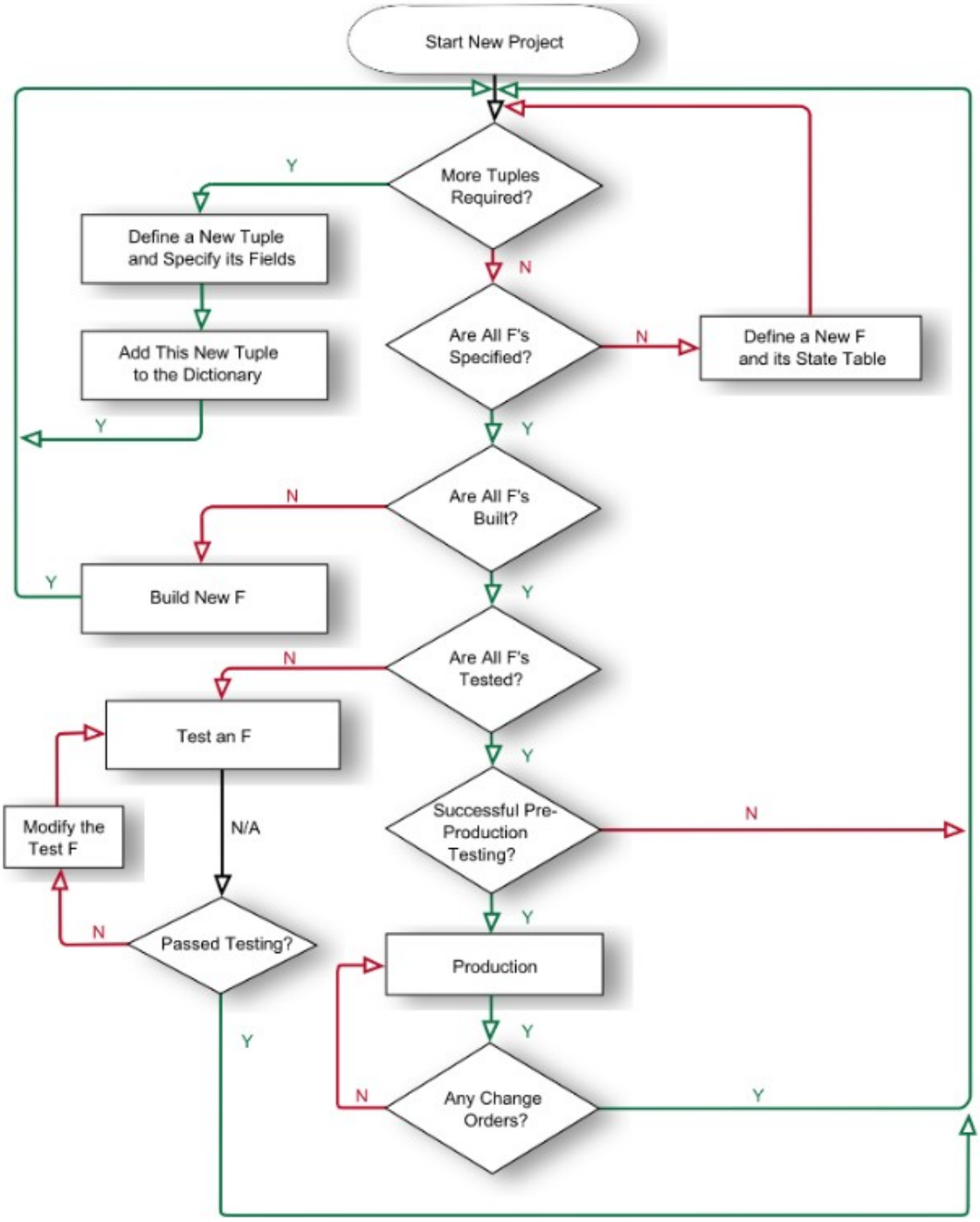
teams, they are fully specified, and the system strategy has been set.
Let’s examine a very complex payroll system. The Phoenix Payroll system has been called an “incomprehensible failure” by the Auditor General of Canada. Lots of things went wrong, and they cannot be fixed.
The Eventz process follows:

1. The stakeholders examined the collective agreement for Border Services, FB. They noticed that of the 62 articles, only 9 articles mentioned pay.

2. The Stakeholder group characterized the different pay scenarios and extracted the significant factors. Like Full or Part Time, Rest Days are not only weekends, Appointment level, acting pay for work at a higher level, options affecting union dues, RRSP deductions, Garnishee, tax deductions, etc. They designed prototype ERs with fields to capture all of these factors.

3. The Stakeholder group included Subject Matter Experts on the payroll logic and they enumerated the various pay scenarios using binary logic. One example is that a part time worker who works fewer than 4 hours on his rest day is grossed up to 4 hours pay. But a full time worker working less than 3 hours on her rest day is grossed up to 3 hours. There are dozens of these rules, and they were captured by the SMEs and written into the Function specifications. This effort dramatically reduced the analysis work normally required of the Development team.

SDLC


Start New Project
More Tuples Required?
Y
Define a New Tuple and Specify its Fields
Add This New Tuple to the Dictionary
N
Are All F's Specified?
N
Define a New F and its State Table
Y
Are All F's Built?
N
Build New F
Y
Are All F's Tested?
N
Test an F
N/A
Modify the Test F
Passed Testing?
N
Y
Successful Pre-Production Testing?
N
Y
Production
Y
Any Change Orders?
N
Y

APPENDIX 1 Agile Manifesto authors' comments on modern Agile

Mike Beedle (1962 - 2018)

"From 2010 – today, I own Enterprise Scrum Inc. that exclusively teaches, mentors, coaches and implements Scrum in enterprise environments."
He was a keynote speaker at countless Agile and Scrum conferences worldwide.
https://www.scruminc.com/mike-beedle-on-early-history-of-scrum/

Adam van Bennekum

"You have to change the paradigms. It starts at the top: management must be the role model for the rest of the organization. The objective is to break down silos and get people working together in a team while rethinking all approaches, such as document flows and traditional bureaucratic processes. This is not an easy job. The old paradigms are embedded in bureaucracy, personal development plans, trainings, role descriptions, career paths, etc. It is a difficult thing to change paradigms because people instinctively go back into old habits and old solutions whenever something goes wrong. And when you are learning new things, it can go wrong sometimes. So, you need people who can keep you from reverting back into old ways of working and help you to develop the forward-learning process. That's why organizations need Agile transformation coaches, who remind them to stay the course and guide them back to more Agile solutions."
https://blog.wemanity.com/arie-van-bennekum-agile-is-much-larger-than-just-it/

Alistair Cockburn

"However, Agile is still gaining popularity and is used by most software producing companies (according to the version one survey), how do you explain why people cant do the transition with their own people, and how can they improve their

chances of success?

I think those two questions are related. Actually, I'll take the second one first…

Consider that two factors are in play in agile adoption: diversity in personalities, and the shu-ha-ri progression.

Diversity in personalities should indicate that not everyone will like this different way of working. Agile development calls for:

> increased personal communication,
> speaking and accepting bad news about the project or situation, and
> living with increased ambiguity, flux, uncertainty.

Not every personality will enjoy those, indeed, it is easy to imagine a large percentage of people find them very difficult. For me, the ordinary diversity of personalities indicates an upper limit to the acceptance of agile practices.
https://www.adventureswithagile.com/2016/08/17/interview-alistair-cockburn/

Ward Cunningham

Ward Cunningham has contributed to the practice of object-oriented programming, in particular the use of pattern languages and (with Kent Beck) the class-responsibility-collaboration cards. He also contributes to the extreme programming software development methodology. Much of this work was done collaboratively on the first wiki site.
https://en.wikipedia.org/wiki/Ward_Cunningham

Martin Fowler

“Our challenge at the moment isn't making Agile a thing that people want to do, it's dealing with what I call faux-agile: agile that's just the name, but none of the practices and values in place. Ron Jeffries often refers to it as "Dark Agile", or specifically "Dark Scrum". This is actually even worse than just pretending to do agile, it's actively using the name "agile" against the basic principles of what we were trying to do, when we talked about doing this kind of work in the late 90s and at Snowbird.”
https://martinfowler.com/articles/agile-aus-2018.html

Jim Highsmith

“Authoritarian managers use power, often in the form of fear, to get people to do something their way. Leaders depend for the most part on influence rather than power, and influence derives from respect rather than fear. Respect, in turn, is based on qualities such as integrity, ability, fairness, truthfulness—in short, on character. Leaders are part of the team, and although they are given organizational authority, their real authority isn't delegated top-down but earned bottom-up. From the outside, a managed team and a led team can look the same, but from the inside they feel very different.”
https://www.goodreads.com/work/quotes/552587-agile-project-management-creating-innovative-products-the-agile-softwa

Andrew Hunt

The Failure of Agile

https://toolshed.com/2015/05/the-failure-of-agile.html

Ron Jeffries

Ron Jeffries Says Developers Should Abandon "Agile"

[Agile adoptions] often lead to more interference with developers, less time to do the work, higher pressure, and demands to "go faster". This is bad for the developers, and, ultimately, bad for the enterprise as well, because doing "Agile" poorly will result, more often than not, in far more defects and much slower progress than could be attained. Often, good developers leave such organizations, resulting in a less effective enterprise than prior to installing "Agile".

https://www.infoq.com/news/2018/06/developers-should-abandon-agile

Jon Kern

Jon: I'm happy that "agile" has spawned on new ideas. I like rocking the status quo. The only thing that has me mildly torn--and only mildly, because "to each his own" and "what-evs" come to mind--is the craze surrounding Scrum. Scrum, scrum, scrum, scrum. Part of me is happy that there are legions out there turning the sod under, trying to sow the agile seeds via Scrum. Another part of me thinks it is like Corn-to-Ethanol. A waste of energy for the dumbfounded among us, spurred on by lots of lobbying.

http://www.informit.com/articles/article.aspx?p=1739476

Brian Marick

Brian: I've drifted away from Agile consulting because the clever creation of the Scrum certification, and the nature of Scrum itself as basically a “management practice” without the icky programming stuff that XP had, caused it to be wildly successful among people who don't necessarily take it seriously. So you get these half versions of Scrum ("We do Scrum but ..." etc.), and the way that's taken over the enterprise is disheartening to me because there's a lot more potential there that I actually believe will never be realized simply because of the nature of large enterprises.

http://www.informit.com/articles/article.aspx?p=1739477

Robert C. Martin

Agile has failed. A peek at the future of programming. The real problem is the unrealistic expectation that programmers can see through the uncertainty of the customer and clearly craft the correct solution. We programmers ought to know that is pure crap.
https://www.reddit.com/r/programming/comments/5fx9bs/agile_has_failed_a_peek_at_the_future_of/

Ken Schwaber & Jeff Sutherland

Scrum.org sustains and enhances these largely free Scrum facilitation tools and processes. You can either figure them out or use them yourself, and I have created trainers and consultants to help you, if needed.
https://kenschwaber.wordpress.com/2017/01/12/scrum-21/

Dave Thomas
GOTO 2015 • Agile is Dead • Pragmatic Dave Thomas
GOTO Conferences 14 Jul 2015

Dave Thomas was one of the creators of the Agile Manifesto. A year ago, he told us that Agile is Dead. How could this be? Why had he deserted us? And what are we to do?

It turns out that while the "Agile" industry is busy debasing the meaning of the word, the underlying values are still strong. In this talk, Dave will draw a distinction, and show us how to reclaim agility for ourselves.

https://www.youtube.com/watch?v=a-BOSpxYJ9M&t=1475s

APPENDIX 2 Agile failure patterns are resolvable in Eventz

Stefan Wolpers: Analyzing my past projects, I identified the following four cross-organizational patterns (Organizational, Team, Process, Facility) that are making Agile transitions to much more harder, less effective and significantly more expensive.
(Below the failure pattern is numbered, and the Eventz position is indented beneath.)

Agile Failure At Organizational Level

1. Not having a (product) vision in the first place: If you don't know where you are going, any road will get you there.

   In Eventz it is the stakeholders who guide and refine the system vision. “We’ll know it when we see it.”

2. The fallacy of "We know what we need to build." There is no need for product discovery or hypotheses testing; the senior management can define what is relevant to the product backlog.

   Stakeholders at all levels are encouraged (and enabled) to participate in the SDLC process, all through the life of the system.

3. A perceived loss of control at management level leads to micro-management. (The ‘what-is-in-for-me-syndrome.’)

   Management can participate in the SDLC along with everyone else. Managers are free to commission their own customized reports from the Archived records.

4. The organization is not transparent about vision and strategy hence the teams are hindered to become self-organizing.

   Teams will self-organize using the SDLC process. Vision and strategy can be articulated in the beginning.

5. There is no culture of failure: Teams, therefore, do not move out of their comfort zones but instead play safe. (Mathgeek on HN suggests to replace "culture of failure" with "culture of learning to get back up again after falling.")

    The Functions (or Microservices) can be design competitions. Some will fail and the cost is low. Discovery brings failure and success.

6. The organization is not optimized for a rapid build-test-learn culture, and thus departments are moving at different speed levels. The resulting friction caused is likely to equalize previous Agile gains.

    The Eventz SDLC supports and implements a rapid build-test-learn culture.

7. Senior management is not participating in Agile processes, for example, the sprint demos, despite being a role model. Instead, they do expect a different form of (push) reporting.

    Senior management can and should participate in the SDLC process. It is civilian-friendly.

8. Not making organizational flaws visible: The good thing about Agile is that it will identify all organizational problems sooner or later. (“When you put problem in a computer, box hide answer. Problem must be visible!” Hideshi Yokoi, former President of the Toyota Production System Support Center in Erlanger, Kentucky, USA.)

    The SDLC supports test-and-adapt improvement cycles. The cost of change is low. Commitment to the current path is flexible. Eventz imposes a posiotive organization.

9. Product management is not perceived as the “problem solver and domain expert” within the organization, but as the guys who turn requirements into deliverables, aka “Jira monkeys.”

    Product management is just stakeholders, like everyone else. Fred George says these titles are obsolete, leftover from waterfall.

https://www.youtube.com/watch?v=2Fy_xidc11w

10. Other departments fail to involve product management from the start.
    Typical behavior in larger organizations is a kind of silo thinking, featured by local optimization efforts without regard to the overall company strategy, often driven by individual incentives, e.g., bonuses. (Personal agendas are not always aligned with the company strategy.)

    The stakeholders should come from all departments. They can customize their reports. Operations should be optimized for all departments. Any conflicts can be negotiated so all are served.
11. The sales organization guards access to customers, thus preventing teams from learning.

    Customers can and should be stakeholders in the SDLC process.
12. Core responsibilities of product management are covered by other departments, e.g., tracking metrics, thus leaving product dependent on others for data-driven decisions.

    Eventz is always data-driven. Product management is shared by all stakeholders.
13. Product managers w/o a dedicated team can be problematic if the product management team is oversized by comparison to the size of the engineering team.

    The engineering team builds the Functions. Eventz Product Management is shared by all stakeholders.
14. Engineering teams are not free to choose “their” tech stack.

    Eventz is tech agnostic in that the creator of each Function can choose his technology. There can be design competitions.
    Currently we use Python only. We require AMQP and RabbitMQ. The technology can and should be implemented in other languages but for now, we restrict them.

Agile Failure At Team Level

1. There are too many junior engineers on an engineering team. They tend to appreciate micro-management as part of their training. Usually, they have no or little experience with Agile methodologies. Hence they hardly can live up to processes, particularly they fail to say “No.”

   Junior engineers can perform well in creating the Functions. These are single-person programs that are very well defined. Test records are delivered with the assignment. The engineers may well be remote, outsourced.

2. Engineers with an open source coding mentality: Tasks are discussed on pull requests once they're finished, but not in advance during grooming or sprint planning sessions. (They tend to operate in a distributed team mentality.)

   Functions built using Python will be largely open source. Tasks are assigned as Functions, defined by the stakeholders using the SDLC. Eventz specifies input and output. Gui standards should be specified. The developer if free to operate within these constraints.

3. Teams are too small and hence not cross-functional. Example: A team is only working on frontend issues and lacks backend competence. That team will always rely on another team delivering functionality to build upon.

   The business stakeholders operating in the SDLC are defining both frontend and backend functionality. The deployment of the Functions may well be at the network edge. Wait and see.

4. Teams are not adequately staffed, e.g., scrum master positions are not filled and product owners have to serve two roles at the same time.

   The staff is the business stakeholders. Fred George advises that we don’t need the titles like scrum master.

5. Team members reject agile methodologies openly, as they do not want to be pushed out of their comfort zones.

The stakeholders operating in the SDLC are in their comfort zones.

6. Teams are not self-organizing. That would require accepting responsibility for the team’s performance and a sense of urgency for delivery and value creation. A "team" in this mode is more acting like a group—for example, people waiting at a bus-stop—: They are at the same time in the same place for the same purpose, but that does not result in forming a team. (The norming, storming, forming, performing cycle seems to be missing.)

   In Eventz the stakeholders are self-organizing. It can be a productive group effort. Success arrives daily in the form of Functions that test well.

7. Even worse, team members abandon Agile quietly, believing it is a management fad that will go away sooner or later.

   Eventz is a very comfortable fad. No reason to abandon it. Eventz will be perceived as valuable to all players and will not be abandoned.

8. **Faux Agile**: Teams follow the “Agile rules” mechanically without understanding why those are defined in the first place. This level of adoption often results in a phenomenon called "Peak Scrum": There is no improvement over the previous process, despite all Agile rules are being followed to the letter. Or even worse: morale and productivity go down, as the enthusiasm after the initial agile training wears off quickly in the trenches.

   The stakeholders operating in the SDLC are perpetually engaged. The fruits of their labors arrives daily in the form of tested Functions.

9. Moving people among teams upon short notice. Even if required for technical reasons, this has a negative impact on a team's performance & morale.

   Stakeholders can move freely between “teams”. The Eventz process is the same everywhere. This can be a problem for any team endeavor.

Agile Failure At Process Level

1. Agile light: The management abandons self-organization the moment a critical problem appears to form ‘task forces’ instead.

   The SDLC is easy to tolerate. Critical problems are easy to diagnose and repair. The Archive is a perpetual log of all events. It can be elevated to a high watch if needed to detect elusive faults.

2. Agile processes are either bent or ignored whenever it seems appropriate. There is a lack of process discipline.

   The SDLC is easy to tolerate. No reason to bend of ignore the process. If the rules are broken, it is easy to diagnose and put the SDLC back on track.

3. Agile processes are tampered with, e.g., the scrum product owner role is reduced to a project manager role. Often this is done so by assigning the task of backlog ownership to a different entity at management level. ("Scrum" without a product owner makes a magnificent Waterfall 2.0 process.)

   Fred George advises to ditch the titles. They are obsoleted in the Eventz SDLC process, which only recognizes stakeholders.

4. Stakeholders are bypassing product management to get things done and get away with it in the eyes of the senior management, as they would show initiative.

   Fred George advises to ditch the titles. They are obsoleted in the Eventz SDLC process, which only recognizes stakeholders.

5. The organization is not spending enough time on team communication and workshops to create a shared understanding of what is to be built.

   What is being built is defined by the growing dictionary of DS Records and Functions. This is open and easily inderstoof by all stakeholders. Communication is good but Microservices architecture implies that

the Microservice builder does not need to know where his service fits. Only what it is given and what it must provide. (This may not be true for GUIs)

Agile Failure At Facility Level:

1. A team is not co-located, not working in the same room, but scattered across different floors, or worse, different locations.

   The contributions of stakeholders to the SDLC is supported by cyber meetings. Different time zones are not problematic.

2. The work environment is lacking spots for formal and – more important – informal communication: cafeterias, tea kitchens, sofas, etc.

   Debate by the stakeholders is focused on the DS Records and the definition of the Functions. This is normally very productive. Disagreements about Reports need not get heated, since each stakeholder can create or commission her own reports.

3. The work environment is lacking whiteboards. The absence of whiteboards on every available wall within the office should be questionable, not having them.

   Debate by the stakeholders is focused on the DS Records and the definition of the Functions. This can be efficiently done using cyber meetings and emails.

4. Agile requires suitable offices to further collaboration: spacy, with plenty of air and light. But they should not be a mere open space, which tends to get too noisy, particularly when several Scrum team have stand-ups at the same time.

   Eventz is by nature not very sensitive to the office configuration. People can toil in the dark.

https://age-of-product.com/agile-failure-patterns-in-organizations/

1.10 Data Representation vs. Encapsulation

David Bau has written a very helpful paper on this topic.
http://davidbau.com/archives/2003/11/12/encapsulation_or_representation.html

1. **Encapsulation**. The engineers can agree upon the "behavior" they must build and the "behavior" other components of the system must support.

2. **Representation**. The engineers can agree upon the "data" they are allowed to produce and the "data" they need to consume.

“As soon as we move to larger, more loosely-coupled networks of systems where individual programs can come and go while the network of data exchange remains, data representation becomes a much more powerful technique.”

OO Data Encapsulation

- Data is owned by an Object
- The Object’s Methods expose the data
- This is Encapsulation
- HOWEVER, the data-relational impedance mismatch arises because the data is in fact in Database tables!!
- Representational Data is data stored elsewhere, outside the Object, and not in tables.

Data vs Information

- **y** tuples are Event Records. They are Data.
- Information is derived from Data. In the future.
- So **Reporting** can be deferred to a future time!

- 80% of change orders are in Reporting!
- Because every ER ever published is available in the Archive, Reporting is certainly possible!
- Functions that Report can be written anytime, by anyone!
- Can computing be so simple?

Systems that use databases with corrupt data seldom recover their integrity. But the ERs used by Eventz systems can be refined over time to purge the errors and converge toward accuracy.

1.11 David Parnas' 1972 paper

"We propose that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to **hide** such a decision from the others." David Parnas, 1972

On the Criteria to Be Used in Decomposing Systems into Modules

http://sunnyday.mit.edu/16.355/parnas-criteria.html

Eventz seems to meet the objectives set out in this classic paper. Our Functions act as Dr. Parnas' "Modules" were intended. We welcome discussion and debate on this topic.

1.12 Disruptive Technology

Eventz has the potential to be disruptive to conventional software development methods, because it has attributes that are recognizably superior.

- Inherently Agile as conceived by the 2001 Agile Manifesto.
- Breaks down the barriers between business and programmers
- Solves Brooks Law (Adding staff makes a late project later).
- Enables outsourcing of Functions
- Enables parallel development of Functions without strain
- Reduces costs
- Late arriving changes are not disruptive
- Relieves the shortage of technical talent
- Enables Greenfield development of large systems
- Supports systems with many rules
- Disaster Recovery can be built in and tested easily
- All data is persisted in immutable Archives
- Uses Functional programming
- etc.

1.13 Impostor Syndrome in Eventz

Imposter Syndrome is a psychological pattern in which an individual doubts their accomplishments and has a persistent internalized fear of being exposed as a "fraud". Despite external evidence of their competence, those experiencing this phenomenon remain convinced that they are frauds, and do not deserve all they have achieved. Individuals with impostorism incorrectly attribute their success to luck, or as a result of deceiving others into thinking they are more intelligent than they perceive themselves to be.
https://en.wikipedia.org/wiki/Impostor_syndrome

This is common in computing professionals. The challenges inherent in Eventz are smaller than usual, because the Functions and the ERs are quite simple and straightforward. Unlike other methodologies, Eventz guides the stakeholders and developers to a reliable solution in the form of a simple reliable well-documented system.

1.14 Data vs. Information

Are data and information the same thing? No!

Data is just the record of events that happened.

Information is the meaning attributed to these events.

Eventz separates the two concepts in a very productive way.

When an event happens, a Function will spring into action and create an ER that fully describes the event, and then publish this ER. That’s all the Function needs to do.

Other Functions will have subscribed to the ER and will spring into action to do something based on the meaning of the event. Hereby, the data becomes information.

Eventz is Data-base free

Eventz has no central database for system-wide data persistence. Instead Eventz uses the Archive of old ERs for all data persistence.

However the Functions are free to employ local databases, and they frequently will. Indeed, there are hundreds of special-purpose databases, and Functions are free to select one of these to meet a particular mission.

1.15 Complexity – Can Eventz be so simple?

We often hear the comment, “How can it be that the Eventz solution to my problem is so simple?”
Eventz is designed to be simple. Nothing else in computing is simple, and complexity is perhaps worn as a badge of intelligence.

Fred Brooks identifies two kinds of complexity:

1. Essential Complexity
2. Accidental Complexity

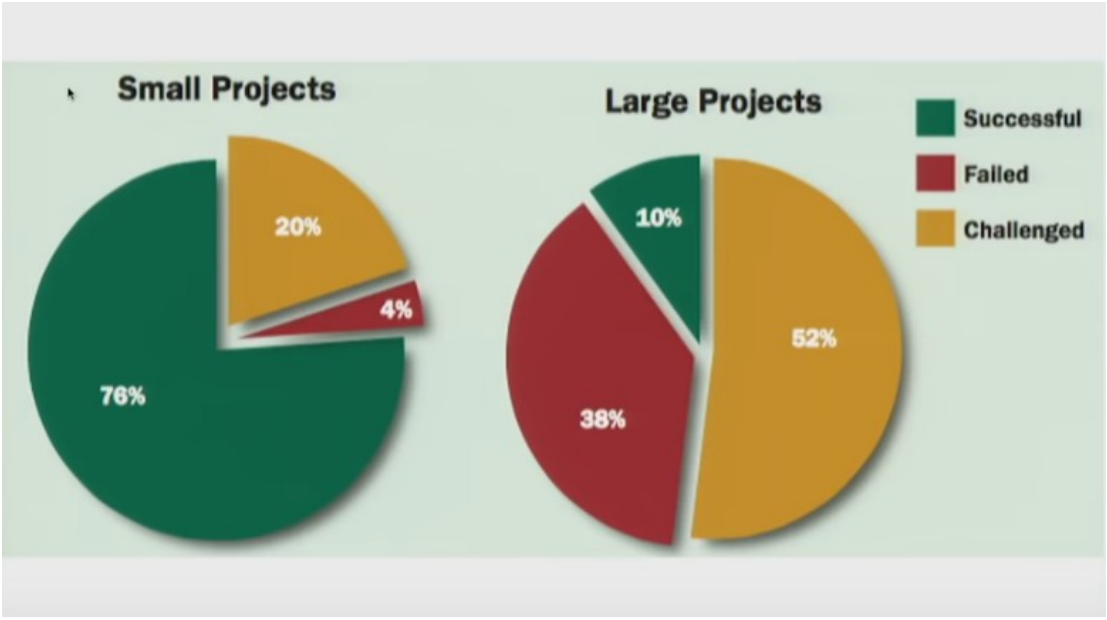
Essential is just some Credit and Debit postings and some conditional logic.
Accidental is encapsulated data, complex rules engines, large commercial databases, etc.
Accidental Complexity = haystack
Essential Complexity = needles

Fred Brooks, 2008:
“I know of no field of engineering where people do less study of each other’s work, where they do less study of precedents, here they do less study of classical

models. I think that this is a very dangerous thing."

Eventz helps to resolve this problem, because the immutable data ERs describe events and the Functions create these ERs. Both can be discussed in terms that make sense to the stakeholders. Weak Functions can be re-written to be stronger. Missing data fields can be appended to the ERs.

Software in crisis – Agile failure at scale



https://www.youtube.com/watch?v=rNQR1HqfEl0 Chad Fowler

Jeff Gothelf: Fixed time, fixed scope projects always end in 1 of 3 ways. None of them good:

1. We move the deadline.
2. We reduce scope.
3. We implement "crunch mode", everybody puts in 80 hour weeks till

the deadline, burns out, quits and goes to work somewhere else. https://medium.com/swlh/fixed-time-fixed-scope-projects-always-end-in-1-of-3-ways-none-of-them-good-9fa66e7d129e

Quoting Eventz project cost and duration, after proof of concept

In a Eventz project it is wise to finish a preliminary analysis using the SDLC methods. This yields a dictionary of the ERs that will be needed, and a list of Functions required to produce these ERs. With these in hand, it is possible to make up bare-bones minimum viable product proof of concept. The work can then be quoted with some confidence.

What skillsets are required in Eventz?

$$y = F (Y, e)$$

1. Skill to identify the events, e
2. Skill to specify and program the Functions, F
3. Skill to identify the necessary fields in y

Is this hard?

Chapter 2 Specification

2.1 Requirements Specification

We need to talk about Requirements Specifications.

It is widely accepted that if your Specifications are vague, then your software project is doomed to failure.

The problem with this is that the world keeps changing, so it is unrealistic to lock and freeze your Specifications. This is a major problem!

The Eventz SDLC diagram above is exempt from this dilemma. Because the Specifications remain flexible! They are never frozen.

So Eventz is malleable software!

2.2 The Eventz SDLC (Software Development Life Cycle)

Eventz works on novel principles:

1. The sole data persistence is the set of immutable tuple Event Records. (Eventz does not use a database to persist data.)
2. The sole logic is the Functions that create and publish these Event Records.
3. There is no formal requirements specification up front.

The business stakeholders are trained in the events that constitute the business processes. These are the policies of the organization. Each event is specified with all the data fields necessary to fully characterize the event.
The logic of the Eventz system is just the Functions that are required to format and publish the Event Record to the Archive. Additional Functions will calculate Reports from the ERs in the Archive.

The SDLC process:

1. Create prototype ERs for all or most Events.
2. Specify the Functions that calculate the necessary fields for each ER.
3. Program and test each Function.
4. Change Orders are handled by altering the ERs and Functions as required.

In Eventz, the Software Development Life Cycle is shown below. The steps are:

1. List each of the Event Records (ER) and the data fields that are required for each ER. [1, 2, 3, 4, 5]
2. List each of the Functions (microservices) that are required, and describe what each must do. [1, 6, 7, 8]
3. Build and test the Functions. [1, 6, 9, 10, 5] & [12, 19]
4. Perform the pre-production testing. [13, 14, 15]

5. Go to production. [16, 17, 18]
6. Respond to Change Orders [15, 1]

Steps 1 and 2 are best done by the business stakeholders, while steps 3 – 5 are normally done by IT staff. Often step 3 can be partially done by “citizen developers” who do not have formal training.

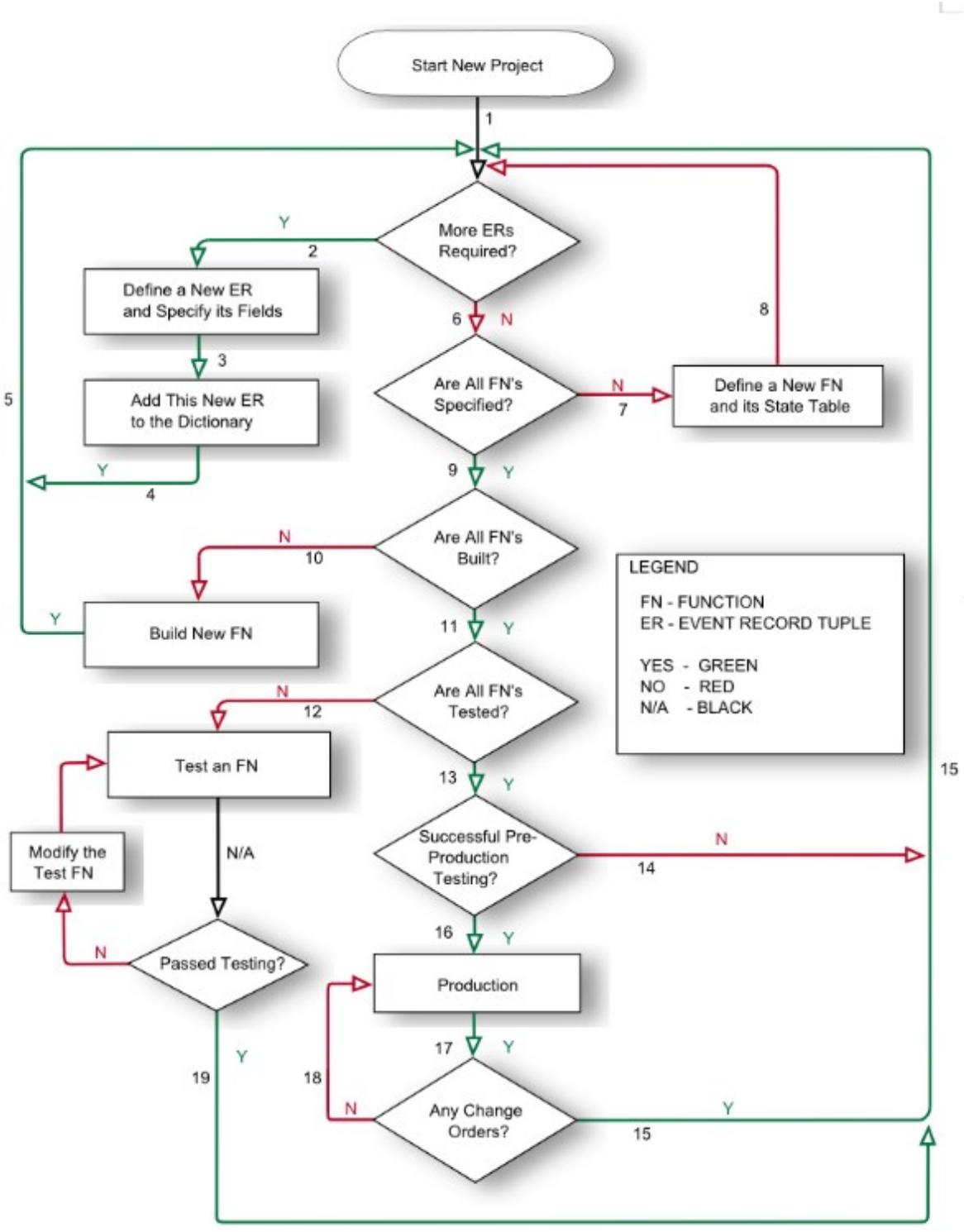
The ERs are like the rows of a spreadsheet where the data fields are **spreadsheet cells**. The Functions are like **spreadsheet formulas**.

So the skills required to specify the Eventz system are just familiar spreadsheet skills!

The impact of late-arriving changes is minimized, because changes to the SDLC are limited to changing the fields of some ERs and changing the formulas calculated by some of the Functions. Only the Functions that subscribe to a changed ER will need to be modified.

This simple scheme tends to avoid the “bad fix” events where fixing one thing breaks other thing(s). It also leads the maintenance programmer directly to the area that needs changing.

Notice that this change process continues to work even in production. Often a change will result in a new field appended to the end of an ER. This is not disruptive to the other fields and does not disrupt the Functions unless they need to make use of the new field(s).



SDLC DIAGRAM

2.3 How to specify an Eventz Function

The defining relationship for Eventz is:

$$y_n = F_x(Y, e_n)$$

where:

- $\mathbf{e_n}$ is some Event, like a new message, a key-press, or a timeout, etc.
- $\mathbf{y_n}$ is an Event Record (ER), an immutable tuple that describes the event, e_n.
- $\mathbf{Y = \{y_0, y_1, \ldots, y_{n-1}\}}$ is the set of every ER ever published, the Archive
- $\mathbf{F_x}$ is the Function (program) that calculates the necessary fields for the ER that describes event e.

The process we use to specify a Function F_x is to **describe the event** e_n (it can be a keypress, timeout, incoming message, etc.) and to **define the fields we require** for the ER tuple y_n and the **calculations** required. Normally some Archived ERs are involved in the calculations.

The Function specification will also have to include test events, and a reference Archive of previous ERs. The developer of the Function should only submit Functions that pass all the tests.

Finally, the Function must pass a Disaster Recovery test, proving that it can do a cold boot using only the Archive ERs.

EVENTZ Functions are technology-agnostic

The Developer of each microservice Function is free to choose her tools. There is no requirement for the whole project to use the same language. We like Python 3 for its simplicity. As long as the ERs that are published are correct, it matters little how they were computed.

Of course the stakeholder team that is managing the SDLC process can make its own decisions on standards, and it may elect pure Python for maintainability

reasons.

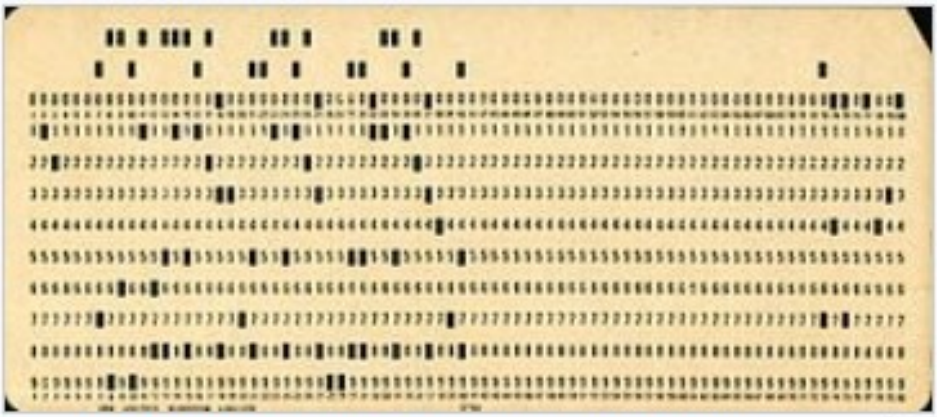
If a certain Function needs to run faster, then it might be re-written in C, say.

2.4 How to develop the Event Record tuples

The Event Records (ER) are specified by the business stakeholders, who are familiar with the events and policies of the organization. A remote Eventz coach may be helpful to assist and guide this work.

Each ER is an immutable tuple composed of every data field that is of interest to define and characterize the event. If new or different fields are required in the future, these can easily be appended to the tuple. This is the main method used to execute change orders.

Deja vu - A sequence of immutable data fields was common 50 years ago!



A punched card from the mid-twentieth century.

Imm

We [illegible] (ERs). Our **tuples** are ordered sets of data and metadata that characterize and journalize transactional actions or events.

Our tuples are **immutable** (as are all tuples in the Python language), meaning that once a tuple is published it remains unchanged in the Archive forever. If an error is found, then a correcting ER can be published, with a UUID that links back

to the original ER.

The collection of immutable ERs in the Archive is a record of all the past events.

Reports can be created at any time by anyone who has access to the Archive.

Changes to Reports (not changes in the way events are recorded) are the most frequent change orders.

2.5 Business Stakeholder Role

The business stakeholders are trained in the policies and events of the organization they serve. They probably are using spreadsheets, whose cells are destined to become the fields of the Event Record tuples.

The stakeholders are invited to specify the ERs and their data fields, to support and empower the Eventz management system.

We anticipate that the stakeholders will participate enthusiastically, because they have pride of participation. The Eventz system will be created as they decide, and the business events will be well supported.

Eventz can heal the divide between business and programming

Eventz is an open source framework for microservices. Eventz is designed to be simple, with a short learning curve.

It is a methodology for creating and maintaining transaction processing software that is easy to create, easy to test, and easy to change.

Using Eventz, teams of “civilian” stakeholders (who are not computer professionals) can assert control over the design of their computing systems. The stakeholders can use their familiar spreadsheet-like skills to express what they need their systems to accomplish.

Developers continue to create the Microservices (or Functions) which execute the specified logic, but the design is governed by the stakeholders. This can ***heal the divide between business and programming!***

https://video.log3.org/8944.html

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, **we value the items on the left more.**

Principles Behind the Agile Manifesto

We follow these principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of

months, with a preference to the shorter timescale.

- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

https://agilemanifesto.org/principles.html

Extract, translate and load

Sometimes you will need to take specific data from the Archive records and export it to some other computing system.

Extracting data from the ERs in the Archive is easy, because the dictionary defines exactly the meaning of each field of each ER type. This data is clear and unambiguous.

Data conversion from historical sources

Most systems will start with historical legacy data from the previous systems. This conversion normally brings errors, which can be difficult to fix. Porting data from deep inside one database to deep inside another database is a very challenging task!

Happily, in Eventz we do not do that. We export data from the old database in CSV format (say) and then process these records into ERs. This gives us a startup Archive that contains the old data in our new ER format. This is data representation rather than encapsulation.

Systems that use databases with corrupt data seldom recover their integrity. But the ERs used by Eventz systems can be refined over time to purge the errors and converge toward accuracy.

2.6 Eventz Disaster Recovery Strategies

Disaster Recovery is a basic system requirement. But how often does it work when needed? In Eventz we can always depend upon the immutable Archive as the reliable persistence of past events. So we make DR a system requirement. It does not happen automatically so you must make it is priority. Every Function needs to demonstrate its ability to cold start from the Archive alone, with no surviving internal files. If a Function cannot do this, send it back!
In conventional systems it is challenging to recover from a crash. Recovery from backups sometimes fails to work when needed. Some systems are fragile, for example an airline that is grounded for 5 days. In a ransomware attack, Databases can be lost.
https://bgr.com/2016/08/14/delta-finally-explained-how-one-power-outage-grounded-an-entire-airline/
Eventz systems have a unique advantage in DR, because the immutable Archive has every Event Record that was published since time began, right up to the most recent. So every Function can be booted up from a cold start, anytime, by reading Archive ERs.
Every Delta Airlines ticket, boarding pass, seat assignment, baggage, gate assignment, etc. remains intact.
This can be QA tested, anytime, offline!
DR must be a requirement of every Function in the project!

TechRepublic member sagilbert47201 writes, "We have a VMware Backup server setup on a Synology. Our Domain Controller got infected and the synology was hooked up directly to the domain controller and looks like it got infected too.

I have done some research on these .harma files and it looks to be ransomware so the only solution for data recovery would be to restore from a backup.

However, the backup was on the Synology that was hooked up directly to the domain controller. I guess I am trying to look for maybe some other solutions on getting the data back?

Thank you for your help!"
https://www.techrepublic.com/article/vmware-server-is-infected-with-ransomware/?ftag=TRE684d531&bhid=24733659505574652644636918420915

2.7 Eventz Greenfield Developments

Large systems like MRP, ERP, Payroll are believed to be too complex for a Greenfield (blank sheet) development.

However the Eventz paradigm grows by processing Events one by one, until they are all handled. This is incremental Greenfield development. Most Event Records are influenced by previous ERs, which are all in the Archive.

Payroll systems, for example, have numerous Events and there can be a large number of Rules, especially if there are 80 different collective agreements.

We believe that complex systems must be developed as Greenfield projects, since modifications to commercial off the shelf software functionality have not been very successful.

2.8 Report Functions Can be Deferred

Reporting is facilitated in Eventz. It can be done independently from the Archive.
Most change orders are for Reporting
The collection of immutable Event Records in the Archive is a record of every past event. Reports can be created at any time by anyone who has access to the Archive. Business stakeholders are empowered to create their own reports, and this helps to break down the barrier between business and programmers.
Change orders are frequently for changes to reports, not changes in the way events are recorded.
As the Eventz project advances, it is very helpful to be able to defer most reporting work. We are required to include every field that will be required for reporting, but we can delay the report Functions until later when the whole picture is known.

2.9 Systems Having Many Rules

High profile public computing failures are not rare. These have tended to be in systems that must process many rules. A government payroll system like Phoenix needs to cope with 82 collective agreements.

We developed a payroll system for one collective agreement using Eventz, and the results were very encouraging. See fixphoenix.ca

We believe that complex systems must be developed as Greenfield projects, since modifications to commercial off the shelf software functionality have not been very successful.

2.10 Design Competitions, Kaizen

In Eventz systems the cost of each Function is rather low, so it is feasible to try out different candidate strategies. Kaizen continuous improvement is possible in Eventz. This appears to be a good remedy for technical debt.

Chapter 3 Development

3.1 Eventz Documentation

Documentation lives in the form of the Event Data Definition records (EDD), Function descriptions and pub-sub lists.

Accurate documentation is a challenge for all software systems. Eventz projects keep useful documentation current in the form of EDDs, the p & s roster, Function test records and Function descriptions. The Archive is a 24/7 log of all ERs published, which constitutes a dynamic testimonial as to how the system works. In this sense, Eventz systems are self-documenting. The code in each Function may not be commented well, but the mission of the Functions is clear. Functions are: “Code that fits in my head.”

3.2 Datetime in Eventz

Each Event Record is timestamped using GMT time. The offset to local time is included. ERs may arrive out of order but they can always be sorted into time order. This can be important for many reasons.

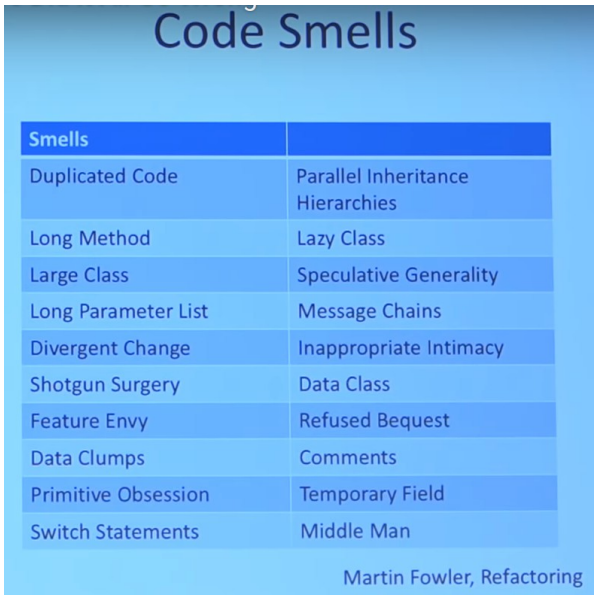
3.3 Eventz Technical Debt

In Eventz the code is solely in the Functions y = F (Y,e). The only task of a Function is to format and publish an Event Record to memorialize an Event. So there is little reason to accept sub-optimal code. If a Function performs properly and passes the tests, it is accepted for production. In addition, it is easy to review each Function and decide whether it is worth re-writing for better speed, style, maintainability. This is Kaizen continuous improvement.

So in Eventz systems any technical debt can be repaid.

“Refactoring” a Function is improving the design of existing code after it has been written (and tested). This can be a safe, small change.

Code smells:



You’re on your own now.

Imagine a junior programmer hired to work under an experienced mentor developer. Then the senior person leaves, so the new hire is on her own!

In an Eventz environment, this is actually not so scary. The maintenance work is orderly and rational. The documentation of the fields in the ERs is current and

accurate (based on the development contracts for the Functions). The business stakeholders remain in charge of the SDLC. There are likely to be remote developers who created some of the existing Functions. The Archive is eternal. Disaster recovery is in place and has been tested.

> While technical debt is often characterized as the result of "taking an easy but suboptimal decision in the software development process," I like Dormain Drewitz's suggestion that "all code is technical debt." Why? Because it's easy in hindsight to negatively classify code decisions and the resultant cruft, but at the time of writing the code there might not have been a better option. Indeed, it might have been the absolute best option at the time.

https://www.techrepublic.com/article/in-praise-of-developers-who-delete-code/?ftag=TRE684d531&bhid=247336595055746526446369184209 15

3.4 Progress Metrics

As a Eventz project matures, there are more and more Functions on the shelf, tested and known working. This is a trustworthy measure of progress for project management.

Progress against plan is the number of tested Functions on the shelf.

3.5 Risk of Project Failure. Recovery mode.

In conventional software development projects there is a significant failure risk. When a project heads downhill, often there is little prospect of a remedy.

Is Eventz different in this? We believe it is different, because of its simplicity. If there are errors and misconceptions in some of the ERs, this can be detected and repaired. If some Functions are unreliable, this can be detected and repaired. If there are errors in the ERs converted from a previous system, these can be detected and repaired.

As the Eventz project advances, there are more and more Functions on the shelf, tested and trusted.

Late-arriving change orders are not disruptive to the system, since these can be handled by changes to some ER definitions and some Function logic.

Overall, the risk of failure with Eventz is low.

Chapter 4 Deployment

4.1 Eventz Orchestration

Eventz systems require multiple Functions to be running and communicating with the Broker via RabbitMQ.

The System Monitor (SM) uses “ping all” messages to determine the current roster of Functions. Some Functions are intended to be solitary, and others are allowed to be numerous. The SM can launch Functions and also slay them, using System Messages. It can also control the routing list used by RabbitMQ.

4.2 Testing Eventz Functions, TDD

Test driven design. Unit tests.

Eventz Functions can be tested by processing a series of input Event Records. If the Function remains running and publishes the correct ERs then it passes the test. We can maintain a test set of ERs that have caused errors and crashes in the past. It is easy to perform OAT (operational assurance tests) in a lab (not production) environment.

> One of the weaknesses of TDD as originally described is that it can **devolve into a programmer's technique used to meet a programmer's needs**. Some programmers take a broader view of TDD, facilely shifting between levels of abstraction for their tests. However, with ATDD there is no ambiguity – this is **a technique for enhancing communication with people for whom programming languages are foreign**. The quality of our relationships, and the communication that underlies those relationships, encourages effective software development. ATDD can be used to take a step in the direction of clearer communication". Kent Beck, foreword to ATDD by Example
>
> "Instead, I involve business experts (on-site customers) closely throughout the iteration. I do have them create concrete examples, but only when faced with particularly complex topics, and never with the intention that they be run by a tool like Fit. Instead, the programmers use those examples to inform their test-driven development, which may or may not involve creating automated tests from the examples, at the programmers' discretion." James Shore, http://jamesshore.com/Blog/The-Problems-With-Acceptance-

Testing.html

These two problems – that the **customers don't participate**, which eliminates the purpose of acceptance testing, and that they **create a significant maintenance burden**, means that acceptance testing isn't worth the cost. I no longer use it or recommend it.
http://jamesshore.com/Blog/The-Problems-With-Acceptance-Testing.html

I used to believe in the tools, I believed that having to pass those tests kept developers honest, they could not declare done when they felt like it.
But I think now the problem was how they wrote their unit tests – we compensating with ATDD tools for method based testing not behavior based testing.
Instead, we should have been fixing the problem at source – write unit tests that focus on behaviors and thus can be used for acceptance.

"BDD is a second-generation, outside-in, pull-based, multiple-stakeholder, multiple-scale, high-automation, **agile methodology**. It describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software that matters."
Dan North, 2009 Agile specifications, BDD and Testing eXchange

It evolved into a methodology for facilitating the transmission of requirements from customer to developer through scenarios that can be automated.
BDD started with a similar insight - that we misunderstood TDD.
It realizes that specifying scenarios is the key to driving test-driven

development.
It created tools like RSpec and JBehave.

Mock Object

How do you test an object that relies on an expensive or complicated resource? Create a fake version of the resource that answers constants.

Kent Beck, TDD by Example

Classic example is database
-Log time to start
-Difficult to keep ‘clean’
Tie tests to physical location on network
- Write tests against something that acts like Db

BDD = Behavior Driven Development

4.3 Monitoring for exceptional users

Let us imagine an Eventz system that is under attack by a bad actor.
Can we design a supervisory Function that monitors the ERs looking for suspicious activity?
We gain if the subnet traffic is restricted to ERs and other traffic is not permitted. This closes the most attractive exfiltration route.
The authorized ERs were designed by the business stakeholders to memorialize each Event in their space. Presumably there will be no ERs that will export stolen data.
We gain if the Archives are protected from exploits such as re-naming the file, encryption, etc. This can be done because the Archives are WORM stores. And there are several Archives in different locations.

4.4 Conversion of data into Eventz Records

Most systems will start with historical legacy data from the previous systems. This conversion normally brings errors, which can be difficult to fix. Porting data from deep inside one database to deep inside another database is a very challenging task!

The first step in exporting data from a relational database to Eventz is to define an Eventz record for each table. Export processes can be built that bulk convert each table into the Eventz records and then publish them. The published records are stored in receiving archive(s) as initial events in the startup Archive.

Triggers can be configured to intercept database changes and publish the events and keep the archives synchronized with legacy activity.

This is data representation rather than encapsulation. (See 1.10)

Chapter 5 Maintenance

5.1 Maintenance after the original Dev team has left

In Eventz projects the development dynamics are quite novel.

The business stakeholders perform the analysis by specifying the Event Records and their fields, and thereby defining the Events that must be supported in their application space.

The Developers are responsible for writing and testing the Functions that create and publish the Event Records in response to each Event.

So the project is not dependent upon the genius of the Developers. The business stakeholders have good longevity, as the Events tend to remain consistent over time in any organization. Change Orders are handled by altering the ERs and the Functions to support new and changed Events. This is an orderly process.

5.2 Eventz 24/7 Logs for debugging

The immutable Archive contains every Event Record ever published. The ERs are time-stamped and include metadata for debugging purposes.
So when a bug appears, it is possible to trace its origin and the circumstances that caused it. As well, the scenario can be re-run to test the effect of the fixes.
It may be helpful to institute an elevated level of reporting from a difficult Function in order to gain insights into the failure mechanism. These might be temporary debugging ERs and they may be stored in a debugging Archive and may have a short lifespan – the duration of the troubleshooting, perhaps. This option could be initiated by the maintainer of the Function.

5.3 Determining culpability for a fault

Bugs can be very elusive, so it is very helpful if you can determine conclusively which piece of code contains the bug.

In Eventz, the code is solely in the Functions y = F (Y,e). The only task of a Function is to format and publish an Event Record to memorialize an Event.

In Eventz, debugging is facilitated because a bug is always an incorrect Event Record, that was published by a Function. It is normally possible to debug that Function to determine the reason for the incorrect field in the ER.

The cause could be an undetected fault in an Archived ER.

5.4 Correcting Errors in Event Records

Eventz Event Record tuples are **immutable** (as are all tuples in the Python language), meaning that once a tuple is published it remains immutable in the Archive forever.

If an error is found, then a correcting ER can be published, with a UUID that links back to the original ER.

5.5 Our system is slow today

Slow system response is a common complaint. In Eventz systems this complaint can be assessed and investigated, because every ER is time-stamped. Then it is always possible to diagnose the slow response and make the necessary changes to its logic to restore fast response.

This aspect of Eventz helps to reduce the maintenance costs.

Slow Functions might be traced to intermittent connectivity or defective network components, for example.

It is even possible to create QA Functions that monitor and report on slow response conditions.

5.6 Patch Management & Active Directory

Patch Management and Active Directory are not supported today, but can be implemented via special purpose Functions created for these missions. These are examples of the vast scope of functionality that can be integrated into Eventnz systems.

PM Functions might probe the OS they are running in, and report what is running and installed, and any suspicious or unauthorized software.

AD Functions might report the current users and help to manage authentications, etc.